Subject: Date: Time:

ارزیابی درس

میان ترم ۵ نمره

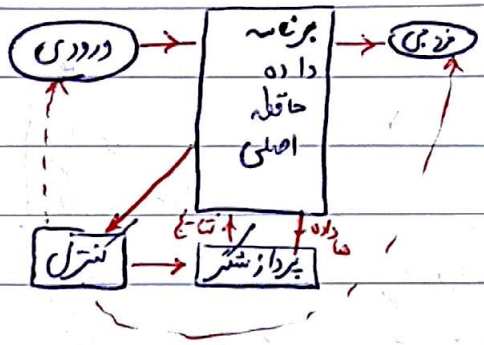
پایان ترم ۱۵ نمره

کوییز و تکلیف و پروژه ۵ نمره

منابع درس

۱- معماری کامپیوتر سید رضی ماندو

۲- Logical computer design Fundamental ماندو



طرح یک کامپیوتر ساده (مینا)

زبان ماشین - زبان اسمبلی - برنامه نویسی

پردسسور Arm

طرح بر اساس قطعات ؟

VLSI - LSI - MSI

روش الگوریتمی

جواب مسئله ترتیبی یا چارت ASM RTL

مدار منطقی

ترکیبی: آنالیز: عنصر حافظه و فیدبک نداشته باشد.

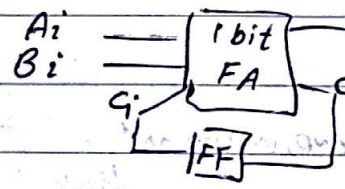
طراحی: خروجی در هر لحظه فقط تابع ورودی است حالا که ذاتا ترکیبی باشه راه حل ترکیبی داره.

ترتیبی: آنالیز

طراحی: حافظه ندارد ذاتا ترکیبی یا ذاتا ترتیبی

مدارهای ذاتا ترکیبی میتونن ترتیبی هم حل بشن.

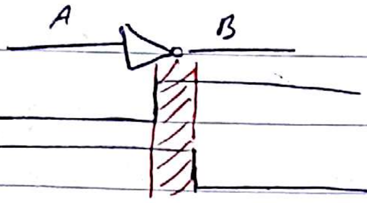
FA ذاتا ترکیبی است مثلا برای 8bit مجدول کارنو منطقی نیست یه راه استفاده از 8 FA یک بیتی است یا راه حل ترتیبی یه FA، 1bit داشته باشیم به شکل زیر



Ai و Bi و Si با شیفت رجیستر

Subject: Date: Time:

ترکیبی { ضبط جدول / رعایت پارامترهای الکتریکی قطعات / رعایت تأخیر



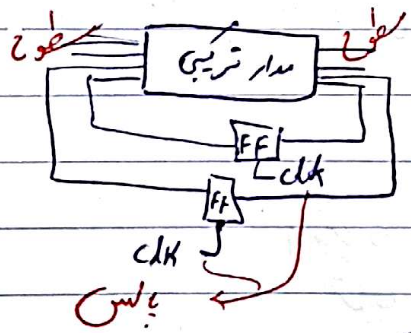
- مدار ترکیبی - خروجی هایش همیشه معتبر نیست.
تأخیر همه گیت های یکی نیست اما یک تأخیر ماکزیمم داریم.

مدل آسنکرون

مدار ترتیبی { آسنکرون / سنکرون { کلاک مد / پالس مد / کلاک کنترل شده

در مدار های بزرگ آسنکرون استفاده نمی شه چون اگر جنتشون ! ! باشن با هم نوسان می کنن ناپایدار می شن ← کلاک می دارن ← سنکرون

مدل سنکرون



به تأخیر مدار ترکیبی

برای این که خروجی معتبر شه باید تایم خطا از پریود clk کمتر باشه، $T_{delay} < T_{clk}$

$(1/T_{delay}) > f_{clk}$

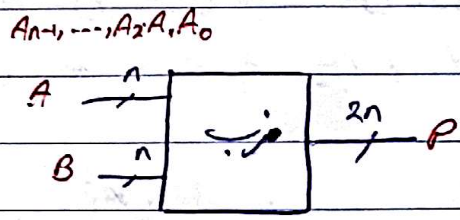
اطلاعات توی ۰ ها و ۱ ها باشه ← ورودی سطح level

" تغییر از ۰ به ۱ یا برعکسه ← ورودی پالس Pulse

اینکه بگیم یه لامپ چقد روشن بوده سطح

" " " " چند بار روشن شده پالس

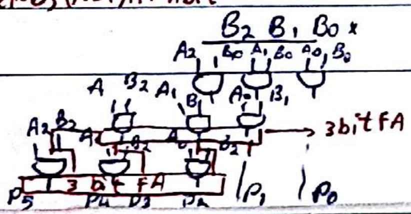
نمایش در چارت و RTL



روش ترکیبی

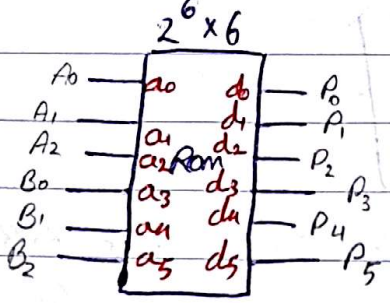
۱- روش کلاسیک جدول کارنو ... $n = 2, 3$

2- جمع و شیفت n^2 AND, $(N-1)$FA nbit $A_2 A_1 A_0$ $n = 8, 16$



Subject: Date: Time:

3. جدول ضرب ← ROM 2^{2n} کلمه $\times 2n$ bit



a_5	a_4	a_3	a_2	a_1	a_0	d_5	d_4	d_3	d_2	d_1	d_0
B_2	B_1	B_0	A_2	A_1	A_0	P_5	P_4	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0	0	0	0	0
⋮											
1	1	0	0	1	0	1	1	0	0	0	1
⋮											
63	1	1	1	1	1	1	0	1	1	0	1

$2^6 \times 6$ bit اطلاعات که به صورت 6bit، 6bit قابل دسترسی.

ازهمه بهتر داشتن n های بزرگ همان جمع و شیفت.

روش ترتیبی

4. جمع و شمارش

$$A \times B = \underbrace{A + A + \cdots + A}_{B} \quad \text{(P)}$$

1FA, 1CNT

بهترین روش ← 5. جمع و شیفت ترتیبی

از یک FA با شیفت استفاده می‌کنیم و n بار (FA(n-1)

1FA, 1CNT

برای bitn روش 4 حداکثر $2^n - 1$ بار می‌شمرد روش 5 حداکثر n بار سریعتر.

کد ضرب روش 5 با

ترتیب اجرا

1- نقطه شروع (قرار دادی)

2- تعاملی (قرار داد)

3- جملات کنترلی (به هم زدن توالی) (شرطی / غیر مشروط)

الگوریتم

جملات کنترلی

جملات غیر کنترلی - پردازشی

```
Read A, B
P = 0
L1, if (B == 0) stop
P = P + A
B = B - 1
goto L1
```

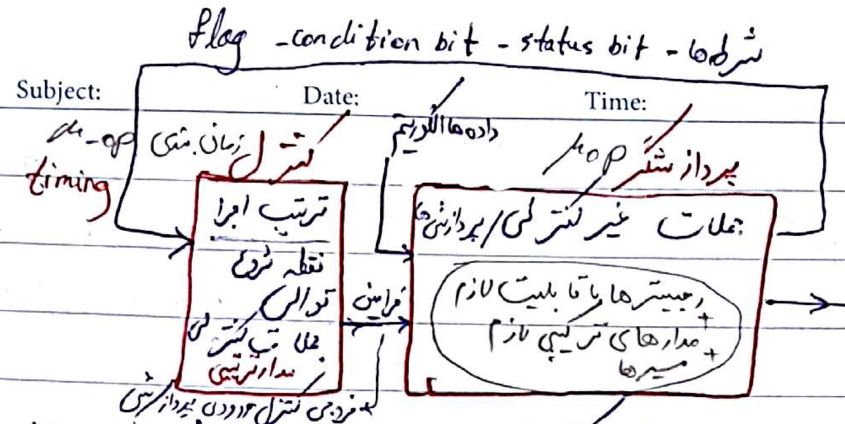
کنترلی شرطی

کنترلی غیر شرطی

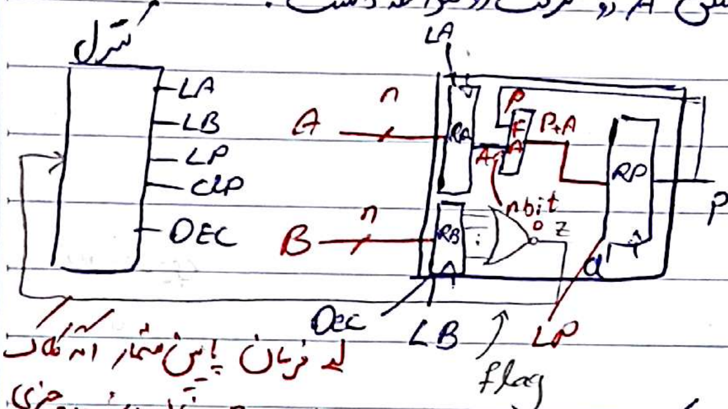
micro operations (پردازشگرها) هر خط از این برنامه یک ریز عملگر

Subject: Date: Time:

flag - condition bit - status bit - شرط‌ها



برای گرفتن A و B می‌تونیم از رجیستر یا RAM استفاده کنیم اینجا رجیستر بهتره چون حجمش کمه همچنین اطلاعات RAM کلمه به کلمه قابل دسترسیه (اگه بردی) از اطلاعاتش همزمان بخوایم دسترسی داشته باشیم باید باز از رجیستر استفاده کنیم چون نمی‌شه مثلن A رو گرفت رو هوا نگه داشت.



رجیستر

L	↑	عملکرد
0	↑	R←R
1	↑	R←I

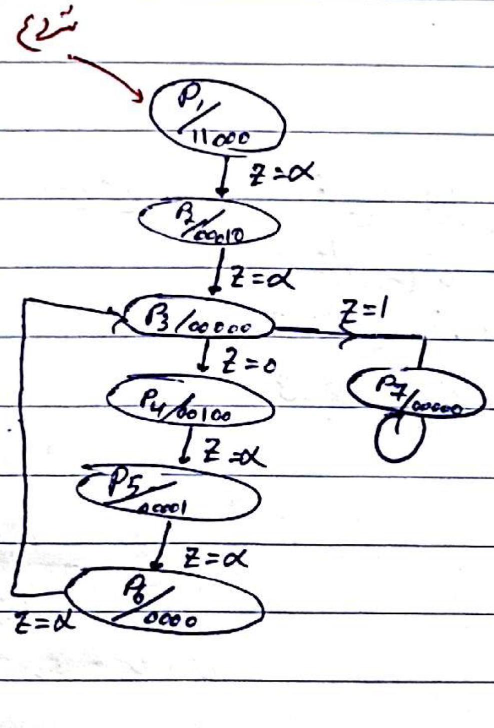
کلاک‌ها همه به هم وصلند CLK

این مدار ⑦ ها

لبه فرمان پایین متمایز اره کلاک بیاد و Dec فعال باشه رجیستری که توی رجیستر Load شده یکی کم می‌شه.

کلاک	LA	LB	LP	cl	DEC
1	1	1	0	0	0
2	0	0	0	1	0
3	0	0	0	0	0
4	0	0	1	0	0
5	0	0	0	0	1
6	0	0	0	0	0

مدار ترتیبی



μ-op : ریز عمل

با clk انجام می‌شود یک رجیستر مقدار می‌گیرد فرمان دارد. $L: R \leftarrow$ مقدار

مقدار تابعی است از ورودی های ثابت ها، مقدار خود رجیستر یا سایر رجیسترها

برنامه رو برحسب μ-op ها می‌نویسیم.

RTL

$T_1: RA \leftarrow A,\ R_B \leftarrow B$ — μ-op لود, goto T_2

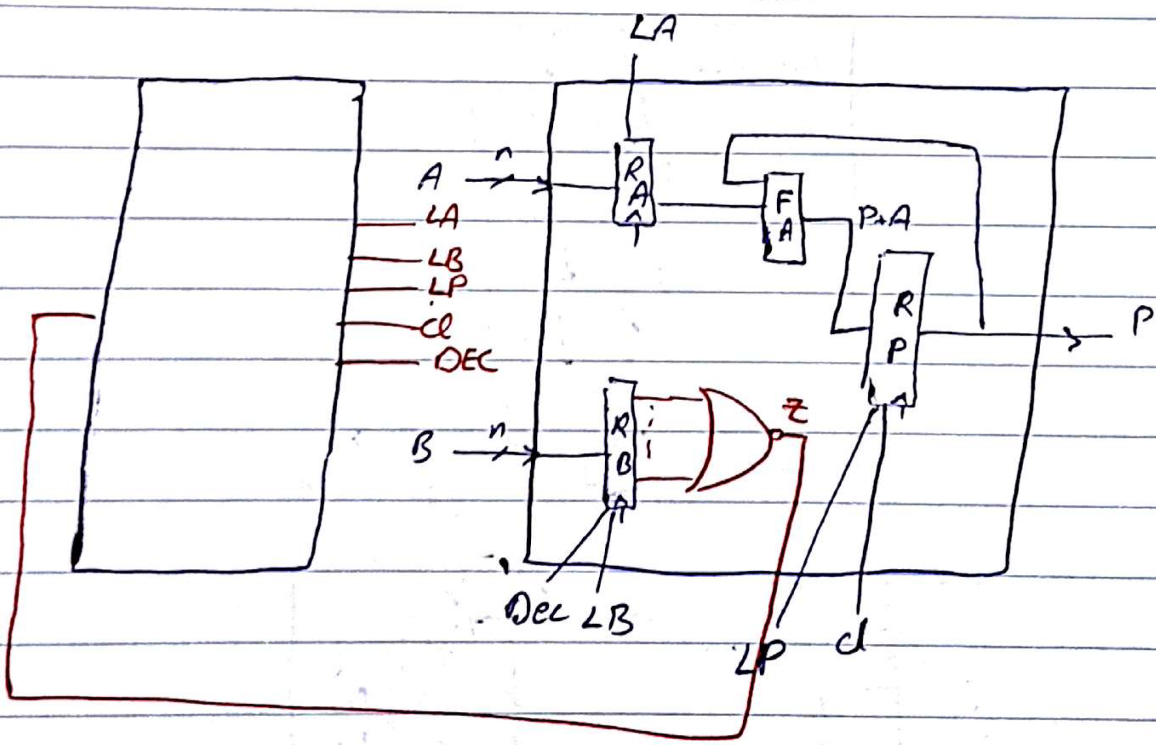
$T_2: RP \leftarrow 0$, goto T_3 → cl و μ-op

T_3: if (z==1) goto T_3 else goto T_4

$T_4: RP \leftarrow RP + RA$ goto T_5 OR — μ-op

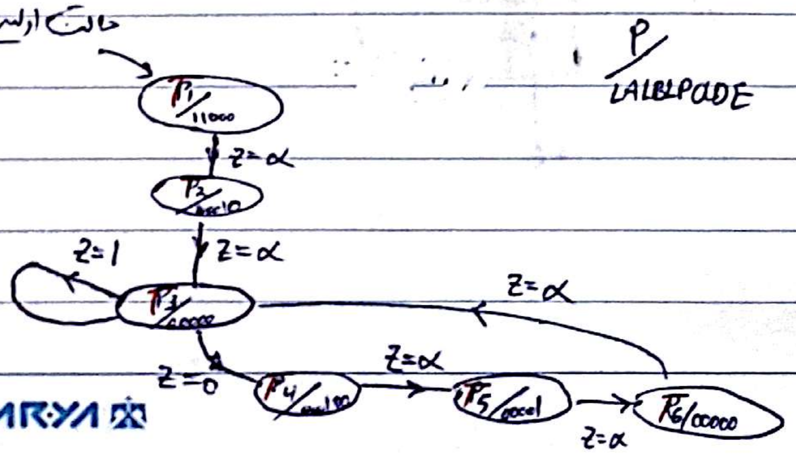
$T_5\ R_B \leftarrow R_B - 1$ goto T_6 → Dec و μ-op

T_6: goto T_3



نمودار حالت بخش کنترلی (مدار ترتیبی)

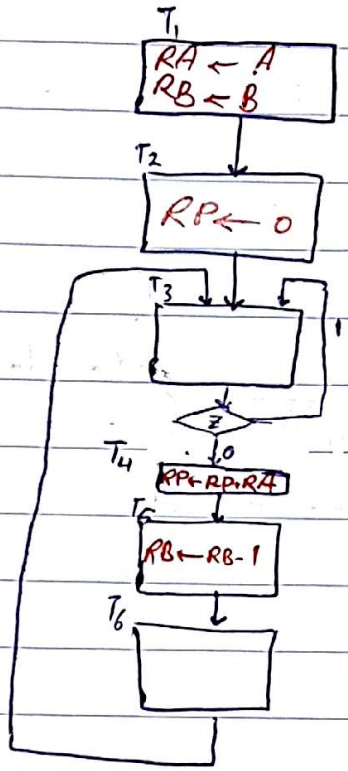
P / LALBLPCLDE

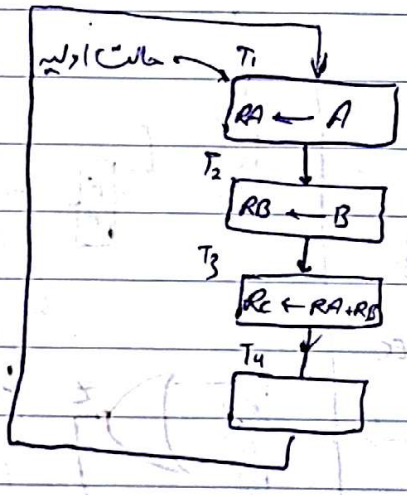


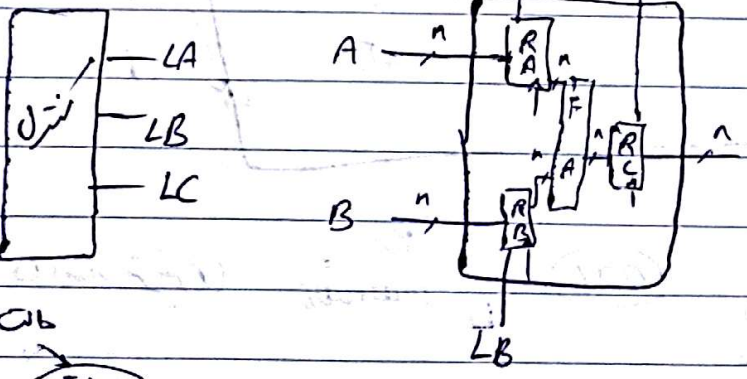
تایمینگ ها حالت های واحد کنترل هستند. ⇐

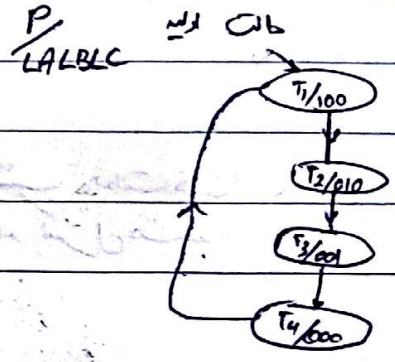
Subject: Date: Time:

SM chart

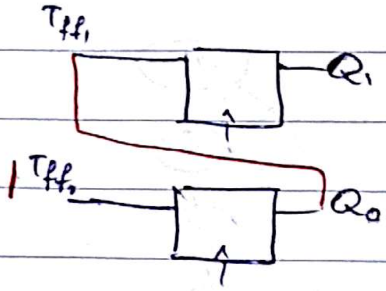








Subject: Date: Time:



حالت فعلی		حالت بعدی
00	T_1	T_2
01	T_2	T_3
10	T_3	T_4
11	T_4	T_1

حالت فعلی Q_1^t	Q_0^t	حالت بعدی Q_1^{t+1}	Q_0^{t+1}	T_{ff_1}	T_{ff_0}	LA	LB	LC	T_1	T_2	T_3	T_4
0	0	0	1	0	1	1	0	0	1	0	0	0
0	1	1	0	1	1	0	1	0	0	1	0	0
1	0	1	1	0	1	0	0	1	0	0	1	0
1	1	0	0	1	1	0	0	0	0	0	0	1

جدول حالت — جدول تحریک — جدول خروجی

$T_{ff_1} = Q_0$

$T_{ff_0} = 1$

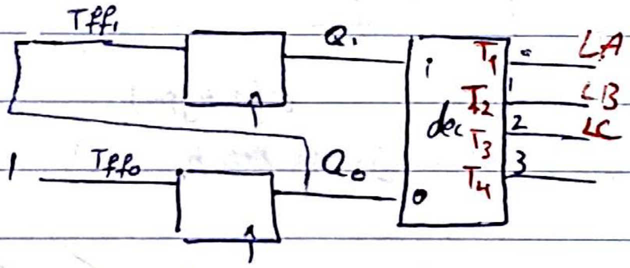
$LA = Q_1' Q_0'$

$LB = Q_1' Q_0$

$LC = Q_1 Q_0'$

Counter-decode

این مثال یک شمارنده است: $00 \to 01 \to 10 \to 11$

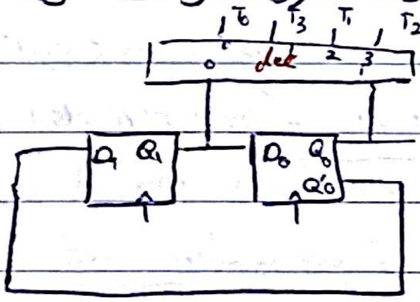


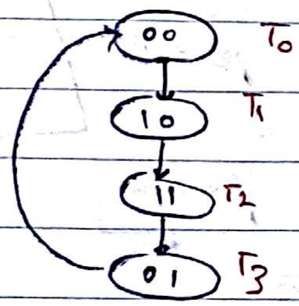
که این تابعی از تایمینگ‌ها و ورودی‌ها هستند.

اگه ترتیب T_2 و T_4 هم داشتیم $RB \leftarrow B$ ، LB می‌شد OR T_2 و T_4

ورودی مشترک‌ها با کلاک کار می‌کنند ورودی آسنکرون هر وقت فعال شد عمل می‌کند.

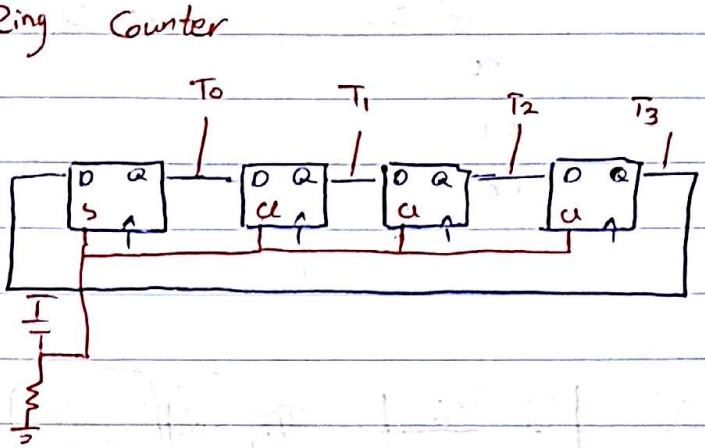
Johnson counter

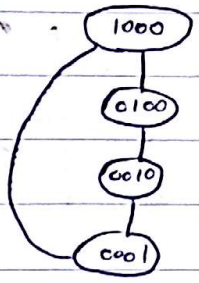


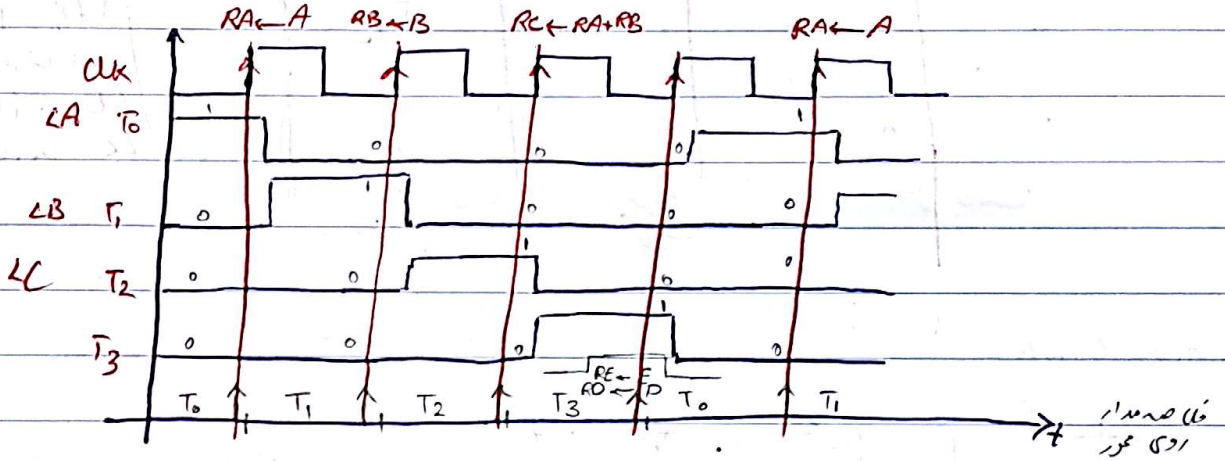


Subject: Date: Time:

Ring Counter



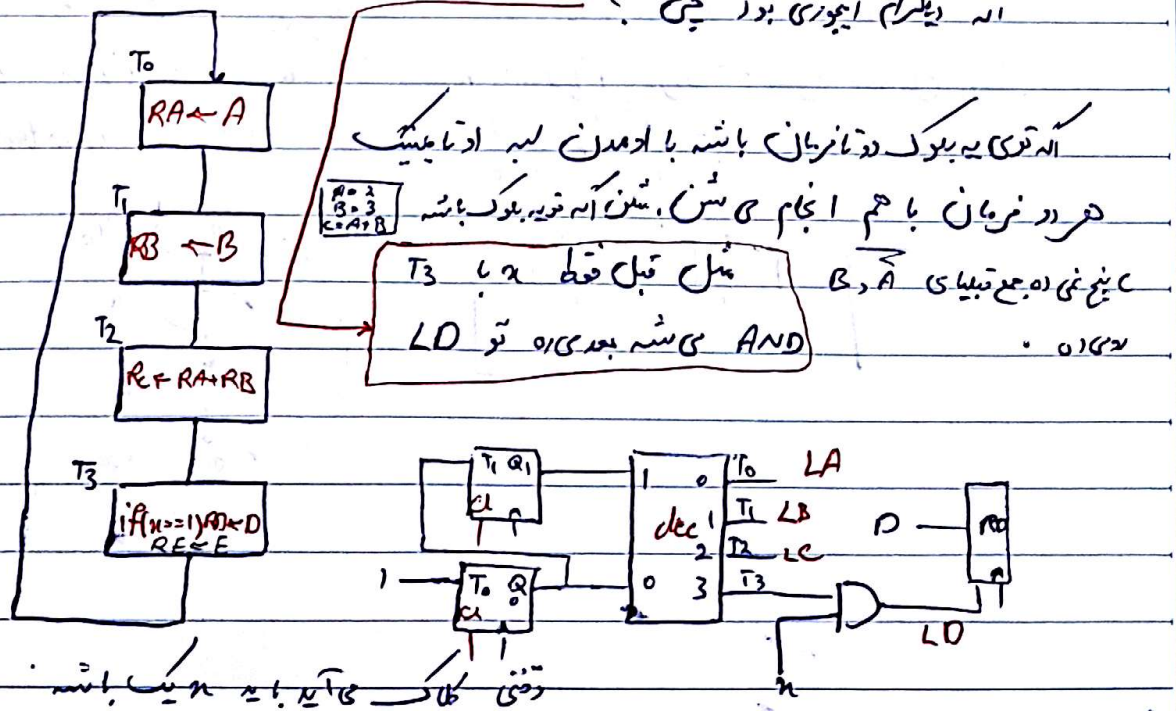




(edge-trigger)

این latch ها چون دقت پشت من به master ـ slave نیاز دارد.

آمه دیاگرام اینجوری بود چی ؟

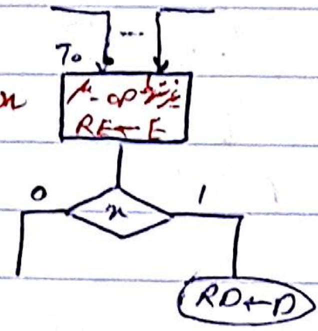


ARYA

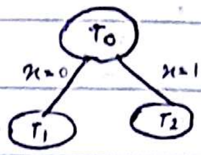
Subject: Date: Time:

S M chart

state box

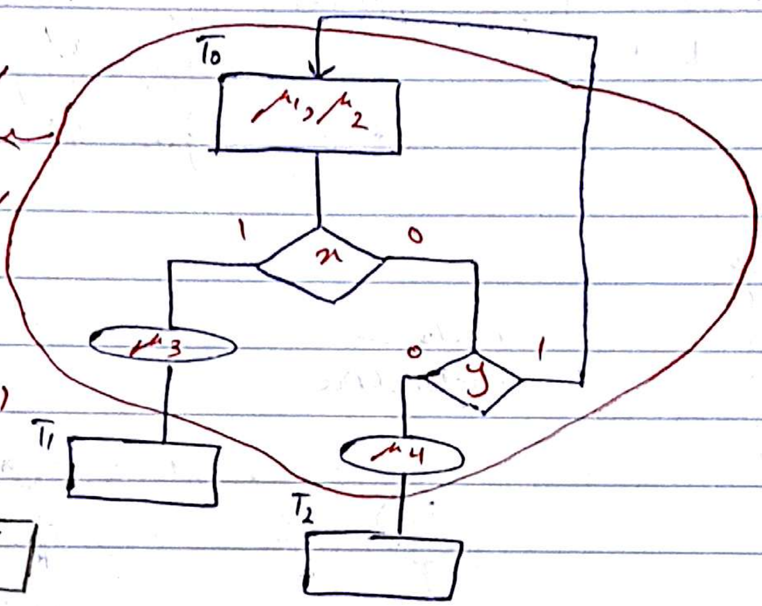


decision box



← یک بلوک ASM که توی یک زمان بهم انجام می‌شه درمان حالتش توی بازه T_0 (یعنی T_0) باید کلاک بعدش باهم انجام می‌شه

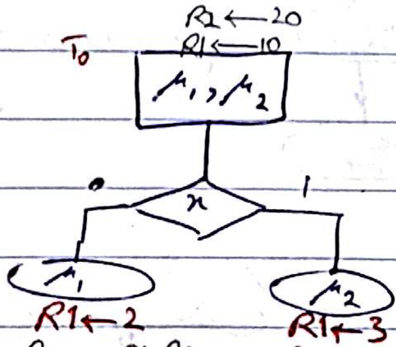
ترتیبی نیست



T_0 : μ_1 , μ_2 ; if (x) { μ_3 } else if (y') { μ_4 } else goto T_0

goto T_1 goto T_2

این حالت پایین غلطه چون توی یک تایمینگ نمی‌شه یه op ـ م دوبار انجام بشه.



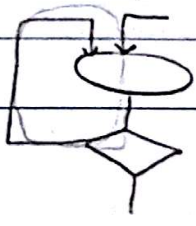
داخل decision box ها ورودی ها

اما اگه ترتیبی باشه درسته.

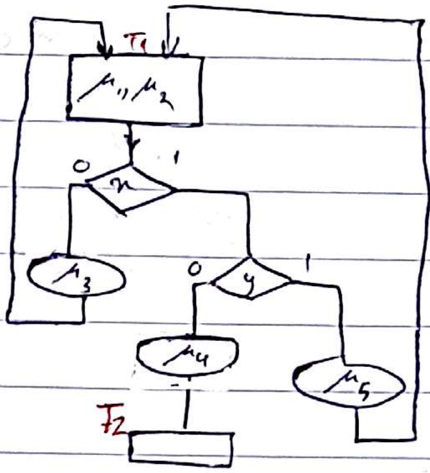
با مدادا R_3 مقادیر قبلی R1 و R2 رو حساب می‌کنه چون ترتیبی نیست

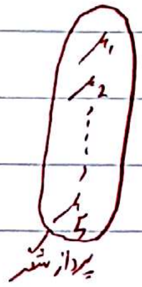
همه باهم حساب می‌شن

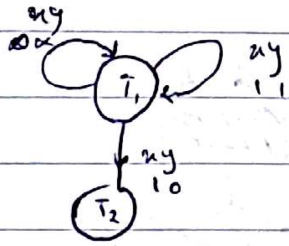
بعد با کلاک باید بره توی حالت نباید توی loop بیوفته



Subject: Date: Time:

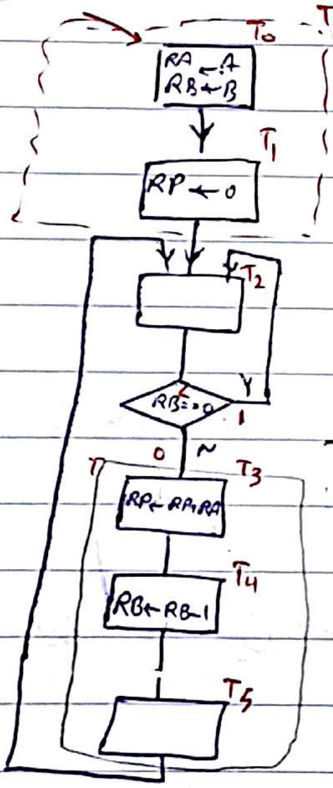




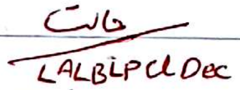


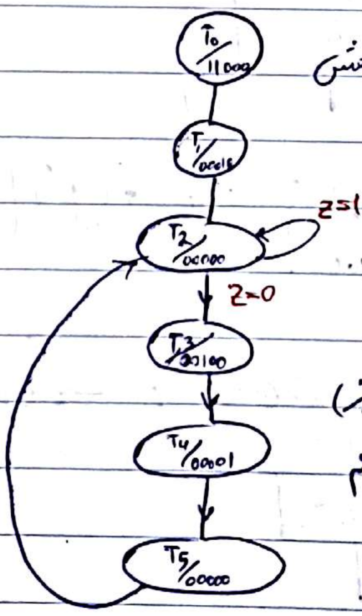
$T_1: \mu_1, \mu_2$, if (x) if (y) { μ_5, goto T_1 },
else { μ_4, goto T_2 }
else { μ_3, goto T_1 }

حالت اولیه



کاهش حالت

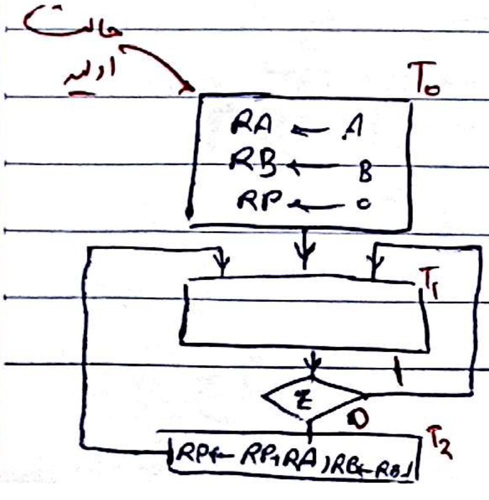




اینجا فقط خروجی T_5 و T_2 یکیه ولی به حالت یکسان نرفتن در نتیجه کاهش حالت نداریم.

اما می‌شه یه کار این کرد مثلن بریم تو الگوریتم ببینیم می‌شه RA, RB, RP با هم مقداراشون کار کنند هر دو بذاریم توی یک تایمینگ (آپدیت) یا مثلن ببینیم می‌شه همه مدادا رو با هم انجام داد یا نه.

حالت اولیه



ASM به شکل زیر در می‌آید:

اما دیگه T_1 رو نمیشه حذف کرد

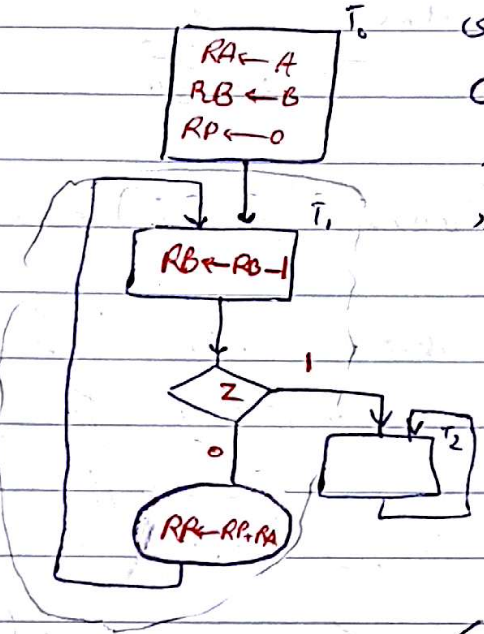
اگه حذف شه بعد چک Z به یه حالت نداره

بعدم بلاک اول رو Z می‌افتن توی یه دونه timing

دیگه نمیتونه B لود شده رو چک کنه B قبلی رو چک می‌کنه.

Subject: Date: Time:

می‌شه چارت رو به شکل زیر هم کشید و حاصل $P=A\times B$ هم درسته.

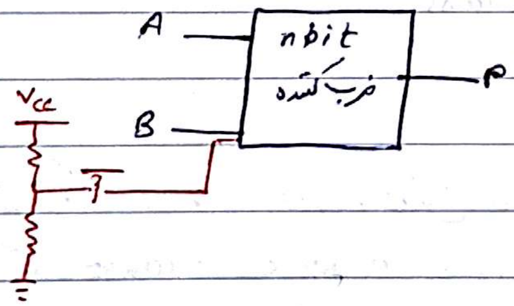


درسته چون ازنایی که توی کار هستن همه باهم توی یک timing انجام می‌شه تنها فرقی که داره قبلی وقتی کار تموم می‌شه $B=0$ ولی اینجا وقتی تموم می‌شه $B=-1$ و اما یک بار $RP\leftarrow RP+RA$ رو نمی‌شه آورد بالا چون یه دونه بیشتر می‌شمره.

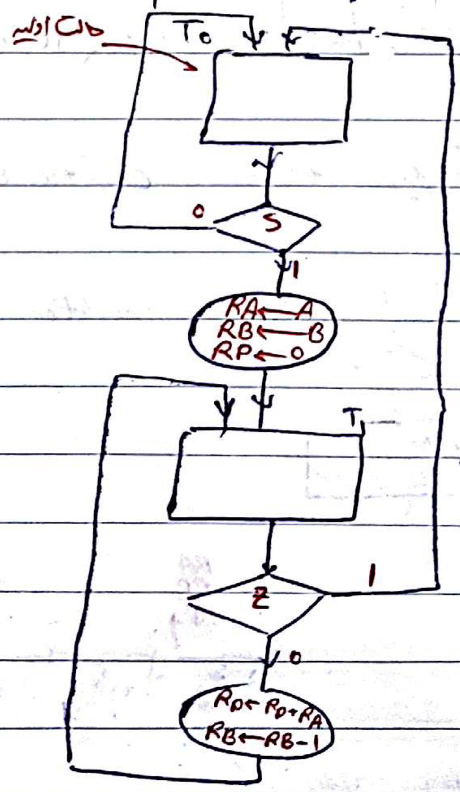
با چارت قبلی می‌ریم جلو

ترکیبی max, min

طرز کار این مدار ضرب این طوریه که مدار و خاموش می‌کنیم توی پایه‌های A و B اون عددی که می‌خوایم وصل سازیم بعد مدار رو دوباره روشن می‌کنیم که بره به حالت اولیه. برای روشن خاموش کردن یه کلید اضافه می‌کنیم.



حالا چارت رو می‌کشیم

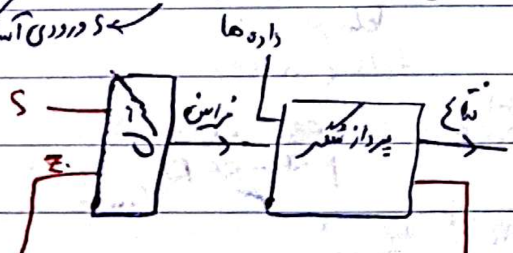


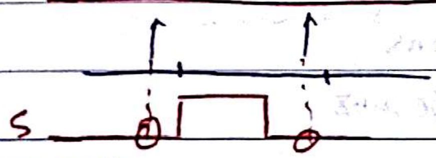
ما ورودی S و Z رو داریم.

می‌تونه یه شکل زیر باشه ولی Z نه چون Z به RB ربط داره RB هم حتماً با کلاک تغییر می‌کنه

Z ← ورودی سنکرون

S ← ورودی آسنکرون





Z از کلاک بعدی می‌ره ۰

Subject: Date: Time:

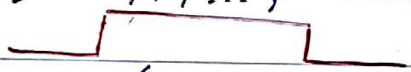
برای دیدن ورودی های آسنکرون باید دوره کلاک و ورودی رو هم بپوشونن که ببینه مدار ورودی رو

$T_{clk} < T_{min} = 1ms$

$\Rightarrow f_{clk} > 1KHz$ → تعلیق کلاک پالس رو می‌بینه.

برای داشتن فرکانس مینیمم ← ورودی آسنکرون یا حافظه دینامیک

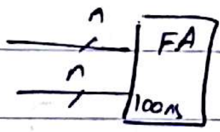
یا مثلا کلاک 1MHz باشه و s، 1KHz هزار بار کلاک میاد روی s هزار بار ضرب می‌کنه



چارت کشیم که توی یه پالس s یه بار ضرب کنه بعدش ضرب نکنه تا s دوباره یک شه. یعنی هر بار که s میاد یه بار ضرب کنه هر بار کلاک یک دید کلاک قبلی صفر دیده بود.

فرکانس ماکزیمم (تاخیر مدار ترکیبی) تواند با چند پریود رعایت شود.

به تاخیرها مربوطه فرکانس ماکزیمم و همچنین به تاخیر غالب مثلن توی مثال خودمون گیت AND بیشترین تاخیر رو داره نسبت به Cl و Dec داینه

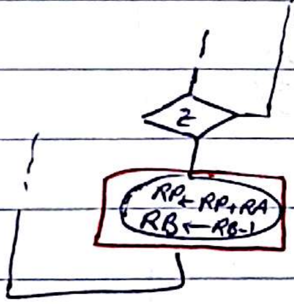


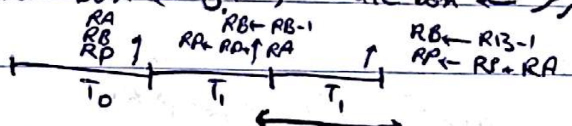
$T_{clk} > T_{max} = 100ns$

$\Rightarrow f_{clk} < 10MHz$

اگه اونی که FA و AND هستن جای condition box و state box بذاریم تاخیر بیشتر می‌شه دگه 10MHz نیست 20MHz ه.

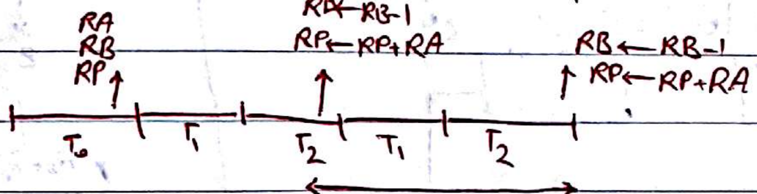
قرمز ← state box ، آبی ← condition box





$T_{clk} > 100ns$

$f_{clk} < 10MHz$



$2T_{clk} > 100ns$

برای اینکه بیشتر 30MHz بشه 3 تا state box بذاریم دگه

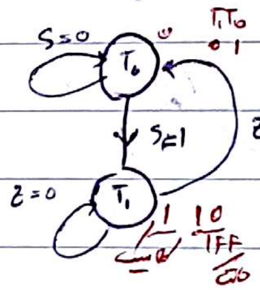
$\rightarrow F_{clk} < 20MHz$

زیر Z

اما جهت این مدارها ضرب رو با سرعت یکسان انجام میدن حاصل ضرب رو وقتی یه نمونه میدن

Subject: Date: Time:

طرح کنترل:
- روش کلاسیک
- حالت IFF
- ROM - Reg
- PLA - Reg
- MUX - Register

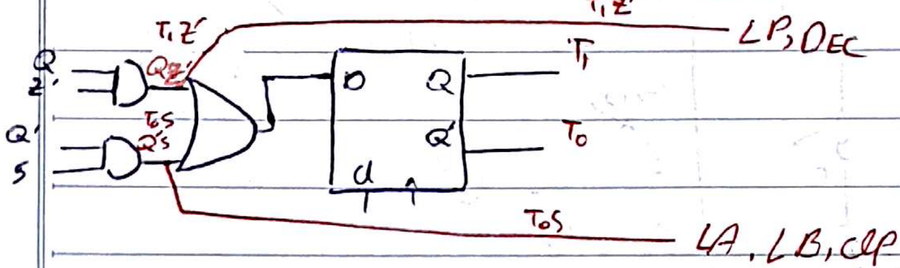


حالت فعلی Q^t		ورودی‌ها SZ: 00	01	11	10
T_0	0	0	0	1	1
T_1	1	1	0	0	1

T_1Z'

روش کلاسیک

$D^t = Q^{t+1} = QZ' + Q'S$



روش IFF حالت

دو حالت ← D-FF 2

حالت اولیه متوجه روشن شدن T_0

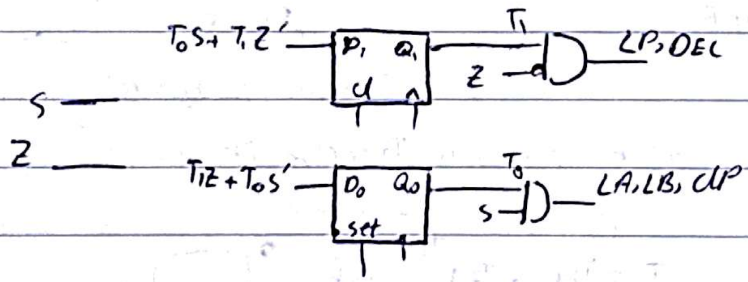
حالا برای D ها کی می‌خوایم توی T_0 باشیم (کی T_0 یک باشه؟)

توی T_0 باشیم S هم صفر باشه

توی T_1 باشیم Z هم 1 باشه

کی می‌خوایم توی T_1 باشیم؟

T_0 و S 1 باشن یا توی T_1 باشیم و Z صفر باشه.



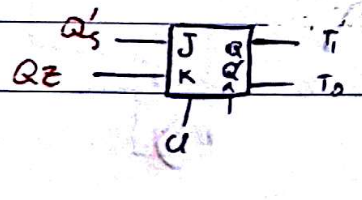
حالا با JK-FF

هروقت بخوایم خروجی 1 بشه ← J=1

" " " Q' ← K=1

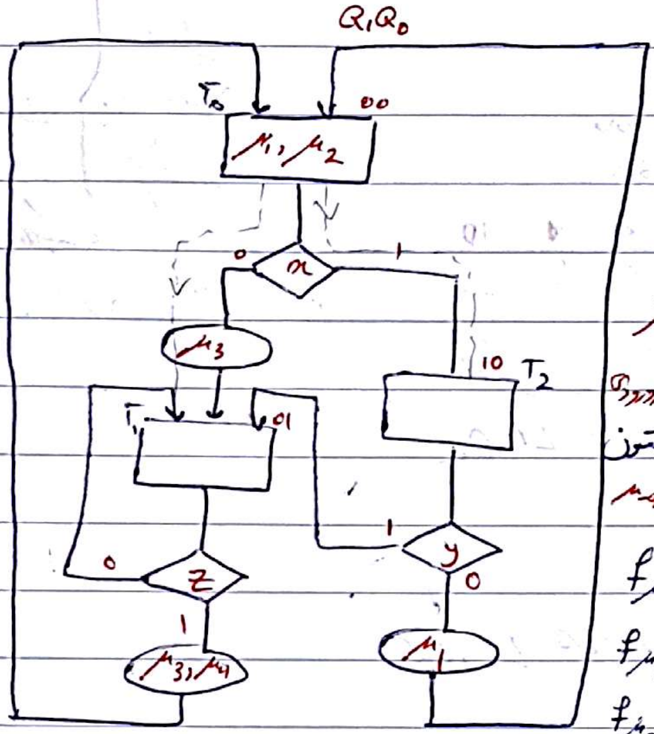
بعد برای ثابت موندن هم لازم نیست چیزی اعمال کنیم.

```
Q'                          Q←0
T0: if (S') goto T0
    else {RA←A, RB←B, RP←0}, goto T1}
Q                       T0S
T1: if (Z') {RP←RP+RA, RB←RB-1, goto T1}
    else goto T0        T1Z'        Q←1
        Q←0
```



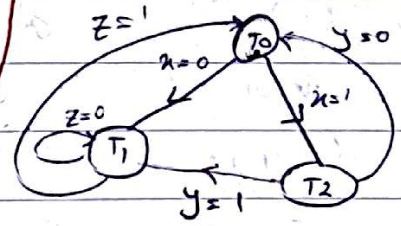
Subject: Date: Time:

روش ROM-Reg



μ_1, μ_2, μ_3, μ_4

پردازشگر



دو FF → سیر دو حالت

اینجا با روش کلاسیک جدول ۴ سطر و ۸ ستون داره

زمان مهم

$f_{\mu_1} = T_0 + T_2 y'$

$f_{\mu_2} = T_0$

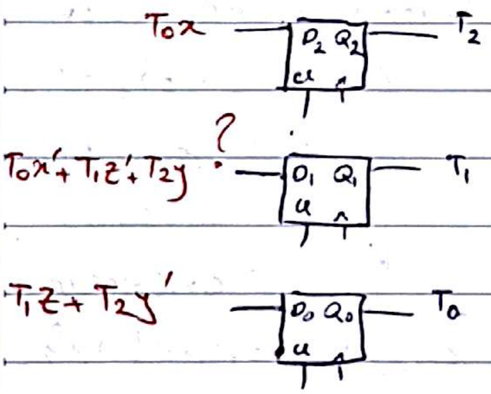
$f_{\mu_3} = T_0 x' + T_1 z$

$f_{\mu_4} = T_1 z$

ورودی ها رو می خونیم

: DFF

JK-FF 1FF حالت



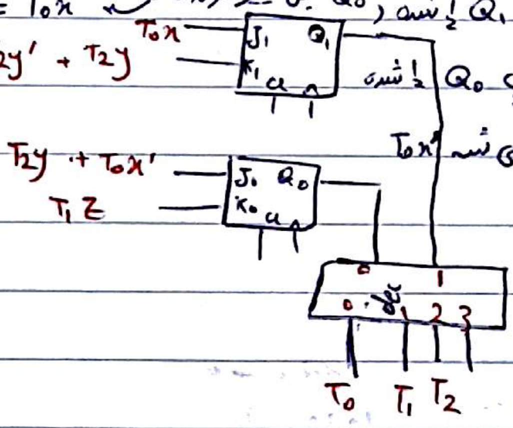
برای هر سیر تغییر حالت ها رو بررسی می کنیم با توجه به دادن شرط‌ها رو می ذاریم توی J یا K توی

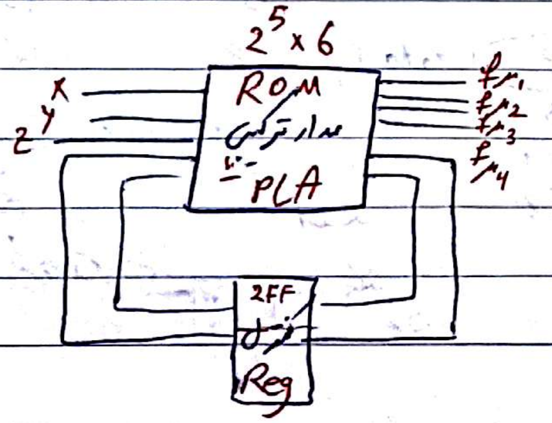
مسیر راست Q_1 یک شده و Q_0 هم تغییر نمونده $J_1 = T_0 x$

$T_2 = T_2 y' + T_2 y$

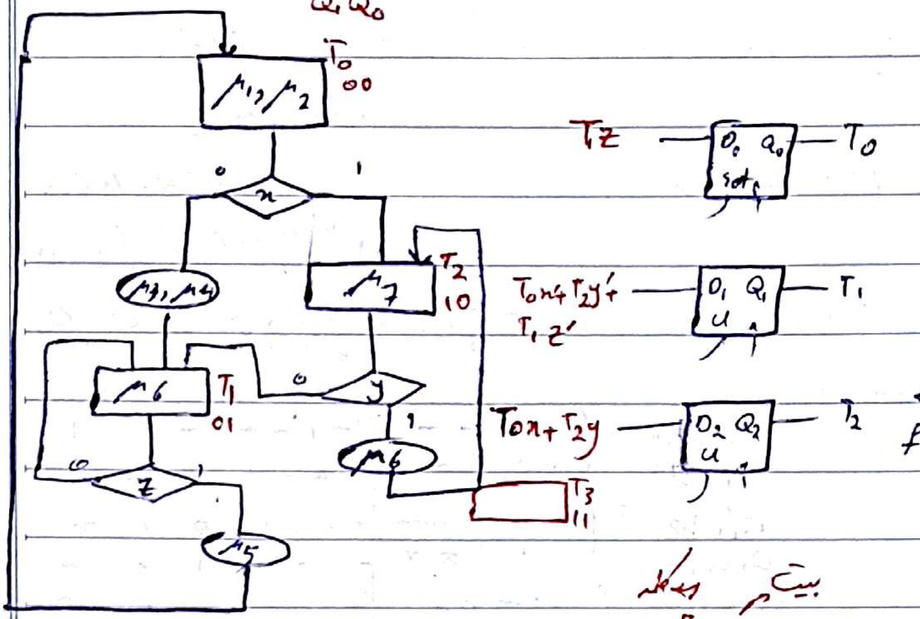
مسیر چپ Q_0 یک شده

پس K_0 می شه $T_0 x'$





Subject: Date: Time:



روش کلاسیک مشخصه

روش FF حالت

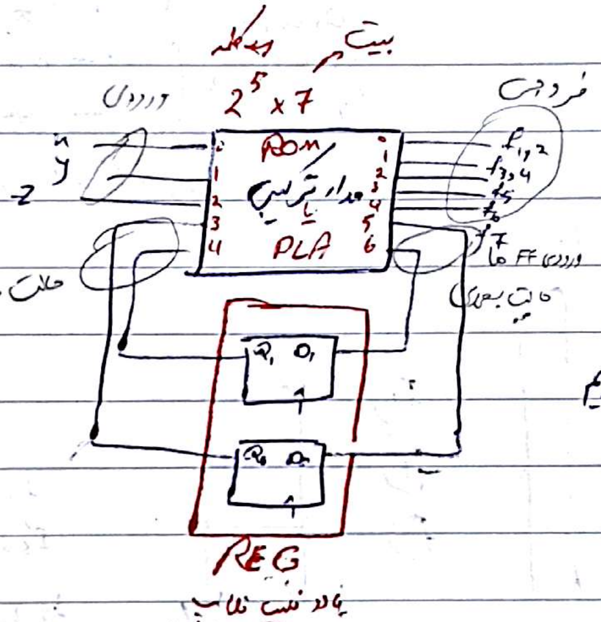
$f_{1,2} = T_0$

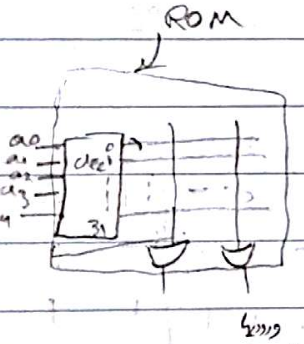
$f_{3,4} = T_0x'$

$f_5 = T_1Z$

$f_6 = T_2y + T_1$

$f_7 = T_2$





حالا به جای مدار ترکیبی

PLA یا ROM می‌ذاریم

یک ROM به تعداد 2^5 AND داره و به تعداد خروجیها OR داره

جدول برنامه ریزی بعدی ROM خروجیها

	حالت فعلی		ورودی‌ها									
	Q_1	Q_0	Z	Y	x	Q_1^{t+1} (D_1)	Q_0^{t+1} (D_0)	f_7	f_6	f_5	$f_{3,4}$	$f_{1,2}$
	a_4	a_3	a_2	a_1	a_0	d_6	d_5	d_4	d_3	d_2	d_1	d_0
0	0	0	0	0	0	0	1	0	0	0	1	1
5	0	0	1	0	1	1	0	0	0	0	0	1
10	0	1	0	1	0	0	1	0	1	0	0	0
11	0	1	0	1	1	0	1	0	1	0	0	0
24	1	1	0	0	0							
31	1	1	1	1	1	1	1	0	0	0	0	0

کلمات $d_0 d_1 d_2 d_3 d_4 d_5 d_6$

با در حالت 00 بودن و $x, y, z = 0$ میریم به حالت T_1 ← 01

و μ_1 و μ_2 و μ_3 و μ_4 باید انجام بشه. اینارو ۱ میذاریم.

سر کلاک بعدی که بیاد μ_7 انجام میشه. اینجا انجام نمیشه.

حالت استفاده به خاطر نویز و مدار از مسیر خودش خارج می‌کنه.

24 اینکه از اینجا بره به مسئله ربط داره ممکنه تو همون حالت بمونه خوب باشه

31 شاید یه آژیر بزنه یا برگرده به یکی از حالت‌ها.

که این مثال هیچجا نمیره

حال چند کلمه متمایز داره ROM؟ هر مسیر یه کلمه متمایز ← 6 مسیر ← حداکثر 6 کلمه متمایز

8 تا کلمه استفاده نشده هم داریم از 24 تا 31 که $Q_1 = Q_0 = 1$

اگه یه مدار 5 حالته و 4 ورودی داشته باشیم ← $3 \times 16 = 48$ حالت استفاده نشده داریم

ترکیب ورودیها که حالت استفاده نشده

Subject: Date: Time:

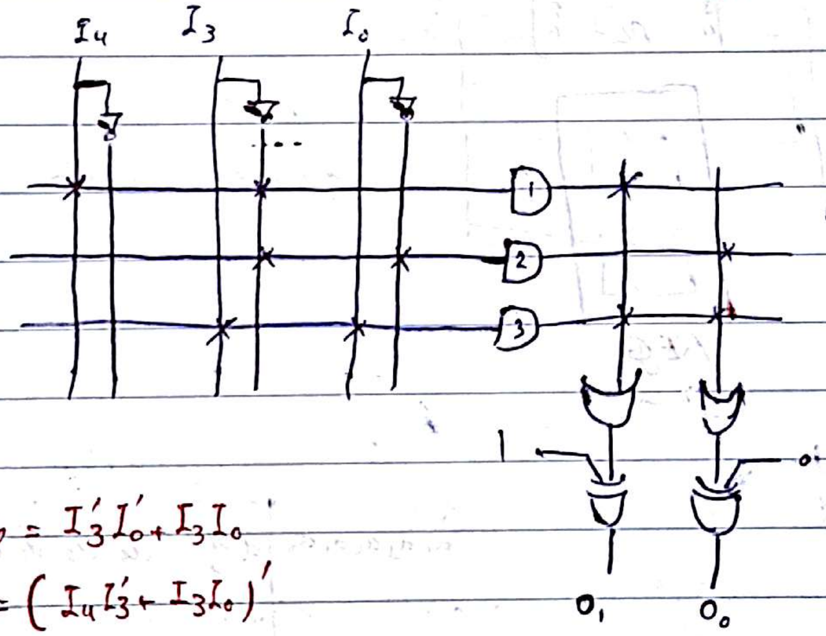
اما با PLA به تعداد مسیرها AND می‌خواهیم و به تعداد خروجی‌ها OR پس با 6 AND و 7 OR
می‌سازیم مدار رو.

PLA-REG — جدول حالت خلاصه شده.

	Q_1 Q_0 z y x	Q_1^{t+1} Q_0^{t+1} f_7 f_6 f_5 $f_{4,3}$ $f_{1,2}$
	I_4 I_3 I_2 I_1 I_0	O_6 O_5 O_4 O_3 O_2 O_1 O_0
$Q_1'Q_0'x'$ T_0	0 0 – – 0	1 1 0 0 0 1 1
$Q_1'Q_0'x$	0 0 – – 1	1 0 0 0 0 0 1
$Q_1'Q_0z'$ T_1	0 1 0 – –	0 1 0 1 0 0 0
$Q_1'Q_0z$	0 1 1 – –	0 0 0 1 1 0 0
$Q_1Q_0'y'$ T_2	1 0 – 0 –	0 1 1 0 0 0 0
$Q_1Q_0'y$	1 0 – 1 –	1 0 1 1 0 0 0
		T T T T T T T

از همین جا می‌شه خروجی‌ها k-map هارو هم مستقیماً نوشت.

جدول PLA (با ترتیب)



خروجی PLA ←

	I_4	I_3	I_0	Q_1	Q_0
I_4I_3'	1	0	–	1	–
$I_3'I_0'$	–	0	0	–	1
I_3I_0	–	1	1	1	1
				C	T

$Q_0 = I_3'I_0' + I_3I_0$

$Q_1 = (I_4I_3' + I_3I_0)'$

توی ROM و PLA به 2FF به حالت استفاده نشده.

اینجا اگر بره به حالت استفاده نشده چون براش AND ای نساختیم (برای 11 ، AND نداریم)

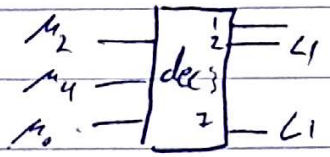
در نتیجه هیچی نمی‌شه صفر میده به حالت صفرصفر همه ورودی‌ها هم صفرن.

اگه به جای Q_1^{t+1} از C اش استفاده می‌کردیم و ... تو حالت استفاده نشده بیشتر می‌ره به 10 یا ورودی‌های صفر
(به ...)

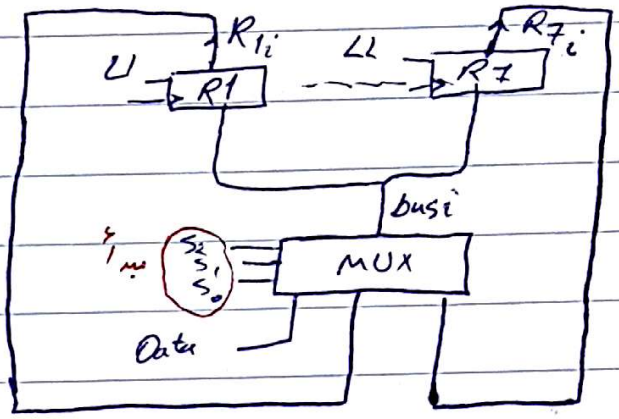
Subject: Date: Time:

با باس { با MUX
با خروجی سه حالته

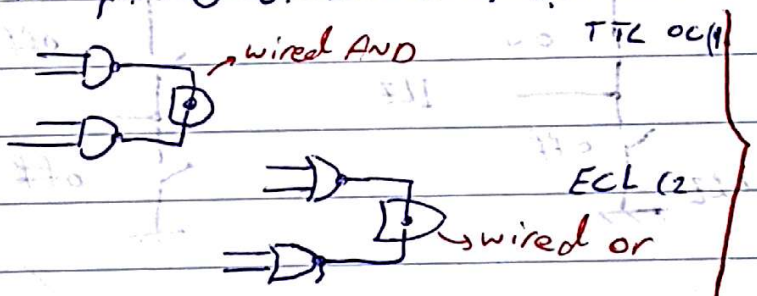
خروجی { OLZ
ILZ
HZ شناور



$i = 0 \_ (n-1)$

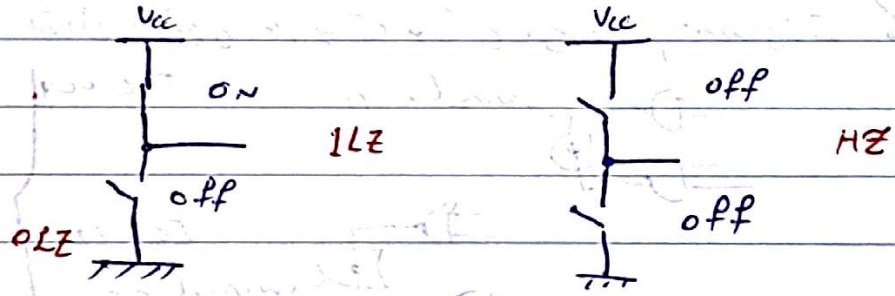


* اتصال دو خروجی مجاز نیست زیرا بی معنی و بهم صدمه می زنند. مگر در موارد استثناء



Subject: Date: Time:

خروجی سه حالته به جای MUX

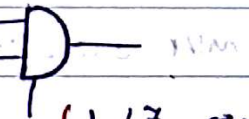


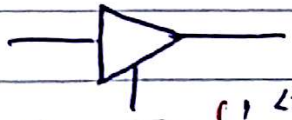
دو تا LZ رو نمی شه بهم وصل کرد

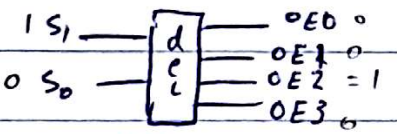
اما اگه 0 تا HZ با یه LZ به هم وصل

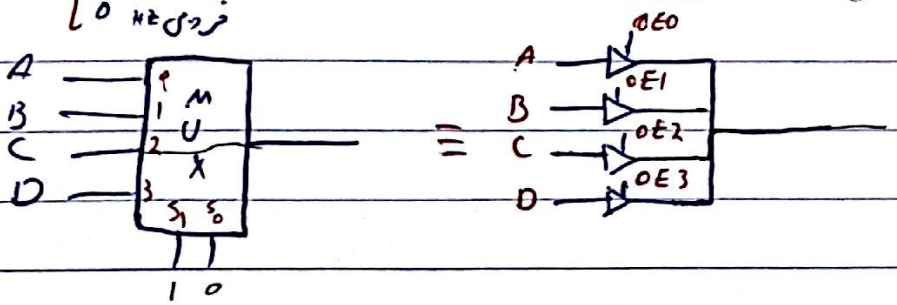
شن ← خروجی ۱ LZ تعیین

می شه چون HZ شناوره (خروجیش قطعه)



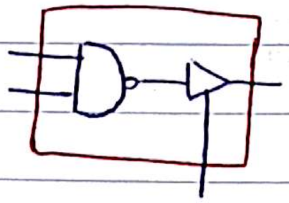




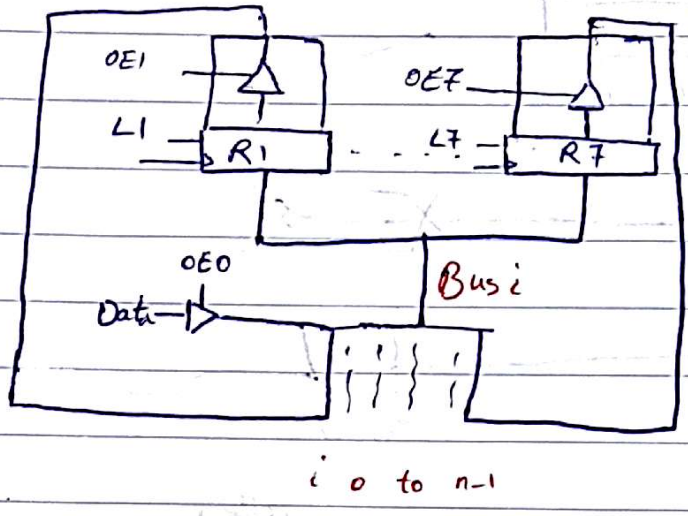


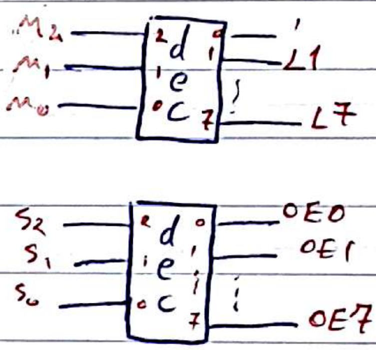
Subject: Date: Time:

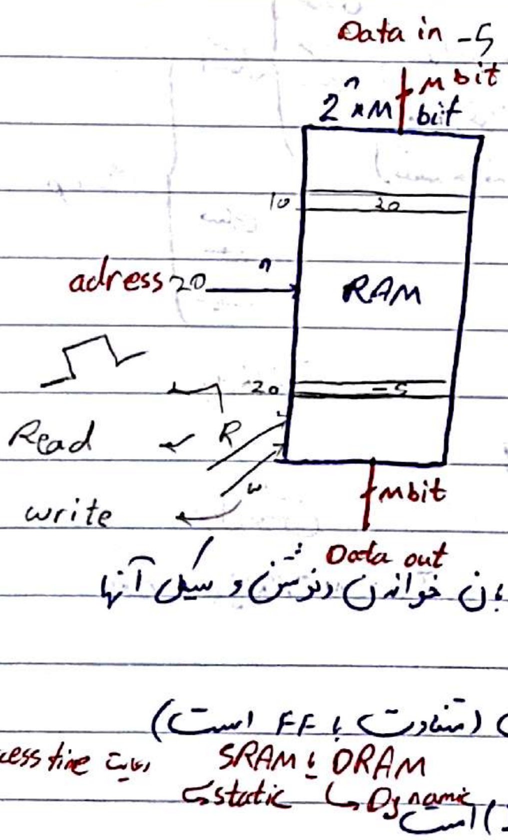
حالا هر چیزی رو می‌تونیم سه حالته کنیم ← مثلاً NAND سه حالته

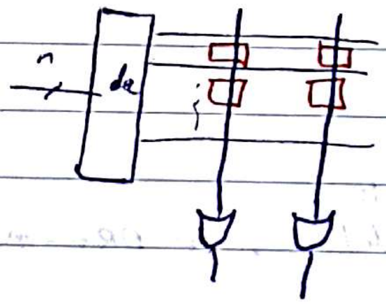


حالا مدار باس با خروجی سه حالته









ورودی‌ها، خروجی‌ها، زمان خواندن، نوشتن و سیکل آنها

سلول حافظه است (متفاوت با FF است)

اتصال حافظه با RAM

رعایت Access time   SRAM و DRAM

static و Dynamic

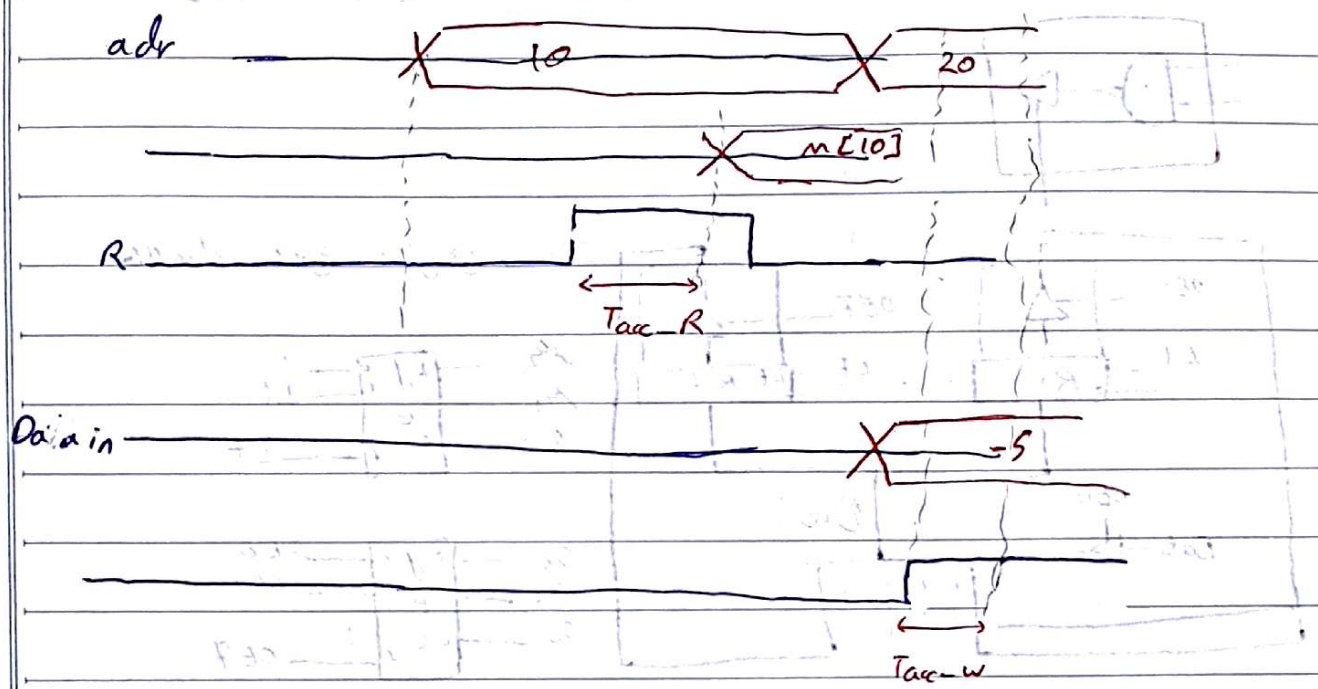
نقشه حافظه و (I/O) است

عرض پالس مهم باید یه حداقلی باشه تا خروجی معتبر باشه.

اینو دیگه master-slave نیست از slave اطلاعات بگیریم همزمان رو هم master تحریک کنیم تا به هم نه ارد

باید بشن رعایت Access time مهم

Subject: Date: Time:



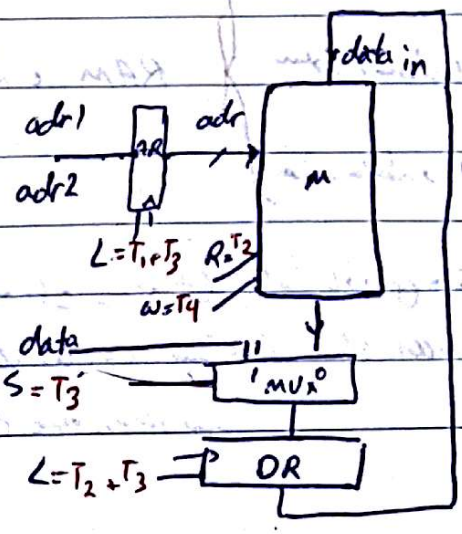
- انتقال
  - موازی
  - سری
  - بایتی
  - سیگنال حافظه و زمان دسترسی
  - سیگنال خواندن و نوشتن
  - نقشه حافظه و I/O
- عملیات
  - حسابی
  - منطقی
  - شیفت

سیکل خواندن

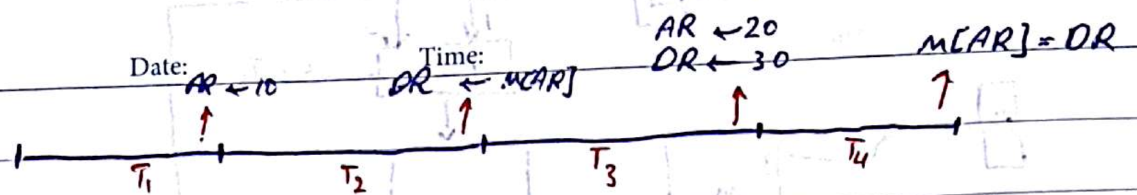
$T_1: AR \leftarrow adr1$ (10), $T_2: OR \leftarrow M[AR]$

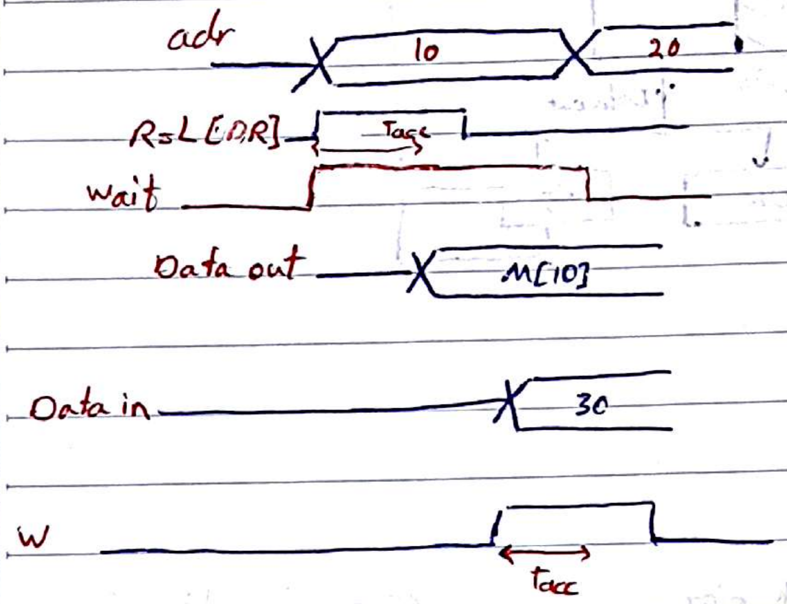
سیکل نوشتن

$T_3: AR \leftarrow adr2$ (20), $OR \leftarrow data$ (30), $T_4: M[AR] \leftarrow OR$



Subject: Date: Time:





اگه تاخیر داشت RAM لازم بود یه تایمینگ خالی بونه که باید حالت wait اضافه کنیم

رو تغییر می‌دیم T_2 رو هم اینجوری می‌ذاریم،

$T: DR \leftarrow M[AR]$, {if wait=1} goto T_2 , else goto T_3

$T_1: R_1 \leftarrow 10,\ R_2 \leftarrow 20,\ R_3 \leftarrow 30,\ R_4 \leftarrow 40$

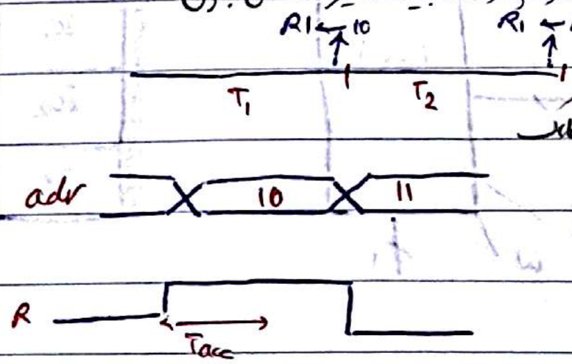
$T_2: DR \leftarrow M[R_1]$ (10), $R1 \leftarrow R1+1$ (30), if(wait) goto T_2, else {goto T_3} (20)

$T_3: M[R_3] \leftarrow R_2$ (30, 20), $R_3 \leftarrow R_3+1$ (31), $R_2 \leftarrow R_2-1$ (19) → بعد از کلاک

$T_4: R_5 \leftarrow M[R_4]$ (40), ~~$R_5 \leftarrow R_5-1$~~, $R_4 \leftarrow R_4+1$ (41, 40)

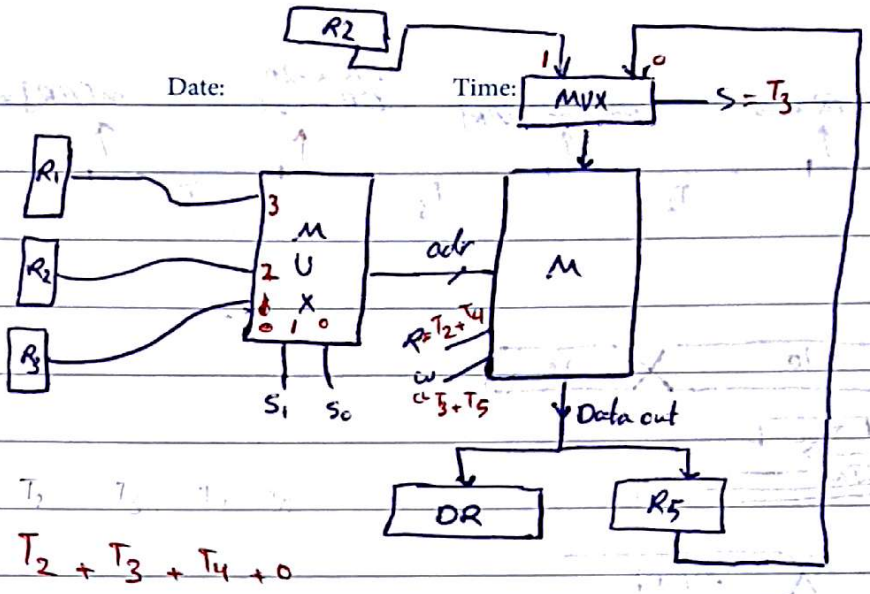
$T_5: M[R_4] \leftarrow R_5$ (41)

این نمودار می‌گه رجیسترهای R_1 و R_3، R_4 باید توی مسیر آدرس بمونن



همه اینا به شرطی درسته که Tacc از کلاک کمتر باشه،

Subject: Date: Time:



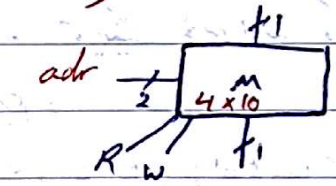
$S_1 = T_2 + T_3 + T_4 + 0$

$S_0 = T_2 + 0 + T_4 + T_5$ (wait اضافه کنیم)

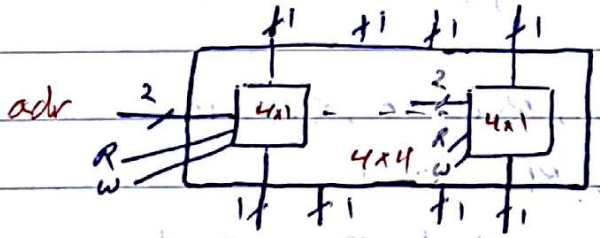
آنه T acc به پریود نبود با یه مثل قبل بریم ولی آنه مجبوری بزرگ غلط چرا که هر بار که بررسی کرده به T_2 مقدار R1 عوض شده پس $R_1 \leftarrow R_{1+1}$ رو می‌ذاریم تو مدار

Memory Mapping (نقشه حافظه (نقشه I/O

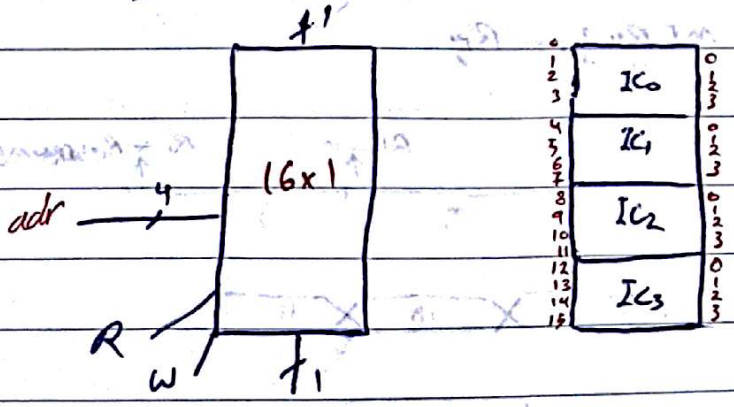
می‌خوایم با عددی



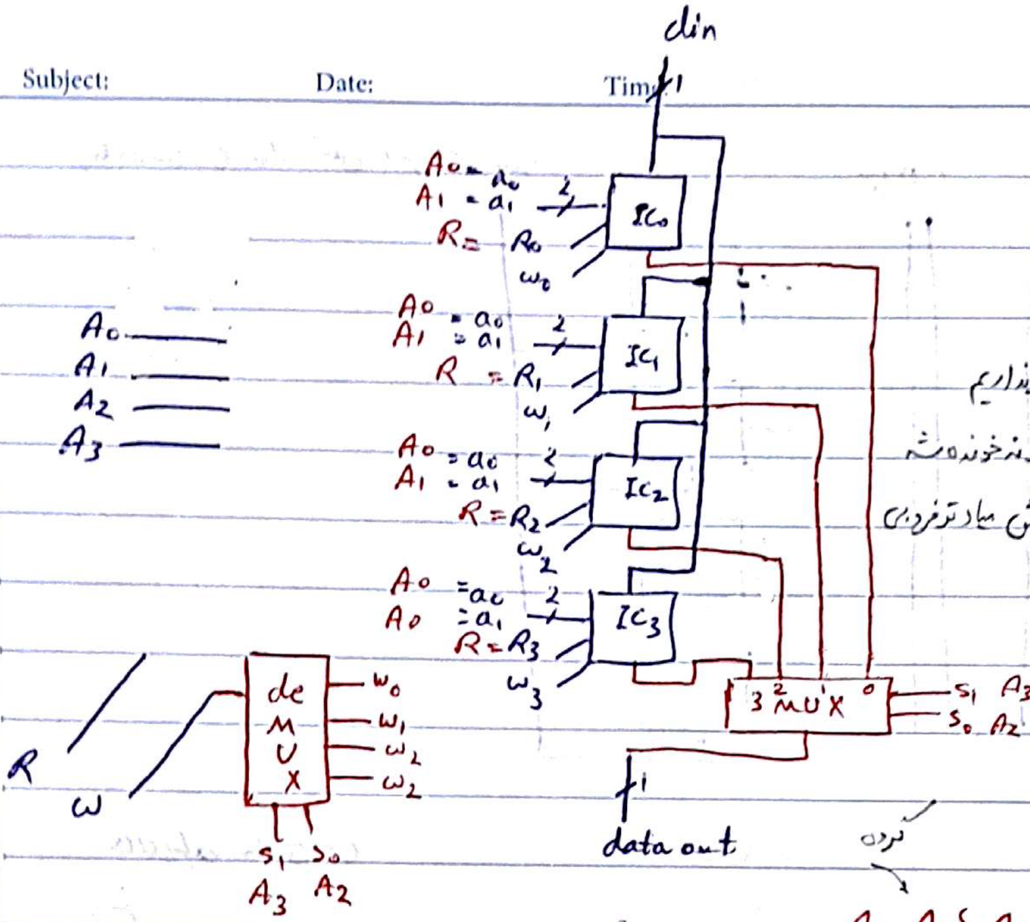
بسازیم 4bit ای عددی 1bit



یا با عددی 4x1 می‌خوایم عددی 16x1 بسازیم



Subject: Date: Time:



می‌تونیم همه R_i ها رو برداریم

R چون همش می‌تونن خونده شه

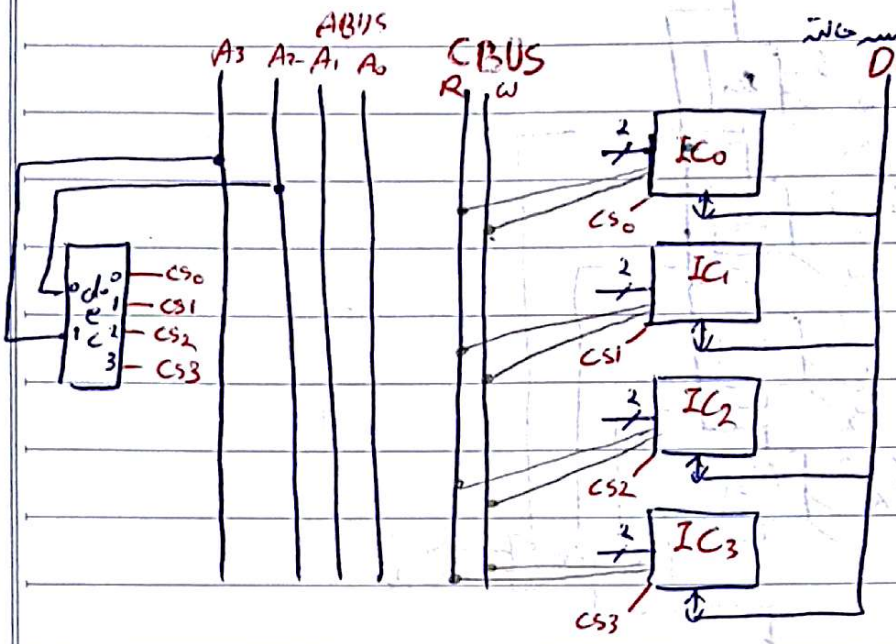
ولی به خاطر MUX یکیش میاد تو خروجی

می‌تونیم به جای MUX از خروجی سه حالته استفاده می‌کنیم.

A_3	A_2	A_1	A_0		a_1	a_0
0	0	0	0		0	0
0	0	0	1	IC_0	0	1
0	0	1	0		1	0
0	0	1	1		1	1
0	1	0	0		0	0
0	1	0	1	IC_1	0	1
0	1	1	0		1	0
0	1	1	1		1	1
1	0	0	0		0	0
1	0	0	1	IC_2	0	1
1	0	1	0		1	0
1	0	1	1		1	1
1	1	0	0		0	0
1	1	0	1	IC_3	0	1
1	1	1	0		1	0
1	1	1	1		1	1

Subject: Date: Time:

پیاده سازی حداکثر تعداد تبلی با فرض سه حالته



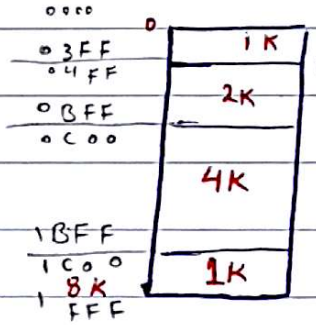
حال پیاده سازی این

به منظور

1- محاسبه آدرس هر IC برحسب Hex

2- خطوط آدرس IC ها

3- CS ها ؟



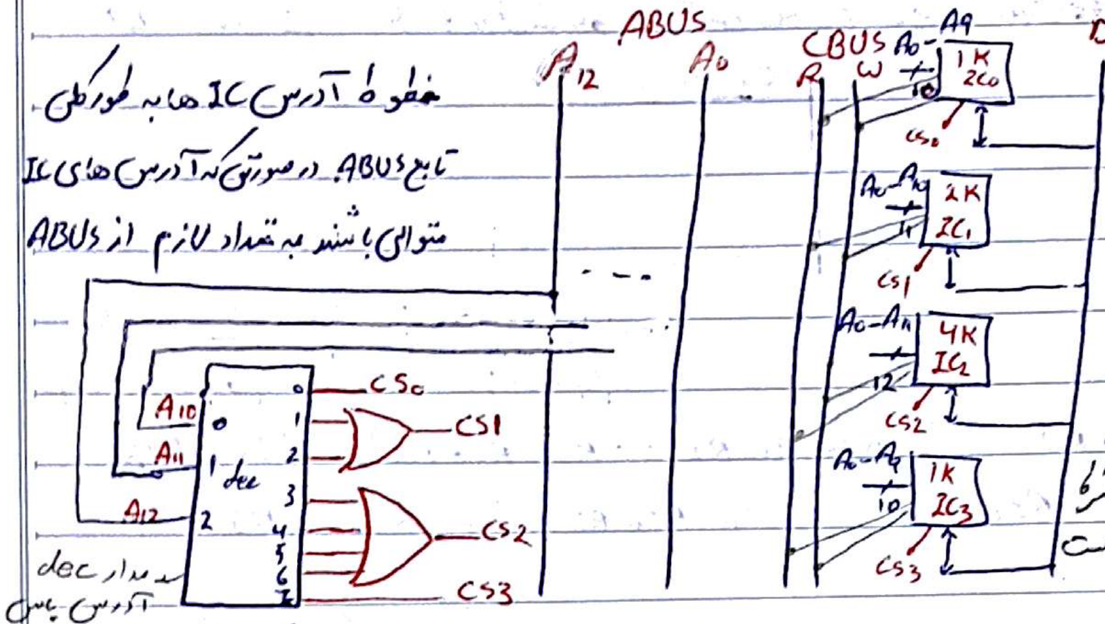
8K → 13 خط

	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Hex	
000	0	0	0	0	0	0	0	0	0	0	0	0	0	0000	IC_0
	0	0	0	1	1	1	1	1	1	1	1	1	1	03FF	
001	0	0	1	0	0	0	0	0	0	0	0	0	0	0400	IC_1
010	0	0	1	1	1	1	1	1	1	1	1	1	1	0BFF	
011	0	1	1	0	0	0	0	0	0	0	0	0	0	0C00	
100															IC_2
101															
110	1	1	0	1	1	1	1	1	1	1	1	1	1	1BFF	
	1	1	1	0	0	0	0	0	0	0	0	0	0	1C00	
111														1FFF	IC_3

Subject: Date: Time:

منظور از آدرس IC ها

خطوط آدرس IC ها به طور کلی تابع ABUS. در صورتی که آدرس های IC متوالی باشند به تعداد لازم از ABUS



اگه نشه باشه تراشه ها از خونه پوشش آدرس بدن باید جوری mapping کنیم که اونجوری نشه. تعلق مثال از ترتیب پنج باشه که ترتیب به پیوا باشه هست

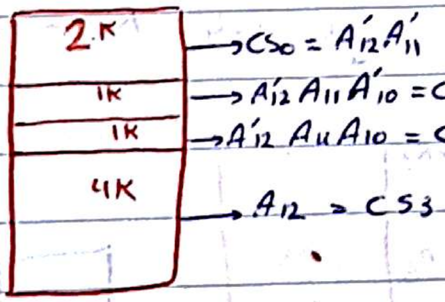
dec سه مدار آدرس باس

CS ها هم به طور کلی تابع ABUS

حالا می‌خوایم مینیمم سازی کنیم و CS ها رو بسازیم

A_{12} \ $A_{11}A_{10}$	00	01	11	10
0	(0) CS_0	(1) CS_1	(3) CS_2	(2) CS_1
1	(4) CS_2	(5) CS_2	(7) CS_3	(6) CS_2

حالا می‌خوایم ساده سازی کنیم اما باید شرط توالی رو هم حفظ کنیم. باید مثلن CS_2 ها توی خونه‌های 4, 5, 6, 7 باشن. برای اینکه CS ها ساده بشن ترتیب IC ها رو عوض می‌کنیم. (به خاطر از بین رفتن توالی اون 5 رو نباید ترکیب کنیم.)



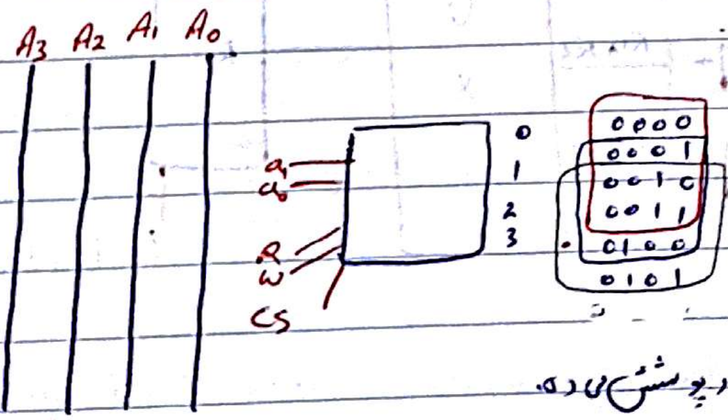
$CS_0 = A'_{12}A'_{11}$

$A'_{12}A_{11}A'_{10} = CS_1$

$A'_{12}A_{11}A_{10} = CS_2$

$A_{12} = CS_3$

AND 3 تا شد جای AND 8 تا و OR 2 تا



هر ترکیب 4 تایی باشه توالی باشه ... میدیم به آدرس چون همه حالت ها رو پوشش می‌ده.

اگه 0، 2، 4، 6 بخوایم ← a_2, a_1 ← 0000, 0010, 0100, 0110

ARYA

Subject: Date: Time:

اما 0 و 2 و 4 و 8 رو فقط با دو bit آدرس ساخت. با چیپ سلکت می‌سازیم.

(0001, 0010, 0100, 1000)

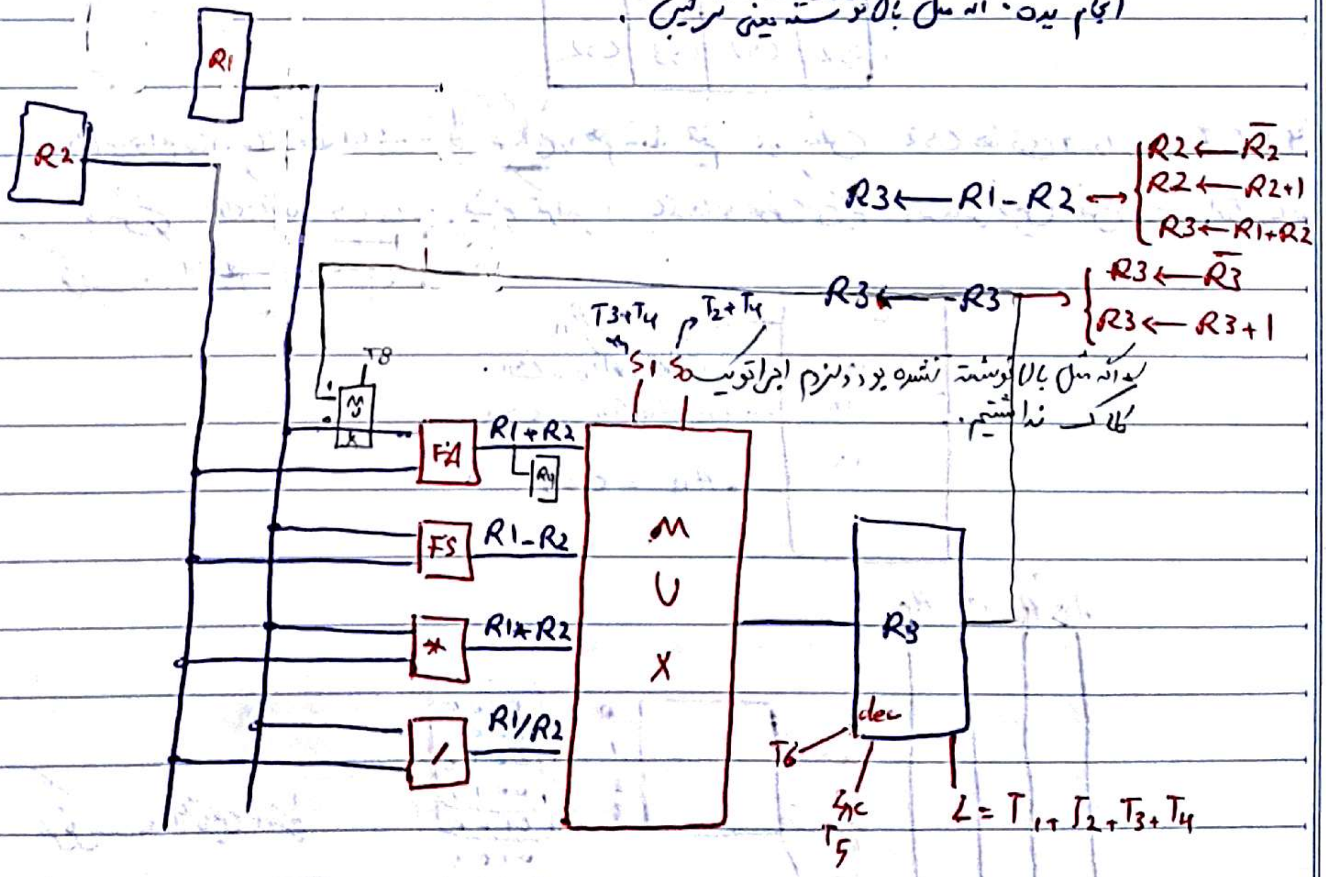
	CS	a_1	a_0
0	1	0	0
2	1	0	1
4	1	1	0
8	1	1	1
سایر	0	α	α

A-op حسابی - محاسبات عدد و شرط‌ها

$T_1 : R_3 \leftarrow R_1 + R_2$ $T_2 : R3 \leftarrow R1 - R2$ $T_3 : R3 \leftarrow R1 * R2$ $T_4 : R3 \leftarrow R1/R2$

$T_5 : R3 \leftarrow R3 + 1$ $T_6 : R3 \leftarrow R3 - 1$ $T_7 : R3 \leftarrow -R3$

ممکنه هر عملگرها مدار ترکیبیش تو Ram نباشه مثلن برای تفریق تو یه کلاک توسط کلاک انجام بشه وقتی جوری که مثل بالا نوشتن باید با 2 کلاک انجام بده. اگه مثل بالا نوشته یعنی ترکیبی.



$R3 \leftarrow R1 - R2 \rightarrow \begin{cases} R2 \leftarrow \overline{R2} \\ R2 \leftarrow R2+1 \\ R3 \leftarrow R1+R2 \end{cases}$

$R3 \leftarrow -R3 \rightarrow \begin{cases} R3 \leftarrow \overline{R3} \\ R3 \leftarrow R3+1 \end{cases}$

چون اگه مثل بالا نوشته نشده بود دو لازم اجرا توی یک کلاک نداشتیم.

اگه می‌گفت توی $R4 \leftarrow R3 + R2, T1$ ادن مرحله باید یه FA اضافه می‌کردم اما الان داشتیم

$T_8 : R4 \leftarrow R3 + R2$

می شه مثل مدار بالا

$T_1: R1 \leftarrow R_2 \wedge R_3$ (and) → selective clear

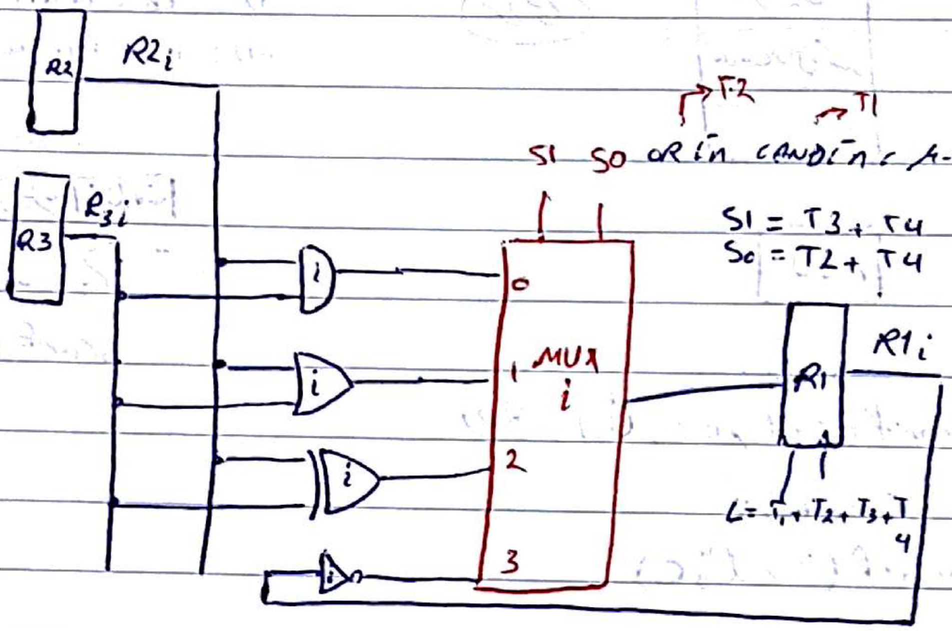
$T_2: R1 \leftarrow R_2 \vee R_3$ (or) → selective set

$T_3: R1 \leftarrow R2 \oplus R3$ → selective comp

$T_4: R1 \leftarrow \overline{R1}$

1010 ∧ ∨
1100
1000
1110

bit به bit با bit and و or، and یا or می شن.



μ-op: انتقال، حسابی → پردازش عددی و تست ها؛ منطقی، شیفت } پردازش های غیر عددی و تست ها

selective comp - selective set - selective clear

R [R3 | R2 | R1 | R0] ⇒ $R3R2 0 R0$: R ∧ 1101

$R3 R2 1 R0$ = R ∨ 0010

$R_3 R_2 R_1' R_0$ = R ⊕ 0010

اگه بخوایم از این دوتا رجیستر این رشته ۴ بیتی زیرو بسازیم:

R1 [$R1_3$ | $R1_2$ | $R1_1$ | $R1_0$]   R2 [$R2_3$ | $R2_2$ | $R2_1$ | $R2_0$] → $R1_1 R1_0 R2_1 R2_0$

اون تیکه هایی از R1 و R2 که نمی خوایم رو selective clear می کنیم بعد R1 رو شیفت می دیم بعد or می کنیم.

تأخیر این مدار، اندازه یه AND و یه بلاک En AND انجام می شه.

Subject: Date: Time:

int A, B

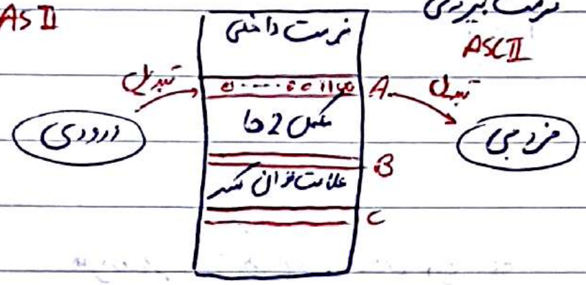
فرمت بیرونی

ASCII

ورودی - خروجی

فرمت بیرونی ASCII، 12 و -12 رو با اعداد وارد می‌کنیم

با کلیدها وارد می‌کنیم.



12 : 00...001100 (دودویی / داخلی)

-12 : 11...110100

حالا اگه float باشه:

علامت	نرمالیزه	توان / اعشار
-	+2	0.12

→ -12

ALU

دستور تبدیل با scanf

decimal، octal

scanf("%d %o", &A, &B) ← دستور ورودی

به جای d حالت مبنای ده یا x یا Hex یا o اکتال

C = A+B;

printf("%d", C) ← چاپ خروجی

	Hex	BIN	DEC
A	41	01000001	65
B	42	01000010	66
⋮			
Z			
⋮			
a	61	01100001	97
b	62	01100010	98
⋮			
z			

اگر حرف بزرگ رو بخواهیم کوچک کنیم +32 می‌کنیم.

if(C1>='A' && C1<='Z')

C1 = C1+32

C1 یعنی یک حرف است.

یا کار زیر را بکنیم

C1 = C1 | 0x20

0010 0000 ← Hex

or

چون فرق حرف بزرگ و کوچک توی bit پنجم با ارزش 32 ه

ورودی

ASCII → BCD unpacked → BCD packed → BIN

12 : 00110001 00110010 | 00000001 00000010 | 0001 0010 | 00001100

(1، 2 — دهگان، یکان)

در کد ASCII اعداد ۴ بیت اول 3 بعد خود عدد

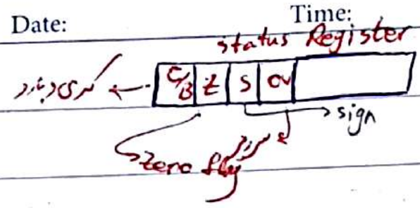
ورودی تبدیل به BIN یا BCD packed می‌شه، خروجی

BCD unpacked به BCD packed آسونه شیفت میدیم ۴ تای اول برهد ولی تبدیل به BIN نیاز به محاسبات داره.

```
    0000 0001
    0000 1010   ×
    ---------
    0000 1010
    0000 0010
    ---------
    ... 001100
```

Subject: Date: Time:

اینا همون رو به واحد کنترل کمک می‌کنن بیت شرط داشته باشیم



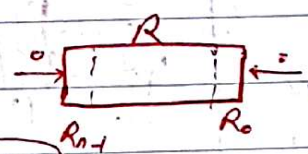
حاصل عملیات status bitها رو مقدار دهی می‌کنند.

$A>B \Rightarrow A-B>0 \rightarrow Z=0,\ S=0,\ B=0$

$A=B \Rightarrow A-B=0 \rightarrow Z=1$

سه نوع شیفت داریم

شیفت منطقی:



R ← shl R

R ← shr R

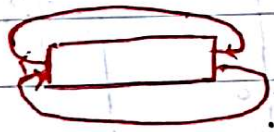
شیفت حسابی:

ضرب در 2 R ← A shl R

تقسیم بر 2 R ← A shr R

شیفت راست حسابی با بقیه فرق داره ولی شیفت چپ حسابی و منطقی یکی هستن.

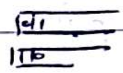
شیفت چرخشی



R ← cirl R

R ← cirr R

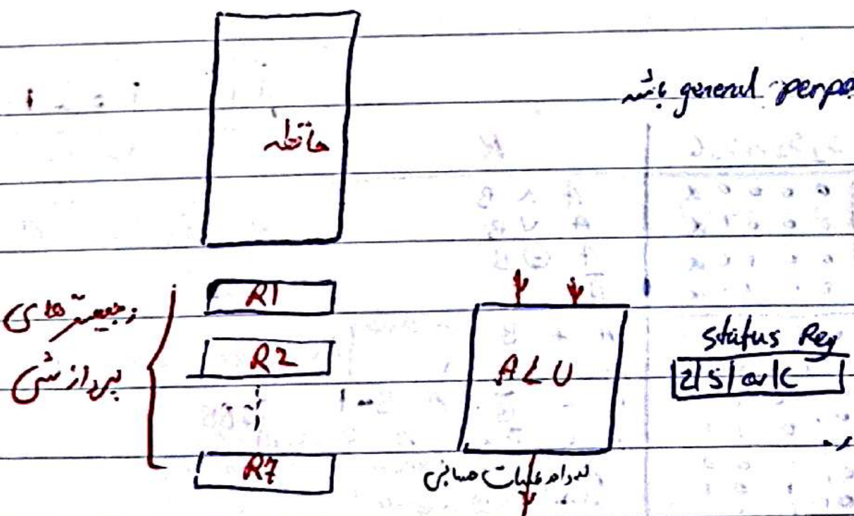
در شیفت حسابی علامت حفظ میشه توی شیفت راست bit علامت تکرار میشه. سرریز در شیفت چپ حسابی زمانی اتفاق می‌افته که دو بیت چپ با هم مخالف باشن. در این موقع اگه یه شیفت بدیم سرریز اتفاق می‌افته.

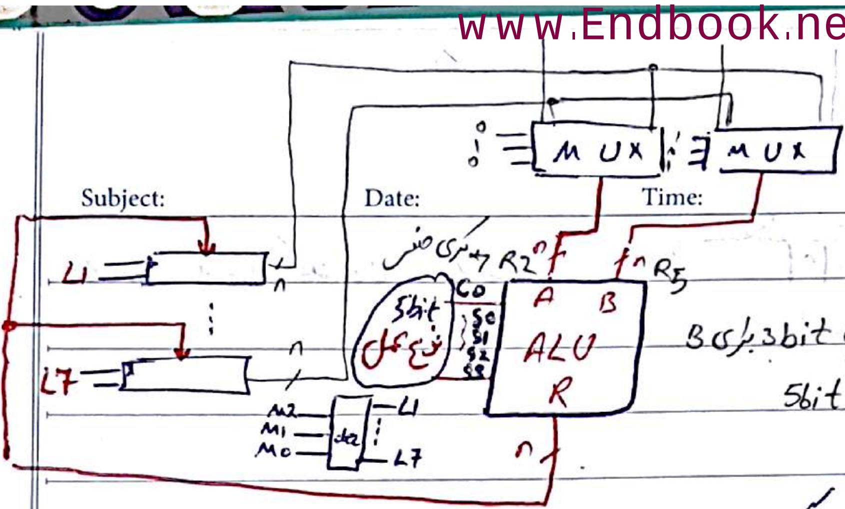


این processor یه general perpose باشه

برای اینکه چند عملیات رو همزمان انجام بدیم باید چندتا ALU ایجاد کنیم

2تا ALU برای … کار.

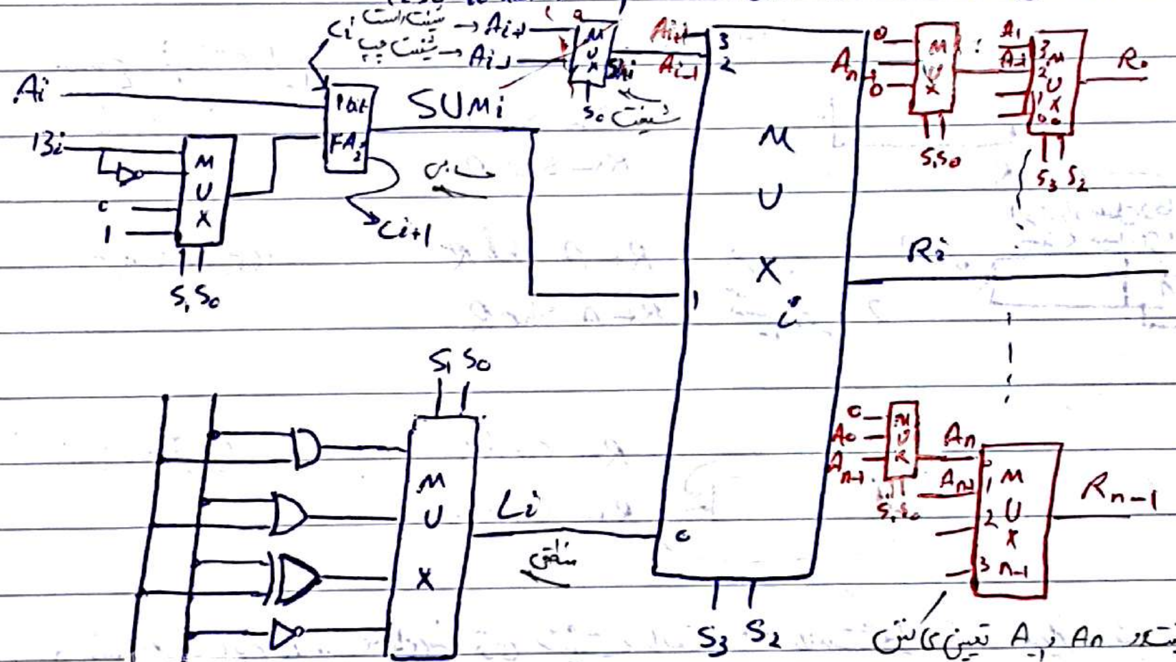




9bit فرمان لازم داره 3bit برای A، 3bit برای B
و 3bit برای خروجی + نوع عمل 5bit
فرمان 14bit

برای عمل‌های تکی مثل NOT، A ورودی می‌گیریم.

داخل ALU برای 1bit، i ام (i=0 to n-1)



نوع شیفت و A_{-1} و A_n تعیین می‌کنن

$A_n \; A_{n-1} \cdots A_1 A_0 \; A_{-1}$

چون هم‌زمان عمل منطقی و حسابی رو نمی‌خوایم می‌تونیم selecter های MUX ها رو یکی بذاریم.

C0 رو هم آزاد می‌تونیم MUX شیفت رو حذف کنیم 3 رو می‌بریم به A_{i+1} و 2 رو می‌بریم به A_{i-1} اتریز

| 1 1 .... 1 = -1 |

$A_{n-1} \cdots A_0$
1 ... 1
A-1

از تفریق منطقی کامله

	$S_3 S_2 S_1 S_0 C_0$	R
منطقی	0 0 0 0 α	$A \wedge B$
	0 0 0 1 α	$A \vee B$
	0 0 1 0 α	$A \oplus B$
	0 0 1 1 α	$\bar{A}$
حسابی	0 1 0 0 0	$A + B$ → ADD
	0 1 0 0 1	$A + B + 1$
	0 1 0 1 0	$A + \bar{B} = A - B - 1$
	0 1 0 1 1	$A + \bar{B} + 1 = A - B$ → SUB
	0 1 1 0 0	A → انتقال (MOV)
	0 1 1 0 1	A+1 → INC
	0 1 1 1 0	A-1 → DEC
	0 1 1 1 1	A
شیفت چپ	1 0 0 0 α	ShlA
	0 1 α	ClA
	1 0 α	AShlA
شیفت راست.	0 0 α	ShrA
	0 1 α	CrA
	1 0 α	AShrA

ADD و A+B+1 } ADC → جمع با کری

SBB → تفریق با بارو

می‌خوایم شیفت‌ها هم کامل باشه تری شیفت‌ها تو درنظر بشن.

ARYA

Subject: Date: Time:

B و A رو یکسان می‌گیریم بعد $A+\bar{B}$ رو انتخاب ...۱۱۱ ← RI می‌کنیم.

RI ← 0 → A که نشه جمش رو می‌دانیم

حالا اگر مثلن processor ۸ بیتی بود و ورودی ها 32bit ی هر ورودی دو کلمه می‌گیره.

long int A, B

واسه اینکه طول این ورودی ها زیاده تو چند کلمه ذخیره می‌شه مثلن توی جمع باید کری ضبط شه کار می‌کنه همینه. اول low ها رو جمع می‌کنیم بعد کری رو نگه می‌داریم برای High ها و با اونا جمع می‌کنیم. SBB هم برای همینه منتها برای تفریق

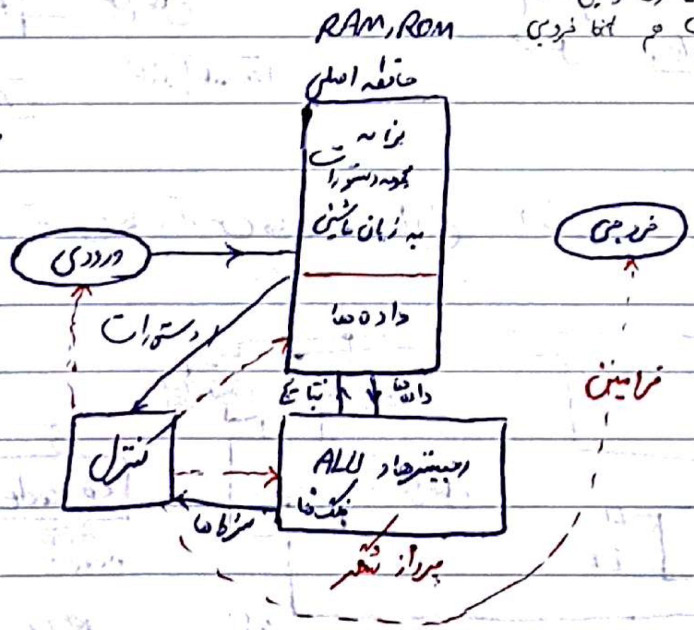
SUB A, B   A−B

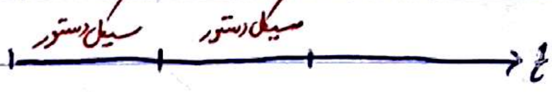
SBB A, B   A−B−borrow

ولی برای ۸ هم شیفت تفریق ندارا از Low شروع شده از

ولی شیفت هم وابسته اند و بیت خروجی High باید ضبط شه.

فلگ کری:
- در تفریق با او خروجی
- در جمع کری خروجی
- در شیفت هم اینا خروجی





دسته بندی دستورات:

انتقال، پردازشی {حسابی، منطقی، شیفت}، ورودی/خروجی، کنترل برنامه، کنترل ماشین

Subject: Date: Time:

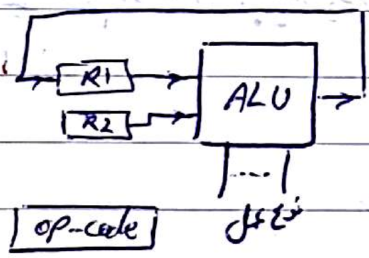
فرمت دستورات (ساختار)

نوع عمل (op-code)	آدرس ها (AFs) — address field

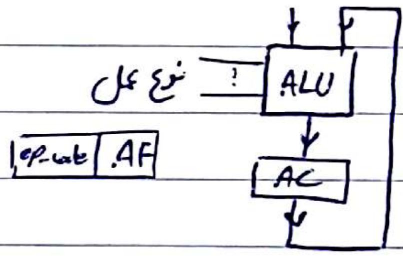
۱- تعداد AF ها در یک دستور را انتخاب

	تعداد آدرس مورد نیاز	تعداد AF در پردازنده‌ها
انتقال	2 ← مبدا، مقصد	
پردازش یونری	2 ← چه چیکار کنه	
پردازش باینری	3 ← داده اول، داده دوم، کجا بریزه	سازمان رجیسترهای عمومی RISC: 3، 2، 1، 0 (اکومولاتور در سازمان AC، stack در سازمان پشته)
کنترل برنامه غیرمشروط	1	
مشروط	2	

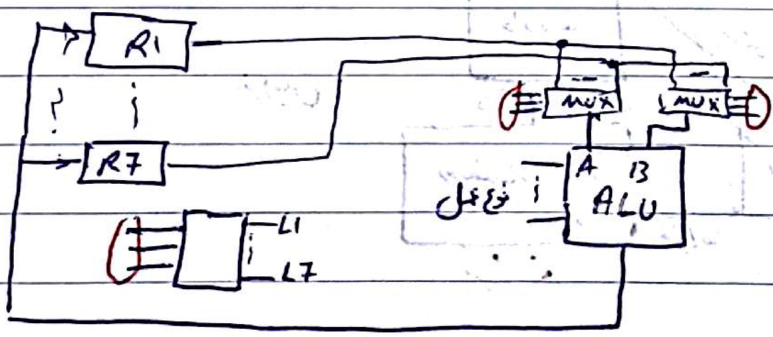
سازمان پشته



۱- سازمان AC



در RISC آدرس ها، آدرس های رجیستران (مثال جلسه قبل)



op-code (5)	AF1 (3)	AF2 (3)	AF3 (3)
ADD	R1	R2	R3
00001	001	010	011

$R3 \leftarrow R1 + R2$

که آدرس فیلد 3 رو حذف کنیم ممکن میشه، $R1 \leftarrow R1 + R2$

پردازنده‌ای که می‌خواهیم طراحی کنیم یکی از این سازمان ها باید انتخاب بشه ما سازمان AC رو انتخاب می‌کنیم.

Subject :
Year . Month . Date . ( )

2- مدهای آدرس دهی و انتخاب (نحوه تفسیر AF است.)

اون آدرس فیلدهایی که نمی‌شه انتخابش کرد و فیکس شده است یکی مدش ضمنی است، سترا است.

1- مد ضمنی

2- مد بلافصل $AC \leftarrow AC+20$

3- مد حافظه‌ای مستقیم $AC \leftarrow AC+M[20]$

4- مد حافظه‌ای غیرمستقیم pointer توی حافظه $AC \leftarrow AC+M[M[20]]$

5- مد رجیستری مستقیم $AC \leftarrow AC+R20$

6- مد رجیستری غیر Pointer توی رجیستر $AC \leftarrow AC+M[R20]$

7- اندیس

8- نسبی ، 9- بیس

بسته به بازدهی و کارایی یک یا چند نوع انتخاب کرد ما چون بازدهی مهم نیست دامنه general purpose بودن مهمه 3 و 4 رو انتخاب می‌کنیم. برای اینکه بفهمه کدوم رو انجام بده 3 یا 4 یه bit لازم داره که بفهمه مد حافظه مستقیم منظوره یا غیرمستقیم. (آدرس مد معین ← 1bit)

mode | op-code | AF

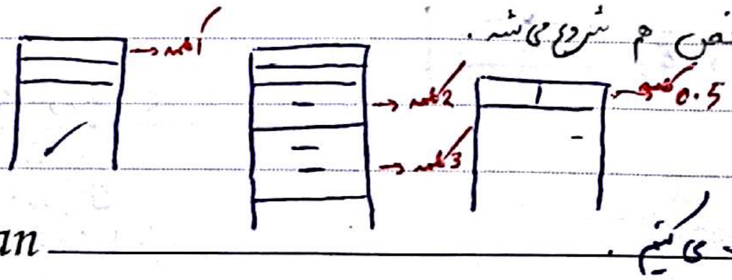
$\Rightarrow AC \leftarrow AC + M[AF]$ ← 2 مراجعه به حافظه

$M[M[AF]]$ ← 3 مراجعه به حافظه

توی RISC راندمان بالاست چون کد دستورات توی حافظه نیست مقادیر رو از رجیستر برمی‌داره.

3- طول دستور و طول کلمه

همه دستورات طول آدرس‌هاشون یکسان نیست. از نظر تئوری طول دستور می‌تونه هر تعدادی از یک یا چند کلمه باشه از هر جایی شروع شه ولی تو پروسسورها اینطوری نیست راندمان داغون می‌شه. توی پروسسورها دستورات رو مضربی از کلمه می‌گیرن از یه جای مشخص هم شروع می‌شه.



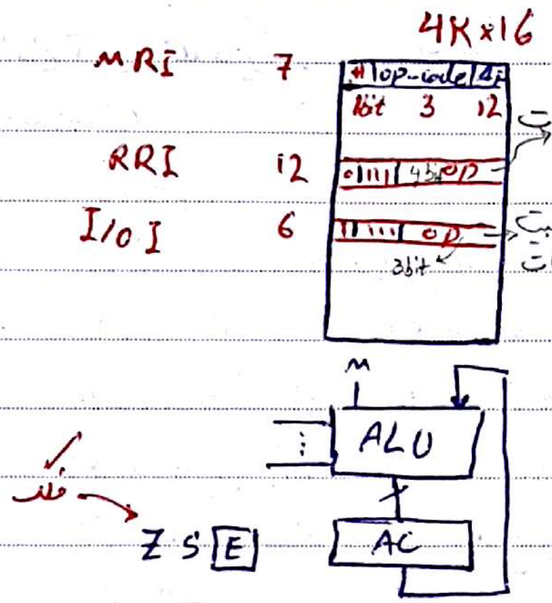
Ariyan

با هر دستور یک کلمه رو انتخاب می‌کنیم.

Subject :

Year . Month . Date . ( )

۴ـ طول کلمه چند بیت (مضرب byte)



تعداد بیت های آدرس فیلد به تعداد آدرس های حافظه ربط داره، اینکه کل ی حافظه رو می خوایم آدرس بدیم اینجا ما 4K می خوایم در نظر می گیریم کل حافظه تلفات 4K هست. پس اینجا 1byte نمی تونه باشه

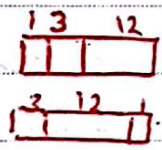
2byte می گیریم. ← op-codes 3bit

3bit ، هشت دستور تولید می کنه.

پیچیدگی بیشتره. یعنی بهتر بودن نیست.

ما می خوایم 25 دستور بدیم ← 3bit کمه

می شه مثلن 2 بیت آدرس رو بیاریم تو op_code

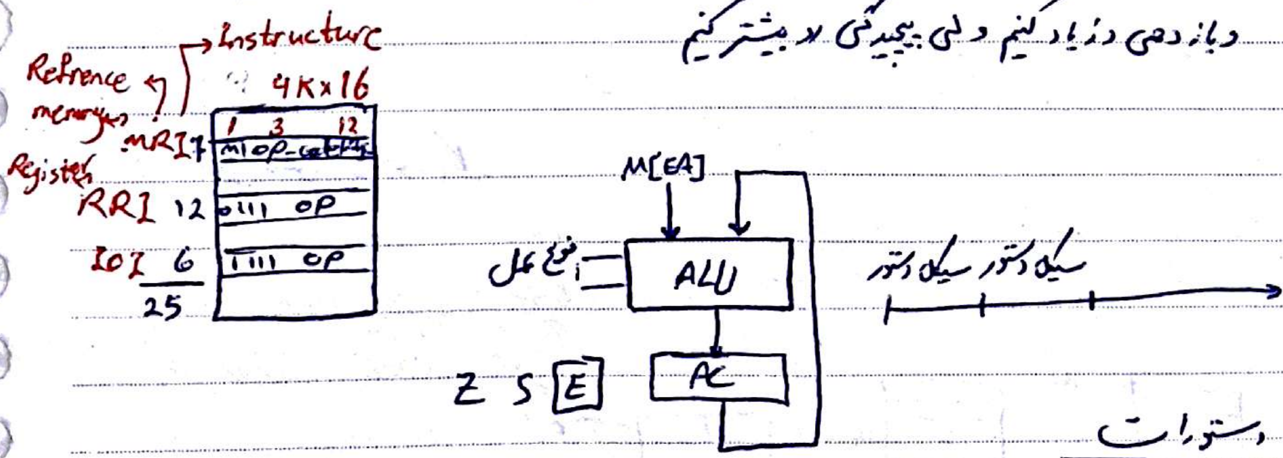


نمی تونیم اینجوری کنیم 8 تا با بالایی تولید کنیم 8 تا پایینی

چون حافظه نمی فهمه کدوم op-code اش کجاست.

از طرفی همه دستورها آدرس نمی خوان مثلاً Inc و Dec، اینا.

برای همین یه کلمه نداریم فقط برای op-code. برای اونایی که آدرس نمی خوان اما باید تمایز قائل شیم و اینا رو تفکیک کنیم واسه حافظه یا باید bit اضافه کنیم یا چند بیت اول رو اختصاص بدیم برای فهموندن به حافظه مثلن اگه 3bit آدرس بده یا بره یعنی پیش op-code ه ۰. یا باید درست کد تبدیل کنیم یه کار کنیم آدرس فیلدها ثابت کنن یعنی می تونیم تلفات و بازدهی رو زیاد کنیم ولی پیچیدگی رو بیشتر کنیم



دستورات

1- انتقال

2- پردازشی

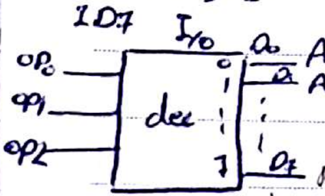
3- کنترل برنامه

4- ورودی و خروجی

5- کنترل ماشین

Subject :
Year . Month . Date .

$I'D_0$ ← AND مستقیم
ID_0 ← AND غیرمستقیم
$I'D_7$ رجیستری
ID_7 I/O



که با بیت m مشخص می‌شه.

گروه دستور	دستور	کد Hex	اثر EXE
2	AND	8 ddd / 0 ddd	$AC \leftarrow AC \wedge M[AF]$
2	ADD	9 ddd / 1 ddd	$AC \leftarrow AC \wedge M[M[AF]]$ → $AC \leftarrow AC \wedge M[EA]$; $E, AC \leftarrow AC + M[EA]$ ⇒ E کری خروجی
1	LDA	A ddd / 2 ddd	$AC \leftarrow M[EA]$
1	STA	B ddd / 3 ddd	$M[EA] \leftarrow AC$
3	BUN	C ddd / 4 ddd	$PC \leftarrow EA$
3	BSA	D ddd / 5 ddd	
3	ISZ	E ddd / 6 ddd	$M[EA] \leftarrow M[EA]+1$; if $(M[EA]=0)$ $PC \leftarrow PC+1$

چون 4 بیت از 12 بیت لازم
3 بیت 3 بیت کاری نیم داشته کدهای
RRI و I/O

که دستور فقط به D_7 اول نیاز
دارد RRI و AF ها
دیگر ندارد.

نوع دستور RRI ها با بقیه بیت
هاست.

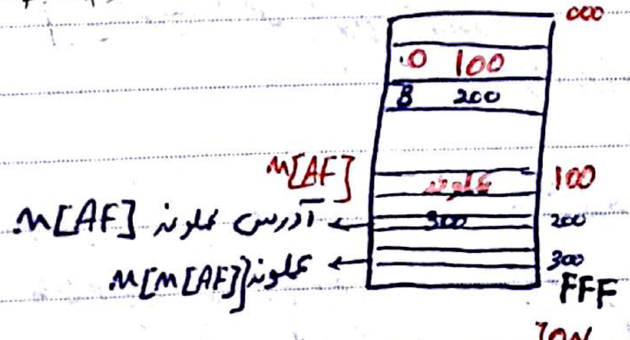
D_7 دستور حافظه‌ای
D_7' غیر حافظه‌ای

گروه دستور	دستور	کد Hex	اثر EXE
ID_7IR_{11}' 2	CLA	7800	$AC \leftarrow 0$
2	CMA	7400	$AC \leftarrow \overline{AC}$
2	CLE	7200	$E \leftarrow 0$
2	CME	7100	$E \leftarrow \overline{E}$
2	INC	7080	$AC \leftarrow AC+1$ آپ
2	Clr	7040	$E, AC \leftarrow$ Clr E, AC → [E ↔ AC]
2	Cll	7020	$E, AC \leftarrow$ Cll E, AC ساده
3	SZA	7010	if (Z) $PC \leftarrow PC+1$
3	SZE	7008	if (E') $PC \leftarrow PC+1$ → S ≥ 0
3	SPA	7004	if $(AC>0)$ $PC \leftarrow PC+1$ (if ($S'Z'$)) ; S → علامت
3	SNA	7002	if $(AC<0)$ $PC \leftarrow PC+1$ (if (S)) ; Z → در بقیه صفر بودنش ; E → کری همچنین
5	HLT	7001	توقف سیکل کلاک IO2 ; توی تک شیفت ها به کار میره

هر M op ها می‌زنن به بخش کنترل
که نه عملی فعال شه.

گروه دستور	دستور	کد Hex	اثر EXE
4	INP	F800	
4	OUT	F400	
3,4	SKI	F200	if (IFG) $PC \leftarrow PC+1$ ~ input flag
ID_7IR_8 ← 3,4	SKO	F100	if (OFG) $PC \leftarrow PC+1$ ~ out put flag
5	ION	F080	فعال کردن وقفه
5	IOF	F040	غیرفعال شدن وقفه

0111 1000 0000 0000
8 0 0
$I'D_7IR_{11}$



EA : محلی که از آن عملوند برداشته می‌شود. effective adress
غیرمستقیم m=1 → M[AF]
مستقیم m=0 → AF

ION
سیکل دستور — سیکل دستور — سیکل وقفه — سیکل دستور → t
F I E — F E
گرفتن دستور
گرفتن عملوند
اجرا
غیرحافظه

Ariyan

سیکل ماشین

Fetch برداشت دستور از حافظه تشخیص ← همه جا ۱ جوره

IND تعیین EA ← دستورهای غیرحافظه ای این مرحله را ندارند فقط F و E دارند ← اجرا

EXE اجرای دستور ← 25 دستور ← 25 شکل مختلف

← 27 سیکل ماشین + 1

سیکل دستور

INT ← سیکل وقفه

← 23 سیکل ماشین

برداشت دستور و تعیین EA برای همه یه جور انجام می شن اما EXE ها هر کدوم یه جور.

درگروه دستورات

3. کنترل برنامه ← تغییر ترتیب اجرا

5. کنترل ماشین ← تغییر رفتار پروسسور

اطلاعات لازم برای کار از پروسسور

هر پردازنده یه نقطه شروع fix داره که از اونجا شروع می کنه دستور رو از اونجا درج داره.

1- حافظه 4K×16

2- رجیسترهای پردازشی و فلگ ها AC و E و (S و Z)

3- دستورات (ترتیب، کد، اثر)

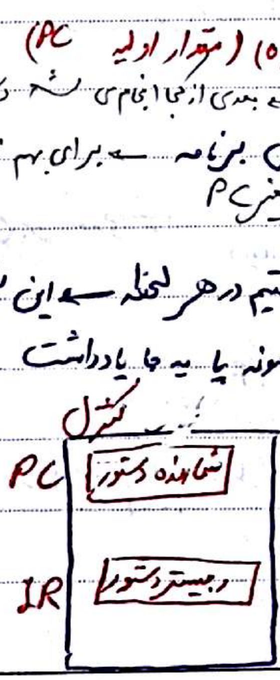
4- ترتیب اجرا | نقطه شروع تعدادی (0) (مقدار اولیه PC)

توالی تعداد ← بعدی از جا انجام می شه دستورهای کنترل برنامه این رو بهم می زنه.

دستورات کنترل برنامه ← برای بهم زدن توالی ترتیب و ایجاد تشکیل حلقه ها ← تغییر PC

ما باید بدونیم در کدام مرحله از اجرای دستور هستیم در هر لحظه ← این شماره باید ضبط شه حتمن مثلن ما می خوایم یه کاری انجام بدیم یا یادمون می مونه یا یه جا یادداشت می کنیم که الان توی این مرحله هستیم.

پس باید نگه داره و ضبط کنه که چه دستوری رو برداشته. وقتی کارش تموم شه بعد بره دستور بعدی.



Subject :
Year . Month . Date . ( )

IR ← M[PC] } سیکل Fetch

PC به نقطه شروع مقدار اولیه

به هم خوردن توالی ـ مقدار جدید تو PC می‌نشه ← توالی PC ← PC+1 | سیکل دستور سیکل دستور

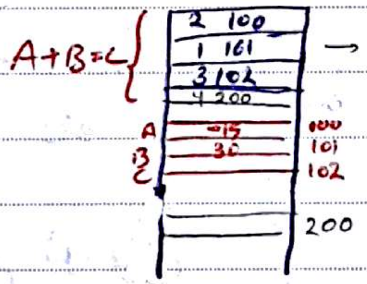
M[PC] M[PC]
PC←PC+1 PC←PC+1

دستورات 3 ، PC رو عوض می کنه : انتقال ـ حافظه به حافظه ، حافظه به رجیستر ، رجیستر به حافظه
رجیستر به رجیستر

اثر سیکل IND

$$EA \; AR \leftarrow \begin{cases} AF \\ M[AF] \end{cases}$$

انتقال معلوم شد ، پردازش مقدار یه رجیستر یا حافظه تغییر می‌کنه ، کنترل برنامه هم PC رو عوض می کنه
مثل goto

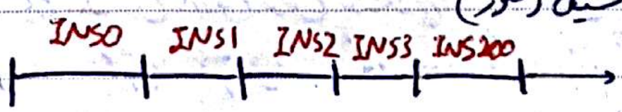


AC ← M[100]
AC ← AC+M[101]
M[102] ← AC

int A=15, B=-30, C=0;
C=A+B;

اگر تهش داشتیم : goto L1
L1 200

محور زمانی (سیکل دستور)



int A=0X123
A++; ⟶ Load A => incA => store A
2αα 7080 3αα

Ariyan

۲

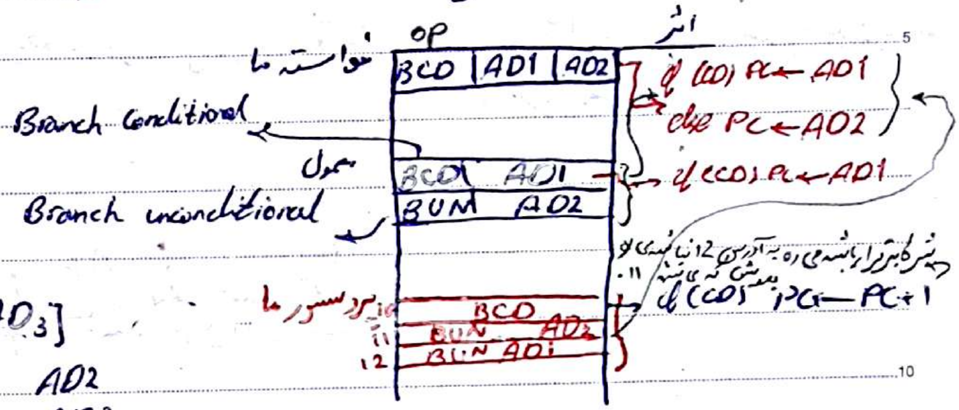
Subject :
Year . Month . Date . ( )

انشعاب غیر مشروط مثل go to (BUN)

مثال انشعاب مشروط (نیاز است.)

if (CD) go to 100

else go to 200



$M[AD1] \leftarrow M[AD_2] \wedge M[AD_3]$

Load ← LDA AD2

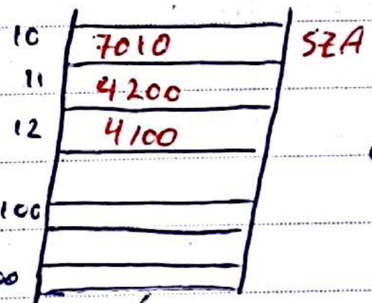
$\wedge$ ← AND AD3

store ← STA AD1

پس چه جوری انجام میدیم؟

skip Z AC ← SZA ← یک zero flag یعنی ← if (AC=0) go to 100

else go to 200



بقیه skipها هم مثل همین

ISZ دو کار می‌کنه یکی اینکه اول آدرس توش یدونه اضافه می‌کنه اگه حاصلش بعد اضافه کردن صفر شد یدونه می‌پره. یه بخش پردازشی + یه بخش کنترلی. کاری که Inc انجام میده کار سه تا دستوره اگه نبود باید Load می‌شد تو Inc AC می‌شد دوباره store می‌شه. استفاده‌اش در تشکیل حلقه‌ها

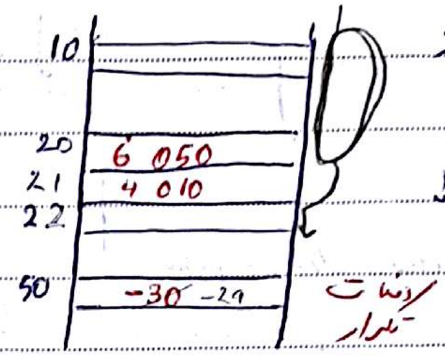
مثال میخوایم از خونه 10 بره تا آدرس 20، 30 بار بعد از حلقه بیاد بیرون (for)

یه خونه رو به عنوان متغیر می‌گیریم که باهاش بشمریم. (تو خونه 50 مثلا)

Ariyan

اگه اینو داشتیم توی هر حلقه ۴ بار باید اجرا می‌شدن، میدونی ۴ خونه یه خونه از حافظه اشغال می‌شه.

توجه حافظه رجیستر نیست قابلیت read, write داره M[EA] ← M[EA]+1 یه 1-op نیست. خونده می‌شه، Inc می‌شه، write می‌شه ≠ که 3 کلاک



BSA → Branch save address (call) ← فراخوانی زیربرنامه ← function

تنهایی هم نیست call داریم و RET (return) (مدیریت برنامه‌های بزرگ)

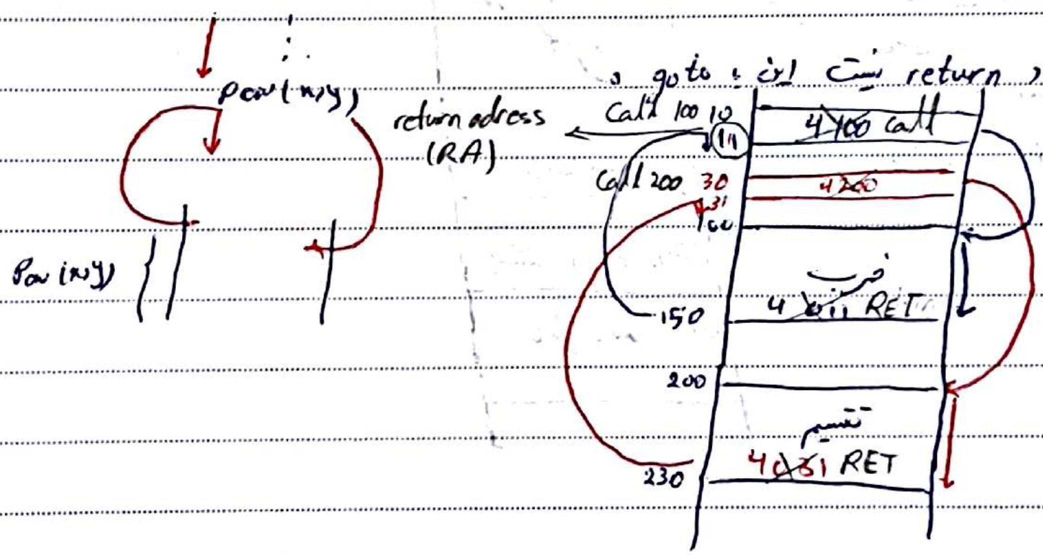
مثلن توی C میریم call می‌کنیم یه function رو Pow(x,y)

این call و return نیست این، goto و ...

مثلن دگ نمی‌نه

از خونه 40 شروع کرد

برگرده به 41



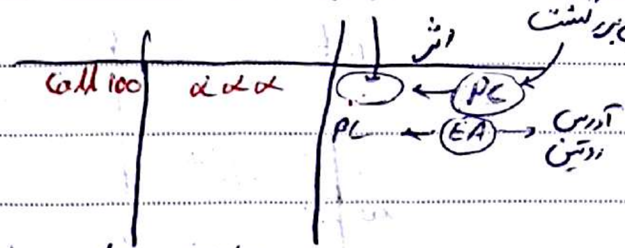
باید آدرس ها save بشن (کی برید) (کجا برگرده) کجا رفتیش ملزم است. ← return adress باید save شه.

ذاتِ call دوتا آدرس return یه آدرس.

دستور خارجی RET نداریم ولی قابل اجراست؛ BSA

پیاده‌سازی call و return.

صفحه بعد.



اول باید PC ضبط بشه و بعد اون آدرس روتین بره توش

return adress register

مثال BSA	پشته حافظه	محل ضبط یک رجیستر

return هم‌ا ذکر شده مثل ARM اینه.

انتر call: RAR ← PC
PC ← EC

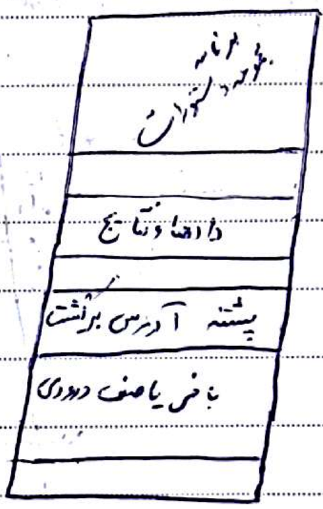
RET: PC ← RAR

first in - first out — (اول ضبط می‌شه اول هم استفاده می‌شه) اونی که بیاد تو اول هم برداشته می‌شه

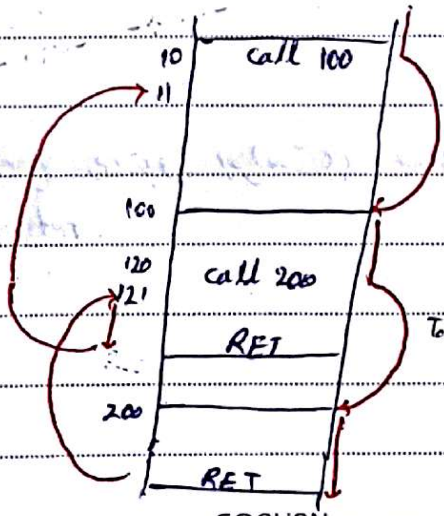
ترتیب ضبط دیتا ها در RAM

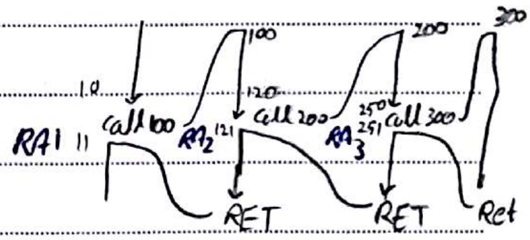
۱- تصادفی
FIFO queue ۲- صف
LIFO stack ۳- پشته

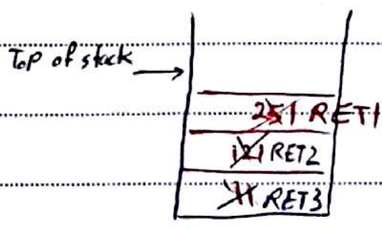
که اونی که آخر ضبط شده اول استفاده می‌شود.



کاربرد پشته توی call های تو در تو عه!





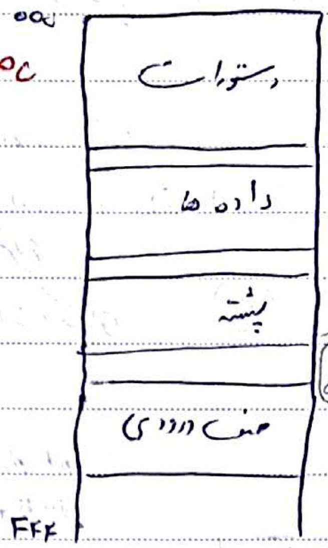


stack

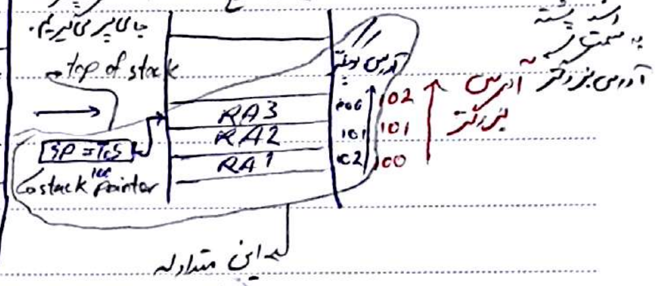
یا هر call روی پشته گذاشته می‌شه با هر RET برداشته می‌شه.

Subject:
Date

وقتی گذاشتیم SP پر نشون بده دگ همیشه باید پر نشون بده



SP ← پوینتری به بالای stack رو فقط نشون می‌ده، هرچند تا پشته می‌تونیم داشته باشیم ولی به تعدادشون پوینتر می‌خوایم. (می‌تونه جای پر نشون بده یا جای خالی) ← انتهای stack

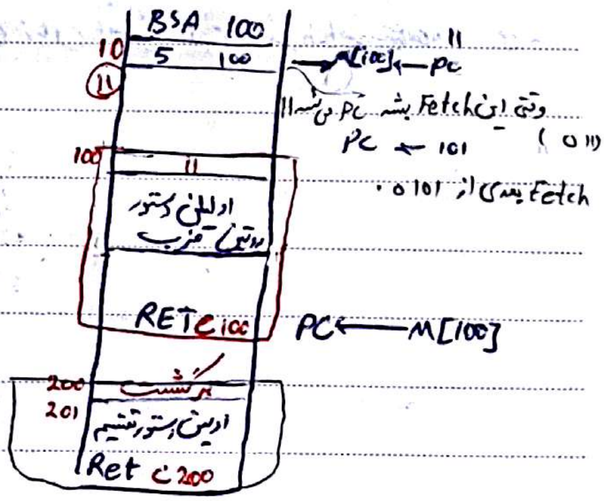


حالا درباره Call و Ret جلسه قبل رو دیدیم.

اثر دستورات	رجیستر	پشته حافظه	BSA → Branch save address
Call	RAR ← PC PC ← EA	SP ← SP-1 M[SP] ← PC PC ← EA	M[EA] ← PC PC ← EA+1
Ret	PC ← RAR	PC ← M[SP] SP ← SP+1	PC ← M[ ]

مثال از BSA

اولین دستور توی تابع همیشه می‌شه برگشت.



Ret ذاتن یک آدرسیه باید توی Ret آدرسشو بهش بدیم

حالا ممکنه توی یه پردازنده دستور Ret نباشه یه چیزی باشه همین کارو بکنه مثل اسمش mov باشه یا BUN این کار رو می‌کنه واسمون.

PC ← AF / M[AF]   BUN که xxx یا اینکه C رو می‌خوایم.

اگه تابع recursive باشه باید نرم‌افزاری پشته تشکیل بدیم که آدرس برگشت از بین نره.

Subject:
Date

ویژگی های یک پردسور (از دید کاربر = برنامه نویس به زبان ماشین)

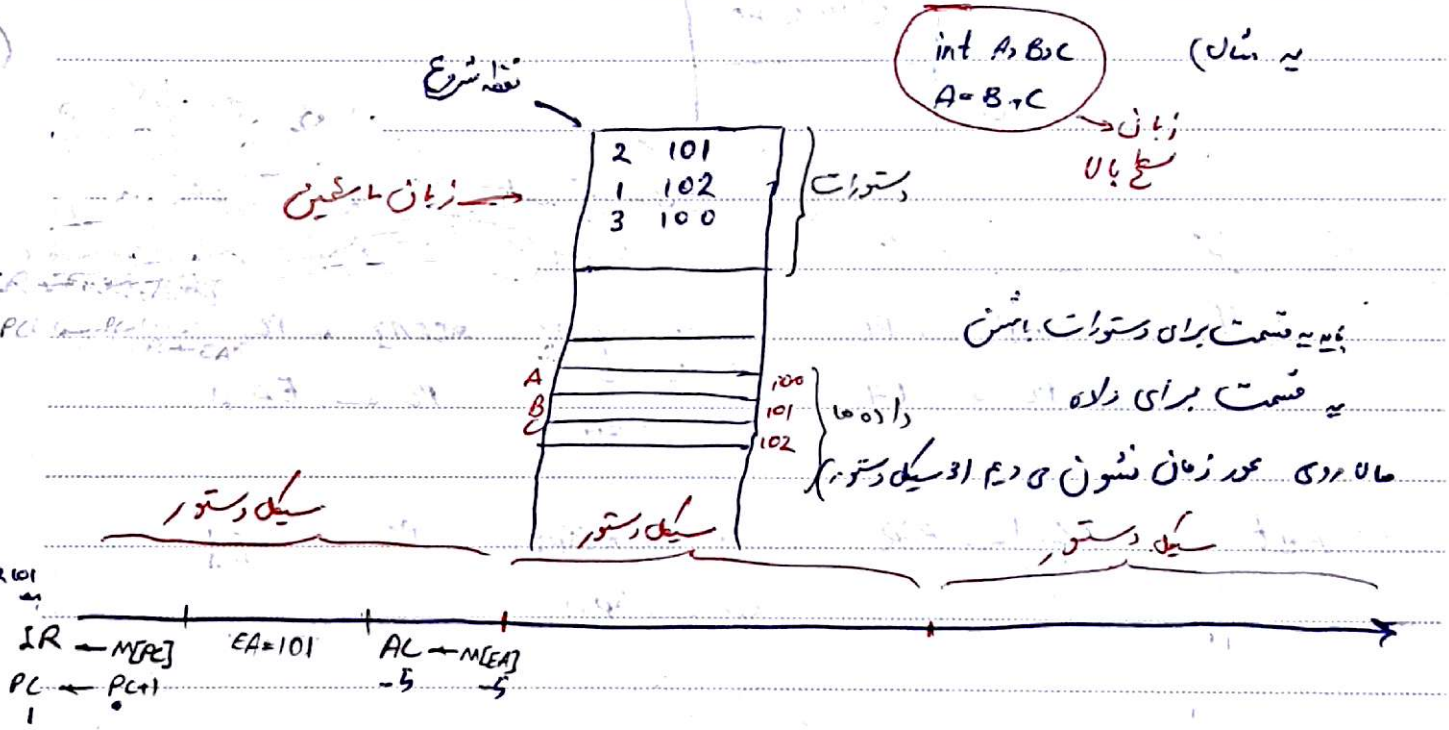
1- حافظه 4K×16

2- رجیسترها AC, S, E و Z

3- دستورات (فرمت - کد - مد - اثر) (جدول) 25 دستور

4- ترتیب اجرا | نقطه شروع / توالی دستورات کنترلی

(PC)



زبان سطح بالا باید ترجمه شه به زبان ماشین (در نهایت زبان ماشین اجرا میشه Fetch و enable می شه.

زبان اسمبلی

LDA B
ADD C
STA A

حالا میریم سراغ طراحی پردسور.

طرح پردازنده با ویژگی های که بیان شد:

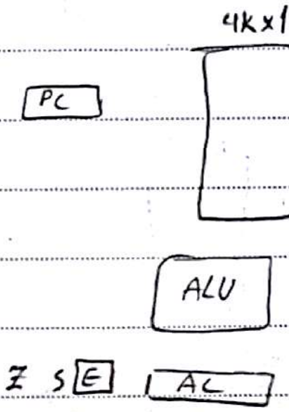
تو لیست دستورات نیست. — 1- رجیسترهای کمکی IR , DR , AR (برنامه‌نویس نمی‌دونه اینا رو)
↳ EA

2- مسیرها (باس) که دادن به دستور

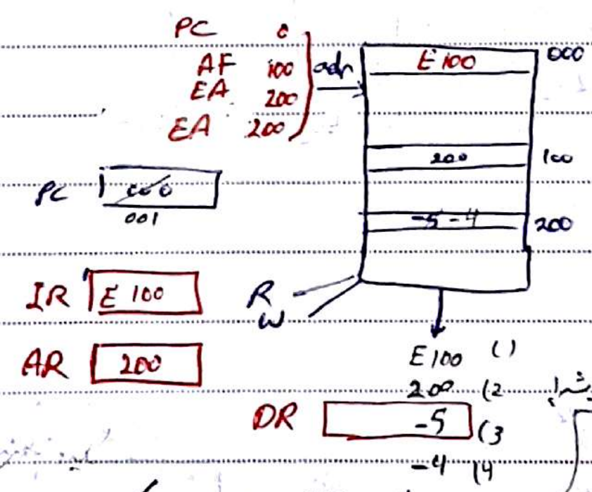
3- ساختار کنترل (ورودی‌ها - خروجی‌ها)

4- جزئیات بر حسب RTL

برنامه‌نویس نمای کلی پردازنده و لیست دستورات رو داره.



حالا اینارو داریم می‌خوایم طرحش کنیم.



خونه 200 می‌شه 4- ، PC هم می‌شه 1

حالا ببینیم واسه اینکه این اتفاق بیفته چیکار باید بکنیم

1) اول آدرس صفر رو به R هم می‌دیم + PC 1 می‌شه
2) بعد از گذاشتن 100 تو مرحله دوباره R می‌دیم
3) 200 می‌ره تو آدرس + R
4) یه 5- یکی اضافه می‌کنیم.
5) 200 می‌ره تو آدرس 4- می‌ره تو Datain + W
6) Z هم صفره، مرحله ادم اجرایی شد.

می‌نهیم ← 0 ISZ

یه سری خروجی‌ها می‌ده که لازمه یادمون بمونه ضبط بشه مثلن E100 باید آخر یادمون باشه ← از IR استفاده می‌کنیم

200 هم باید یادمون بمونه چون تو آخرین مرحله بهش نیاز داریم ← یه جای دیگه AR

(ورودی‌های EA)

5- هم روی هوا نمی‌شه زیاد کرد باید ببره توی یه R ازش Inc شد ← DR

اما مثلن 7800 ، CLA هیچ رجیستر کمکی نمی‌خواد. هیچ دستوری هم بیشتر از این 3 تا رجیستر کمکی، رجیستر کمکی بیشتری نمی‌خواد.

Subject: Year: Month: Date:

بنابراین این انتقال ها میرهم لازمه یا می‌شه میرفض گذاشت یا Bus که جزال باشه.

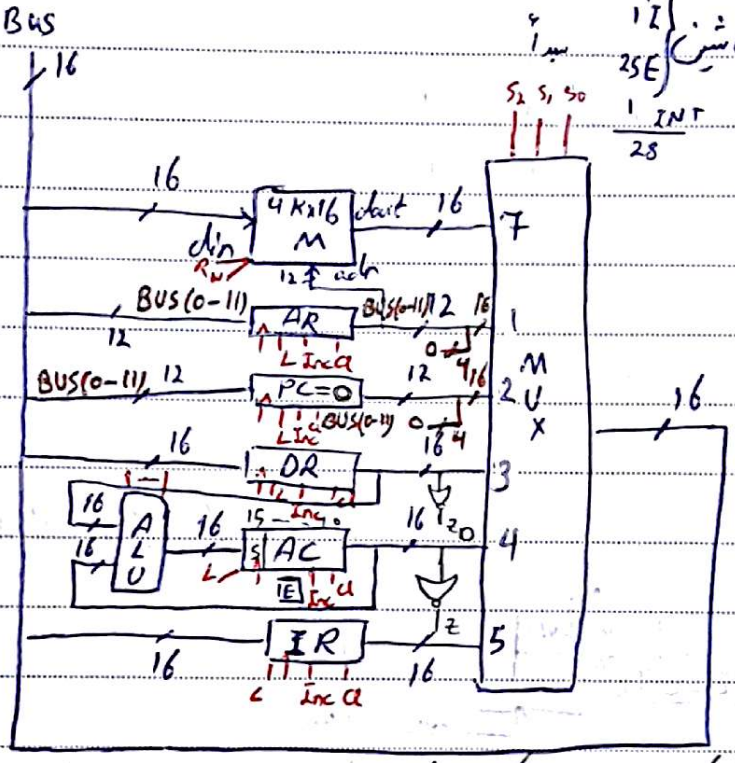
طرح پروسسور مبنا

۱- رجیسترهای کمکی لازم AR, DR, IR

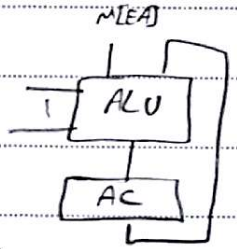
2- میرهای لازم (باس)

3- ساختار کنترل

4- RTL برای 28 سیکل ماشین



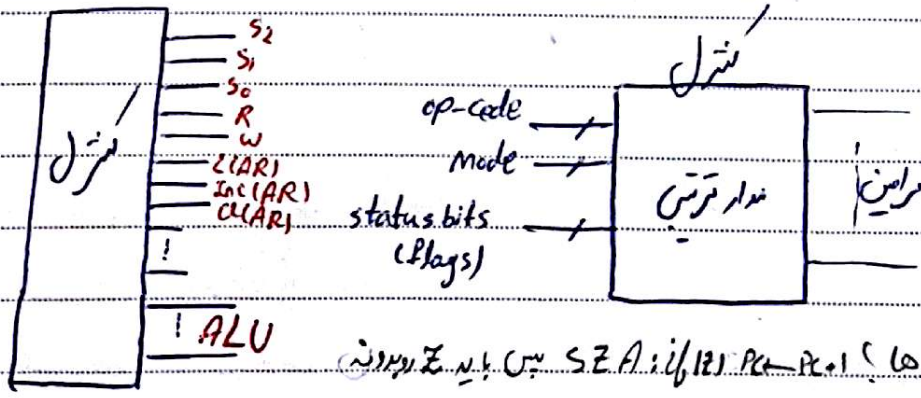
ترامین با تمرین



این M[EA] میاد تو OR از دیتای رم تو ALU

همیشه تغییرش داد آدرس ازیه باسه دیگه یا از OR آدرس بگیره ما فرض می‌کنیم از AR می‌گیره ممکنه همه رجیسترها Inc, Ld نخوان مالا فعلن میذاریم باشه.

واحد کنترل



چرا flag ها؟ SZA: if(21) PC←PC+1 بین با Z برود نه

Subject :
Year .   Month .   Date .   ( )

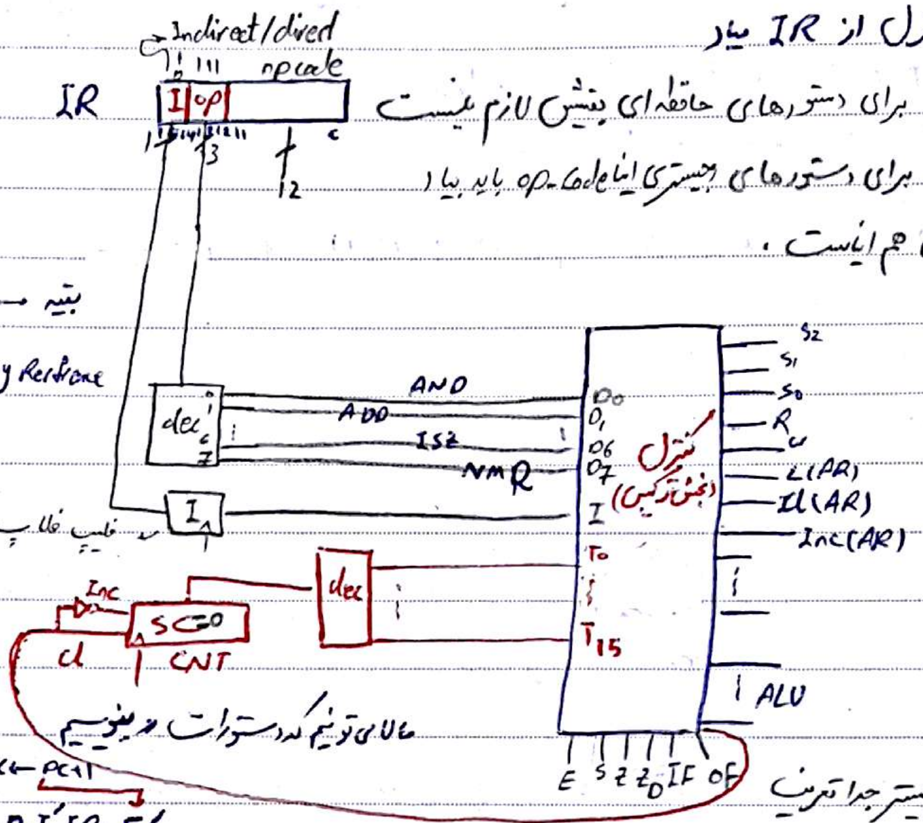
ورودی های کنترل از IR بیاد

برای دستورهای حافظه‌ای بیشش لازم نیست

بیاد برای دستورهای رجیستری اینا op-code باید بیاد

mode هم توی هم اینست.

I/O 1 111
Reg 0 111
memory refrence → بقیه
NMR → not memory Refrence



$D_0 I$ غیرمستقیم AND

$D_0 I'$ مستقیم "

CLA $D_7 I' IR_{11}$

0|111|10---0

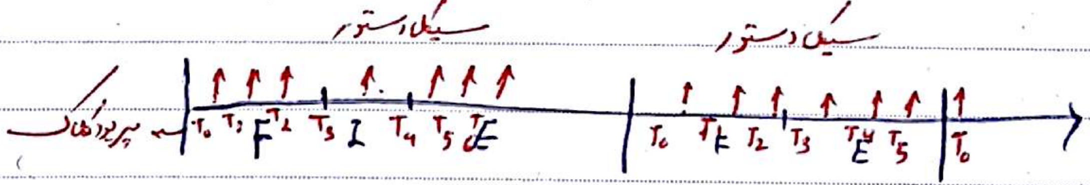
SZE if(E') PC←PC+1

0|111|0---10 → $D_7 I' IR_1 E'$

داسه z و s رجیستر جدا تعریف نکرده (مدارای ترشل تعبی هستند)

برای DR هم z داریم که ببینیم صفره یا نه.

اما اینا کافی نیست باید تایمینگ هارو هم بدونیم که هر مرحله تا منگ چندم؟ باید کلاک هارو بشمریم.



تایمینگ ها می‌زنن تا حالا چندتا کلاک اومدن

یه کانتر می‌ذاریم بشمره کلاک هارو فرض می‌کنیم برای اینجا 4 بیتی به 16 تا تایمینگ بشمره

اینجا dec می‌خوایم cl و Inc داشته باشیم به بعد دوتاش با هم نیست یا cl 0 یا Inc بین آنها NOT هم می‌ذاریمش

زمان دستور آخر همون زمان cl هم هست.

در شروع PC مقدار اولیه داره و بقیه نمی‌خواد SC چون باید تو T0 باشیم یعنی اولین کلاک می‌خواد بیاد.

مقدار اولیه PC و SC رو صفر می‌کنیم.

Subject : Year . Month . Date .

باید m-op بنویسیم یعنی باید کلاک اضافه بشه با ساعت‌ها خط اول با کلاک اضافه نمی‌شه (با ۲ تا می‌شه)

RTL — fetch → $IR \leftarrow M[PC]$, $PC \leftarrow PC+1$

$T_0 : AR \leftarrow PC$, $PC \leftarrow PC+1$, $SC \leftarrow SC+1$

$T_1 : IR \leftarrow M[AR]$, $SC \leftarrow SC+1$

$T_2 : I \leftarrow IR_{15}$, $D_0 \sim D_7 \leftarrow dec(IR(12-14))$ و $SC \leftarrow SC+1$ (m-op نیست)

$AR \leftarrow IR(0-11)$

اگر دستور MR باشه ، به منید $AR \leftarrow AF$ / اگر NMR باشه op نه به عنوان آدرس مطرح نیست

مقدار $D_0 - D_7$ بعد از T_1 بعد از تاخیر dec آماده‌ست چون FF نیست و کلاک نمی‌خواد یعنی op ... مطرح نیست تا لبه داره که تشخیص دستور داره انجام می‌شه ، اما I چون FF می‌باشه با کلاک بعد T_2 فعاله.

$AR \leftarrow IR(0-11)$ هم با یه کلاک با MUX لود می‌شه تو AR

یه دستور مفروض هم هست که دک نمی‌نویسیم دک (قرمزا) به جاش cl در مقطای می‌نویسیم .

پس هر وقت صفر شد می‌نویسیم هر وقت صفر شد نمی‌نویسیم داده اضافه می‌شه .

T_0 یعنی مدار شروع به کار کرده T_1 یعنی PC رفته تو AR ، ... و T_3 یعنی AF توی AR و دستور توی IR و dec هم شده .

T_3 شروع سیکل اجراست برای یکی از دستورها $T_3 D_0$ ← دستوری که برداشته شده بود آدرس‌ش گذاشته شده بود AND بوده

$T_3 D_0$, $T_3 D_1$, ... , $T_3 D_7 IR_8$ (25 جور T_3 داریم)

$$AR = EA = \begin{cases} AF \\ M[AF] \end{cases}$$

می‌دونیم توش AF ه (چون توی T_2 گذاشتیم)

IND — $AR \leftarrow M[AF]$

$T_3 I D_7'$: $AR \leftarrow M[AR]$, $SC \leftarrow SC+1$

دستور و آدرس برداشته شده — غیرمستقیم — حافظه‌ای‌ها — 6 جور

EXE

شروع اجرای دستورات حافظه‌ای : T_4 ← دستور برداشته شده هم EA توی AR

اجرای CLA یا کد 7800 : $AC \leftarrow 0$

CLA $T_3 D_7 I' IR_{11} : AC \leftarrow 0$, $SC \leftarrow 0$ (12 جور)

$T_4 D_0$ ← دستور برداشته شده آدرس موثرش هم محاسبه شده و AND بوده

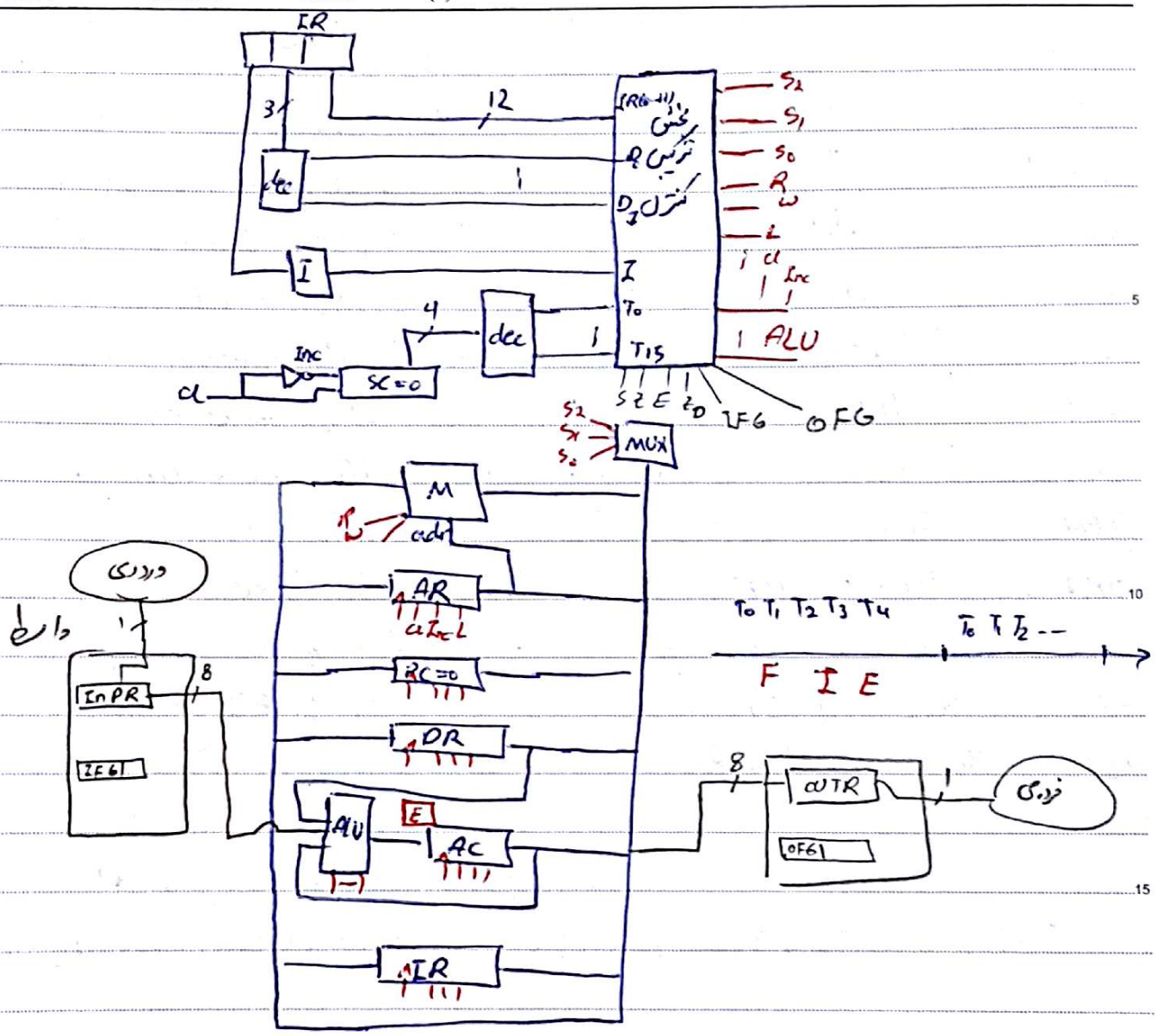
دستورهای رجیستری IND نمی‌خوان با یه کلاک انجام می‌شن .

الان می‌تونیم سیکل‌های دستور رو بنویسیم

Ariyan

Subject :

Year . Month . Date . ( )



Fetch: $IR \leftarrow M[PC],\ PC \leftarrow PC+1, AR \leftarrow AF$

T_0 : $AR \leftarrow PC$, $PC \leftarrow PC+1$ $\quad SC \leftarrow SC+1$

T_1 : $IR \leftarrow M[AR]$ $\quad SC \leftarrow SC+1$

T_2 : $I \leftarrow IR_{15}$, $D_0 \sim D_7 \leftarrow dec\,(IR(12-14))$ $\quad SC \leftarrow SC+1$

$AR \leftarrow IR(0-11)$

درحقیقت AF

T_3

Subject :
Year . Month . Date . ( )

دستورات رجیستری

دستور	کد	اثر	سیکل
CLA	7800	$AC \leftarrow 0$	$T_3 I' D_7 IR_{11}: AC \leftarrow 0, SC \leftarrow 0$
CMA	7400	$AC \leftarrow \overline{AC}$	$T_3 I' D_7 IR_{10}: AC \leftarrow \overline{AC}, SC \leftarrow 0$
CLE	7200	$E \leftarrow 0$	$T_3 I' D_7 IR_9: E \leftarrow 0$ "
CME	7100	$E \leftarrow \overline{E}$	" " " $IR_8: E \leftarrow \overline{E}$ "
INC	7080	$AC \leftarrow AC+1$	" " " $IR_7: AC \leftarrow AC+1$ "
CIR	7040	$E, AC \leftarrow cir\ E, AC$	" " " $IR_6: AC, E \leftarrow cir\ E, A$ "
CIL	7020	$E, AC \leftarrow cil\ E, AC$	" " " $IR_5: E, AC \leftarrow cil\ E, A$ "
SZA	7010	if (Z) $PC \leftarrow PC+1$	" " " $IR_4:$ if (Z) $PC \leftarrow PC+1$ " ک.پ: $T_3 I' D_7 IR_4 Z$
SPA	7008	if (ZS') "	" " " $IR_3:$ if (ZS') " "
SNA	7004	if (S) "	" " " $IR_2:$ if (S) " "
SZE	7002	if (E') "	" " " $IR_1:$ if (E') " "
HLT	7001	توقف سیکل دستور	" " " $IR_0:$ توقف سیکل دستور

الان می‌تونیم تابع INC (PC) رو بنویسیم.

$$INC(PC) = T_0 + T_3 I' D_7 IR_4 Z + T_3 I' D_7 IR_3 Z S' + T_3 I' D_7 IR_2 S + T_3 I' D_7 IR_1 E'$$

نشان Cir و Cil و Comp و اینا توی AC نیست توی ALU، از اونجایی که اینا رو داریم هم به L و AC داده هم به ALU

$S_2 = T_1 + T_2$ $LAR = T_0 + T_2$

$S_1 = T_0 + T_1$ $L_{IR} = T_1$

$S_0 = T_1 + T_2$

$R = T_1$ برای Fetch

$W = 0$

T_3 اگه غیرحافظه‌ای باشه به توجه، حالا اگر غیرحافظه‌ای باشه که سیکل تموم شده

Ariyan

Subject :
Year . Month . Date . ( )

IND $T_3 D_7' I : AR \leftarrow M[AR]$, $SC \leftarrow SC+1$ (AR = AF)

T_2 : $AR \leftarrow AF$, $SC \leftarrow SC+1$

$T_3 D_7' I : AR \leftarrow M[AR]$, $SC \leftarrow SC+1$

$AR = EA \leftarrow \begin{cases} AF \\ M[AF] \end{cases}$ ← EA محاسبه می

رفتیم به T_4 یعنی همه آدرس‌ها داینا آماده است و همچنین دستور حافظه‌ای بوده

T_4 : نه EA تو AR و PC+1 شده و دستور توی IR

T_3 25 جور بود T_4 هفت جوره ($D_0 - D_6$)

AND $AC \leftarrow AC \wedge M[EA]$ اثر

$T_4 D_0$: $DR \leftarrow M[AR]$, $SC \leftarrow SC+1$ (AR = EA)

$T_5 D_0$: $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

ADD $E, AC \leftarrow AC + M[EA]$

$T_4 D_1$: $DR \leftarrow M[AR]$ $SC \leftarrow SC+1$

$T_5 D_1$: $E, AC \leftarrow AC + DR$, $SC \leftarrow 0$

LDA $AC \leftarrow M[EA]$

$T_4 D_2$: $DR \leftarrow M[RA]$ $SC \leftarrow SC+1$

$T_5 D_2$: $AC \leftarrow DR$, $SC \leftarrow 0$

STA $M[EA] \leftarrow AC$

$T_4 D_3$: $M[AR] \leftarrow AC$, $SC \leftarrow 0$

BUN $PC \leftarrow EA$

$T_4 D_4$: $PC \leftarrow AR$, $SC \leftarrow 0$

BSA $M[EA] \leftarrow PC$, $PC \leftarrow EA+1$

$T_4 D_5$: $M[AR] \leftarrow PC$, $AR \leftarrow AR+1$, $SC \leftarrow SC+1$ (AR = EF)

$T_5 D_5$: $PC \leftarrow AR$, $SC \leftarrow 0$

Ariyan

Subject :
Year . Month . Date . ( )

ISZ $M[EA] \leftarrow M[EA]+1$, if $(M[EA]==0)$ $PC \leftarrow PC+1$

$T_4 D_6$: $DR \leftarrow M[AR]$ $SC \leftarrow SC+1$

$T_5 D_6$: $DR \leftarrow DR+1$: $SC \leftarrow SC+1$

$T_6 D_6$: $M[AR] \leftarrow DR$, if (ZD) $PC \leftarrow PC+1$, $SC \leftarrow 0$

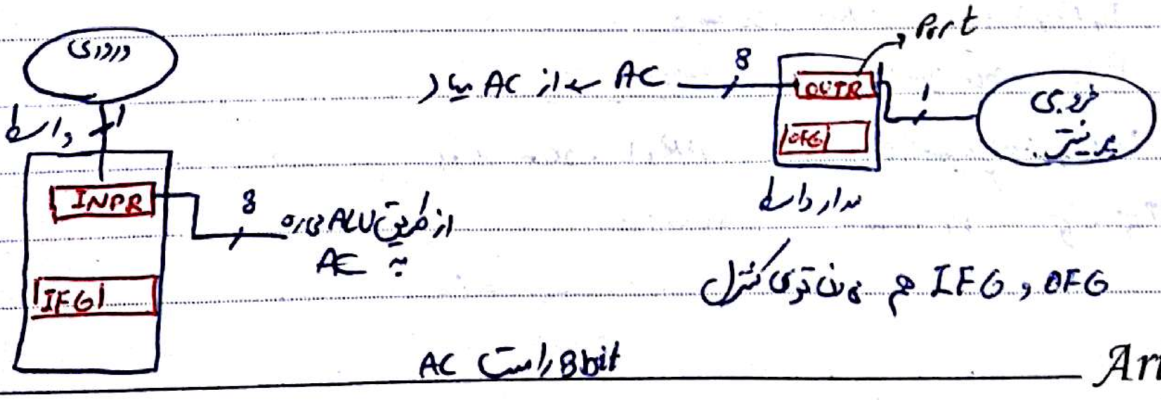
$DR \leftarrow M[AR]$
$AC \leftarrow DR$
$AC \leftarrow AC+1$
$M[AR] \leftarrow AC$
if (Z) $PC \leftarrow PC+1$

اگه می‌خواستیم Inc نداریم برای DR مشکلش اینه که AC عوض می‌شه پس غلطه. برنامه‌نویس به AC دسترسی داره.

اینجا می‌تونیم همزمان از DR بریزیم تو AC و از AC بریزیم توی DR

دستورات I/O

دستور	کد	اثر
INP	F800	$AC(0-7) \leftarrow INPR$, Input Register $IFG \leftarrow 0$ (Input flag)
OUT	F400	$OUTR \leftarrow AC(0-7)$, $OFG \leftarrow 0$ (output flag, output Register)
SKI	F200	if (IFG) $PC \leftarrow PC+1$
SKO	F100	if (OFG) $PC \leftarrow PC+1$
ION	F080	$IEN \leftarrow 1$
IOF	F040	$IEN \leftarrow 0$



IFG و OFG هم فلگ‌های کنترل

AC راست 8bit

Ariyan

Subject :

Year . Month . Date . ( )

INP F800 AC(0-7) ← INPR ; IFG ← 0

$T_3 ID_7 IR_{11}$: AC (0-7) ← INPR , IFG ← 0 , SC ← 0

OUT F400 OUTR ← AC(0-7) ; OFG ← 0

$T_3 ID_7 IR_{10}$: OUTR ← AC(0-7) , OFG ← 0 , SC ← 0

SKI F200 if(IFG) PC ← PC+1

" " " IR_9 : if (IFG) PC ← PC+1 , SC ← 0

SKO F100 if(OFG) PC ← PC+1

" " " IR_8 : if(OFG) " , "

ION F080 IEN ← 1

" " " IR_7 : IEN ← 1 , SC ← 0

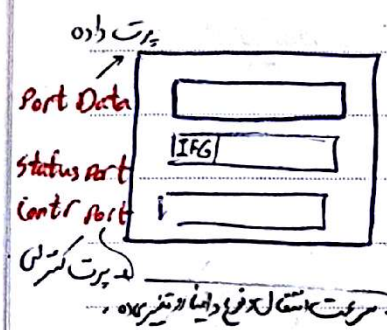
IOF F040 IEN ← 0

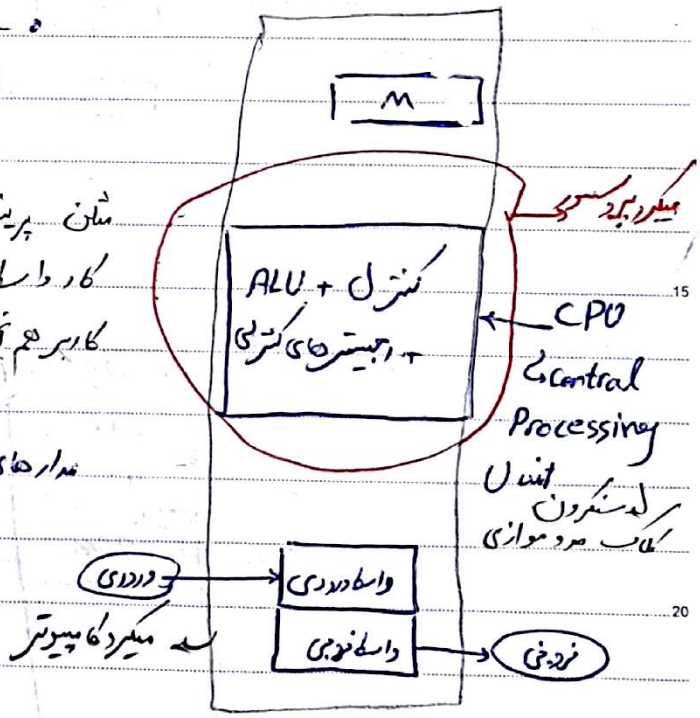
" " " IR_6 : IEN ← 0 , SC ← 0

↳ Interupt enable

مثلاً پرینتر نمی تونه با سرعت کلاک چاپ کنه
کار واسط ها همینست اینارو مدیریت کنه :
کار هم نمی تونه

مدارهای واسط سه نوع رجیستر دارن که بهشون میگیم Port





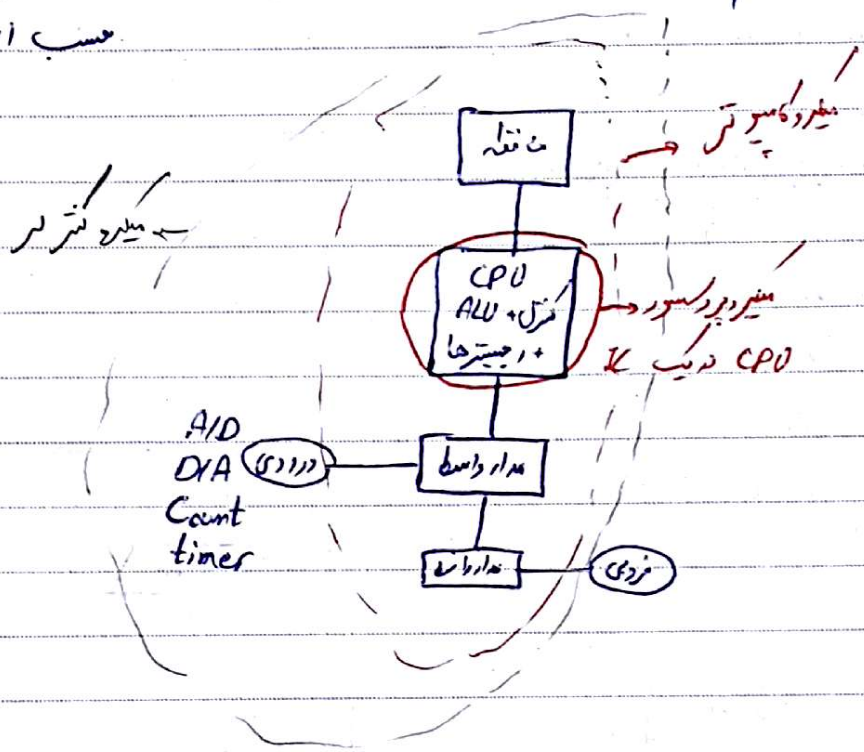
Ariyan

Subject :
Year . Month . Date . ( )

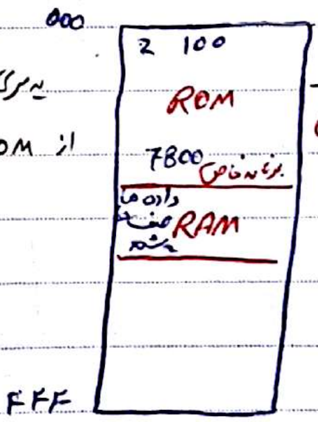
استفاده از یک سیستم { کاربرد خاص / کاربرد عام

زبان های برنامه نویسی ← زبان اسمبلی

← نیازهای برنامه نویسی بر حسب اسمبلی



یه سری برنامه های خاص که تو Ram باید از ROM شروع کنه fetch و exe کنه.



← فقط قابلیت R

فقط یک مسئله خاص ← کاربرد خاص را حل می کند.

کاربرد خاص

کاربرد عام ← مسائل مختلف را حل کند.

Fetch و exe از حافظه اصلی

Subject :
Year . Month . Date . ( )

حافظه های کلی

RAM ↑ ← سرعت ↑

(مدیر منابع) بخشی از سیستم عامل ← ROM

این بخش هر اجرای شده می فهمه پاک می خواهیم.

load می کند توی RAM تست های دستگاه ورودی خروجی که call و RET برنامه ای که می خواهیم و call می کنه داره RET می کنه ← ROM وقتی تموم شد.

command Analyzer

کاربر عام

با مثلث نوع صفحه قبل رد در نظر می گیریم (کاربر خاص)

زبان های برنامه نویسی:

- **زبان ماشین** — باینری — Hex (وابسته به ماشین و استاندارد نیست — دستورات ابتدائی — خوانا نیست)
- **زبان اسمبلی (سمبلیک)** — یک متن (با سمبل های با معنی — خوانا تر شده — استاندارد نیست — وابسته به ماشین — دستورات ابتدائی) مترجم زبان اسمبلی ← اسمبلر
  - اسمبلی ، ماشین: 1 = ADD
  - اسمبلی برای هر پردازنده فرق داره ولی اصول یکیه (مفاهیم)
- **زبان سطح بالا** — استاندارد (مستقل از ماشین) خوانا — به زبان طبیعی نزدیکتر — دستورات کارآ
  مترجم زبان سطح بالا که کامپایلر به زبان ماشین کامپایل می کنه

دستورات اسمبلی در یک سطر

دستورات رجیستری، I/O ← [Lab,] op سمبل [;comment] ← اختیاری
(lable اختیاریه)

حافظه ای مستقیم ← [Lab,] op سمبل سمبل آدرس [;comment]

حافظه ای غیر مستقیم ← [Lab,] op سمبل سمبل آدرس I [;comment]
(I ← سمبل مد)

← 25 دستور

Ariyan

0101
1011

Subject :
Year . Month . Date . ( )

شبه دستور یا راهنمای مترجم ← END , ORG , Hex , DEC

راهنمای مترجم ← int A = 0X12 , B = -5 , C = -0X12 , D = 0

اولین دستور ← D = A - B ;

به زبان ماشین و اسمبلی می‌خواهیم بنویسیم .

راهنمای مترجم همونه تزریق صحیح
int اگر float باشه ذک
نیست ترجمش فرق داره ← کار راهنمای مترجم
دو عملوند باید از یک نوع باشند .

هر اسم در زبان های سطح بالا
و اسمبلی یک لیبل

Hex		Hex	
000	LDA	2 101	دستورات
001	CMA	7 400	
002	INC	7 100	
003	ADD	1 100	
004	STA	3 103	
005	HLT	7 001	
100	A	0012	داده‌ها
101	B	FFFB	
102	C		
103	D		
FFF			

$A - B = A + (\bar{B} + 1)$

نوعاً HLT استفاده می‌شه . 0010 0001

$0X12 = (12)_{16} = 18$

اگر بنویسیم C = 022 ← اکتال

$C = (22)_8 = 010\ 010$

توی حافظه باید Hex وارد شه ← این کار رو مترجم انجام میده .

منفی‌ها باید جاش یه عدد می‌شه مکمل 2 ها

-0X12 = FFEE

دستور	کد
AND	0000
ADD	1000
⋮	
ISZ	6000

← اسمبلر از اینا استفاده می‌کنه .

دستور	کد
CLA	7800
CMA	7400
⋮	
HLT	7001

دستور	کد
INP	F800
OUT	F400
⋮	
IOF	F040

برای یکنواختی توی op های 3bit همه AF ها رو صفر می‌بینیم که 16 بیتی بشه (I هم همین کار رو می‌کنیم .)

| x | op | xxxx |
0 000

سمبل مد	ترجمه
I	8000

| I | xxxxx |
0 0 — 0

حالا خوبی این کار چیه ؟ ترجمه دستور می‌شه OR این ترجمه‌ها (مثلاً سمبل‌ها رو با هم OR می‌کنه .)

Ariyan

Subject :
Year . Month . Date . ( )

جدول سمبل آدرس ها	ترجمه
A	0100
B	0101
C	0102
D	0103
L_1	0001

اسمبلر در مرور اول این رو تولید می‌کنه.

در مرور دوم دستورات روی بینه.

حالا به زبان اسمبلی بنویسیم:

```
        ORG 0
0  →    LDA B      / AC ← M[101]
L1,     CMA        / AC ← AC'
        INC        / AC ← AC+1
        ADD A      / AC ← AC+M[100]
        STA D      / M[103] ← AC
        HLT
        BUN L1
        ORG 100 Hex
A,      Hex 12
B,      DEC -5
C,      Hex -12
        END
```

کامنت برای خودمونه. ترجمه نمیشه.

حال label پیدا شدن اگه بعد HLT توی حافظه خونه 6 برو 4001

BUN ← برای خونه 001 باید اسم بذاریم (اسم خونه‌ای که دستور توشه) ترتیز

لازمش نداره ترجمه دوم

قبل ORG0 بیاد

خونه 100 ← A

اونایی که می‌خوام بهشون رجوع کنیم براشون لیبل بذاریم.

این همه چی زبان ماشین رو داره ولی نمی‌گه الان A, B, ... لیبلن!

نمی‌فهمیم دستورا لیبلن از origin استفاده می‌کنیم بعد origin دیگه ترتیب داریم.

END ← می‌فهمه تموم شد.

از ORG Hex 100 ← 200 بشه A میره تو 200، B 201، C توی 202 و D توی 203

	Hex
AND	0000
⋮	
ISZ	6000

	Hex
CLA	7800
⋮	
HLT	7001

	Hex
INP	F800
⋮	
IOF	F040

	Hex
I	8000

سمبل آدرس ها	
A	0100
L	0102
B	0104
C	0200

شبه دستور
ORG
Hex
DEC
END

Ariyan

تکمیل ادرس‌ها سه صفرهای سمت راست باقی می‌مونه
اولین غیر صفر از (۴) کم می‌شه بعد یا از ۳

Subject :
Year . Month . Date . ( )

ORG 100
A, Hex -10
ISZ A I / Comment
L1, CLA
HLT — اینجا کاربرد معنی کاری نداریم فقط ترجمه می‌کنیم.
B, DEC -10
AND B — تعریف نشده است خطا می‌گیره درستش می‌کنیم
ORG 200 — تکرار بشه یا تعریف نشه
C, STA C — خطا error می‌ده تعریف شده داستفاده نشده warning می‌ده
END

آدرس Hex	محتوای Hex
A 100	FFF0
101	E900
L1, 102	7800
103	7001
B, 104	FFF6 → -10 Hex
105	0104
200	3200

6000 + 0100 + 8000 = E100

6000 + 0104 = 0104

3000 + 0200 = 3200

000A − 0010 = FFF0

000A

اگه بالایی هم ORG200 بود توی خونه 200 مقدار STA C رو می‌ریزه
اگه بگه اینو اجرا کن نمی‌تونیم چون خونه اول حافظه صفره ما از 100 شروع کردیم اگرم بگه PC=100 باز نمی‌شه چون مقدار خونه 100 (FFF0) توی لیست دستورات نیست.

نیازهای برنامه‌نویس:
- انتقال — ادبا کری
- حسابی [ADD-SUB-ADC-SBB / INC-DEC-MUL-DIV]
- منطقی [AND-OR-XOR-NOT]
- شیفت
- حلقه‌ها
- بردارها
- زیربرنامه‌ها به همراه پارامترها
- ورودی-خروجی

A=B:
- از حافظه به رجیستر: LDA A / AC=A
- از رجیستر به حافظه: STA B / B=AC

int A,B,C; C=A+B
```
LDA A
ADD B
STA C
```

C=A-B
```
LDA B
CMA
INC / AC=-B
ADD A
STA C
```

++A
```
LDA A
INC
STA A
```

--A
```
LDA A

STA A
```

Ariyan

Subject :

Year . Month . Date . ( )

long int A, B, C = 0X 12345678 حالا انه بود

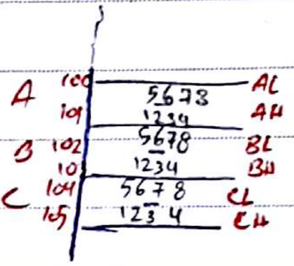
A = B + C

Carry

BH BL

CH CL

AH AL



LDA BL

ADD CL / E, AC = BL + CL

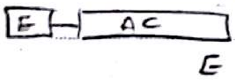
STA AL

LDA BH	CLA
ADC CH	Cil
STA AH	ADD BH
	ADD CH
	STA AH

اینو نداریم ← (LDA BH, ADC CH, STA AH)

عاطع E

Cil → E رو می بره اول AC



A − B = A + B̄ + 1 اینو بخوایم بنویسیم خود B نباید عوض شه.

Carry

AH | AL

BH̄ | BL̄ + → SBB

A + B̄ 1 هم دیگه اضافه می خواد

int C = A & B

LDA	B
AND	A
STA	C

اگه long بشه H و L داره

int کری و اینام دیگه نداره.

A = ~A

LDA	A
CMA	
STA	A

$A = B | C = \overline{(\bar{B} \wedge \bar{C})}$

LDA	B	
CMA		/ AC = B̄
STA	T	/ T = B̄
LDA	C	
CMA		
AND	T	
CMA		
STA	A	

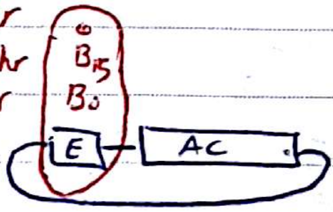
Shr

Ashr

Cir

Cir

Cil

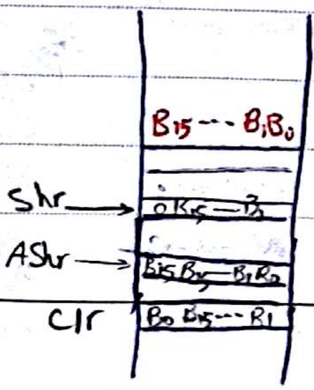


int A, B

A = Shr B   A = Ashl B

A = Shl B   A = Cir B

A = Ashr B   A = Cil B



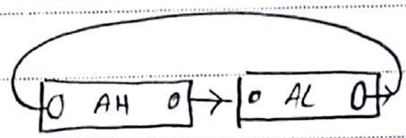
Ariyan

Subject :
Year . Month . Date . ( )

A = Ashr B →
- LDA B
- Cil / E = B_{15} → [B_{14} ---- B_0 E] AC
- LDA B → [B_{15} -- B_0]
- clr / AC = Ashr B
- STA A

ClE داریم و CME داریم.

long int بود =>



int A = 0X12 (↳ 8bit), B = 0X34 (↳ 8bit), P = 0 (↳ 16 bit)

P = A * B

A
B

[0 ---- 00 | A_7 --- A_0] × B_0

A_7 | A_6 --- A_1 0 × B_1

$A_7 A_6$ | --- A_1 0 0 × B_2

0 A_7 --- A_1 | A_0 ---- 0 × B_7

+

در هر مرحله A رو شیفت می‌دیم بعد B رو چک کنیم اگر ۱ بود شیفت یافته A رو اضافه می‌کنیم اگر صفر بود چیزی رو اضافه نمی‌کنیم.

P_0 [ ]

```
ORG 200            ORG 0
P, Hex 0           L1, LDA B
A, Hex 12          Cir   / E = B0 , B1 , ... , B7   (بار اول)
B, Hex 34          STA B
CNT, DEC -8        SZE
                   BUN one
                   BUN ZER
                   one, LDA P
                        ADD A
                        STA P
                   ZER, CLE
                        LDA A
                        Cil
                        STA A
                        ISZ CNT
                        BUN L1
                        HLT
```

CNT, DEC -8 ← چون 8 بیتی است.

STA P ← صفر بوده دیگه انجام نمی‌شه اما باید شیفت بخوره.

اگه قرار بود A عوض نشه STA می‌کردیم توی x ای اولش A رو هم توی x می‌ریختیم.
↳ LDA A
STA x

Subject :
Year . Month . Date . ( )

$P = \begin{cases} PL \\ PH \end{cases}$

حالا اگه 16x16 بیت بشه که P 32 بیتی بشه

AH | AL
[0 | A15 ... A0] x B0

P, Hex 0 {PL, PH}
0 ---- A15 A14 ---- A0 0 x B1

A, Hex {AL, AH}

B, Hex
0 A15 --- A1 A0 --- 0 x B15

CNT, DEC -16

شیفت A

فقط قسمت جمع و شیفت تغییر می‌کنه چون 32bit می‌شه.
AshlA

$P = A \times B_0 + A \times 2B_1 + A \times 2^2 B_2 + \dots$

این مدار برای ضرب unsigned درسته ولی برای signed غلطه چون

اگه می‌خواستیم ضرب signed باشه به A کاری نداریم اگه B مثبت باشه همینه اگه منفی شه هم B هم A رو تغییر علامت می‌دیم. دوباره همین.

در زبان C:

نیازهای برنامه‌نویسی: انتقال، حسابی، منطقی، شیفت، حلقه، پردازه‌ها، زیربرنامه‌ها، ورودی خروجی

```
int A=[10]={1,2,3,...,10}  ,S,I;     <- شبیه دستور
S=0; P=A
for(I=0;I<10;I++)
  S=S+A[I];  !  S=S+*P; ++P;
```

زبان اسمبلی

$\&A[0] \equiv A = 100$ توی اینجا

$\&A[9] = A + 9 = 109$

```
      ORG 100H     <- اسم برداره آدرس شروع
A[0]  A, DEC 1           CLA /AC=0        CLA
         DEC 2           Lop, ADD ? /A[I]  STA S
         :               ISZ CNT          Lop, LDA S
A[9]     DEC 10          BUN Lop          ADD (Pt) I /A[I]
      S, Hex 0           STA S            STA S      { LDA Pt
      CNT, DEC -10                                     INC       } P++
      Pt, Hex 100                                      STA Pt
                                          ISZ CNT
                                          BUN Lop
```

فعلاً اینجا بزن

برای علامت سؤال یه پوینتر تعریف می‌کنیم. که می‌شه صفحه بعد

Ariyan

Subject :
Year . Month . Date . ( )

```
ORG 100H          CLA           /AC←0
A, DEC 1          lop, ADD Pt I /A[I]
   DEC 2               ISZ Pt   /Inc P
   ⋮                 - ISZ CNT
109 DEC 10             BUN Lop
10A S, Hex 0           STA S
10B CNT, DEC -10
10C Pt, Hex 100
```

الان هر مرحله باید Pt INC شه

اگه بخوایم باید LDA، INC و STA کنیم

توی نحوه نوشتن فرض مشکل نداره

ولی اینجا نمی‌شه. اما از ISZ استفاده

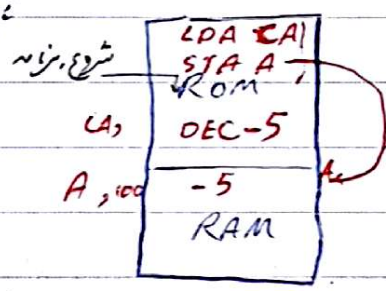
می‌کنیم. چون P آدرسه و مثبت صفر نمی‌شه

هیچ وقت skip نمی‌شه مگه اینکه خونه

های آخر باشد.

حالا برای مقدار اولیه چی؟

ثابت های برنامه رو می‌تونیم توی ROM بذاریم.



می‌شه با mod سیستم هم نوشت.

```
CLA
Lop, ADD A  =>  1100 ⇒(بعدی) 1101 ⇒(بعدی) 1102 ...
ISZ Lop
ISZ CNT
BUN Lop
STA S
```

ولی این مطلوب نیست

ولی به شکل توری درست می‌شه. ولی می‌دونیم دستور رو در حین اجرا عوض کنیم و نامطلوب می‌شه.

اگه دستورات توی RAM باشند. که معمولا نیستند.

```
A, DEC 5        -5 ←
Negate A → NA, Hex 0
B, Hex 20
NB, Hex 0
```

می‌خوایم یه زیربرنامه بنویسیم A رو بگیره منفی‌اش کرده بریزه

زیربرنامه‌ها:
- لینک آدرس
- مبادله پارامترها
  - global، external، متغیرهای خارجی
  - رجیسترها
  - آدرس بلوک پارامترها
  - پشته

7 ⟷ 12

Subject :
Year . Month Date . ( ) (اسم‌ها آدرس)

LDA A
STA X

(مبادله پارامتر)

10 BSA NEG — 15100 آدرس شروع روتین — ORG 100

11 LDA Y
STA NA

NEG, Hex 0 / 11
LDA X
CMA
INC
STA Y

حال مسئله مبادله این پارامترهاست که A در بین A - روی‌ها

LDA B
STA X
20 BSA NEG — متغیرهای یک تابع

21 LDA Y
STA NB

(مبادله پارامتر)

تعریف می‌شن و توی همون

تابع استفاده می‌شن متغیرهای local، محلی

BSA OR

BUN NEG I / C 100 = PC ← M[100]

ORG 200

global extern متغیر خارجی — متغیرها: بقیه متغیرهای global هستند.

OR, Hex 0

الان اینجا A و B local هستند.

BUN OR I

مبادله باید جایی انجام بشه که چنینشون دسترسی دارن ← متغیرها باید global باشند.

ORG 300
GLOBAL
X, Hex 0 ⟶ معمولاً متغیر ورودی می‌ره توی x خروجی
Y, Hex 0 — می‌ره توی y

معادل C

or رو به عنوان تمرین بزنیم — دو تا ورودی می‌خواد

```c
int x, y;
int main ( )
{ int A, NA, ...
x=A; NEG(); NA=y
x=B; NEG() ; NB=y
}
void NEG ( )
{
y=-x;
}
```

به طور کلی استفاده از extern مطلوب نیست (مثل goto)

حالت دیگر استفاده از رجیسترها

Ariyan

Subject :
Year . Month . Date . ( )

از AC استفاده میکنیم

```
int NEG(int x)
{ return -x;
}
```

مبادله ← LDA A / BSA NEG / STA NA

ما اینجا چون یه رجیستر داریم برنامه هایی که نیاز به یه پارامتر دارن رو میتونیم بنویسیم

راه دوم آدرس بلوک پارامترها

```
ORG 100          1 آدرس بلوک ← AB1, Hex 100
                 2 آدرس بلوک ← AB2, Hex 103
A,  ___
B,  ___          (بلوک پارامتر)
OAB, ___
C,  ___
D,  ___
OCD, ___
```

ما نمی‌گیم جواب و بذاره خودش سرجاش

```
LDA AB1   / AC=100
BSA OR
  ⋮
LDA AB2
BSA OR
```

$OAB = \overline{\overline{A} \wedge \overline{B}}$

```
ORG 200 H
OR, Hex 0
STA P        ← متغیر local
LDA P I      / AC=A,C
CMA
STA T        ← متغیر local
ISZ P
LDA P I      / AC=B,D
CMA          / AC=B̄,D̄
AND T
CMA
ISZ P
STA P I
BUN OR I
P, Hex 0     / 100, 103
T, Hex 0     / Ā, C̄
```

حالا اگه یه بردار داشتیم به مقدیرش رو میخواستیم. SUM(A,5) ← تعداد خونه‌ها ، آدرس شروع

```
AC ← آدرس      ORG 50
AA, Hex 100
LA, Dec 5
SA, Hex          ← sum A
AB, Hex 200
LB, DEC 15
SB, Hex 0
AB1, Hex 50
AB2, Hex 53
```

```
LDA AB1  /AC=50
BSA SUM
  ⋮
LDA AB2  /AC=103
BSA SUM
```

ادامه صفحه بعد

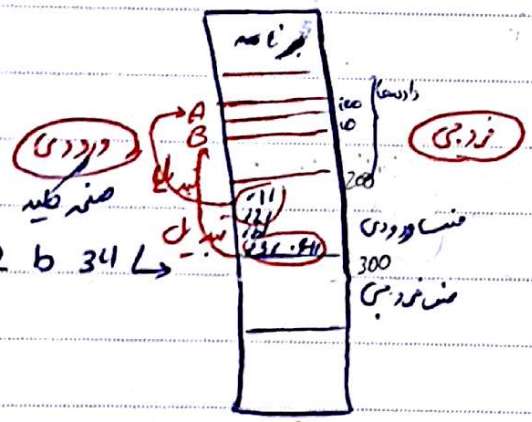
Subject :

Year . Month . Date . ( )

```
        ORG 300
SUM,    HEX 0
        STA P
        LDA P I   /AC=100,200
        STA AAR
        ISZ P
        LDA P I   /AC=5,15
        CMA
        INC       /AC=-5,-15
        STA CNT
        CLA
==
```

ادامه ⟹

```
Lop,    ADD AAR I
        ISZ AAR
        ISZ CNT
        BUN Lop
        ISZ P
        STA P I
        BUN SUM I
CNT,    HEX 0
P,      HEX 0
AAR,    HEX 0
```



text file

ASCII

```
int A, B, C ;
:
scanf("%d%x", &A, &B) ;
:
Printf ("%d", c)
```

به طور متوسط 5 کلاک برای هر دستور

Clk 1 MHz 200 KIPS → Kilo Instruction per second

Clk 10 MHz 2 MIPS

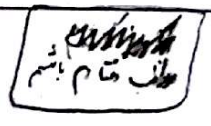
ورودی و خروجی گفته به کلاک انجام نمی شه نمی تونه سنکرون باشه.

توی ورودی و خروجی دستورات ASCII توی حافظه باینری

در ورودی تبدیل از ASCII به باینری مثل float_2

در خروجی تبدیل از نرمالیزه (باینری ....) به ASCII

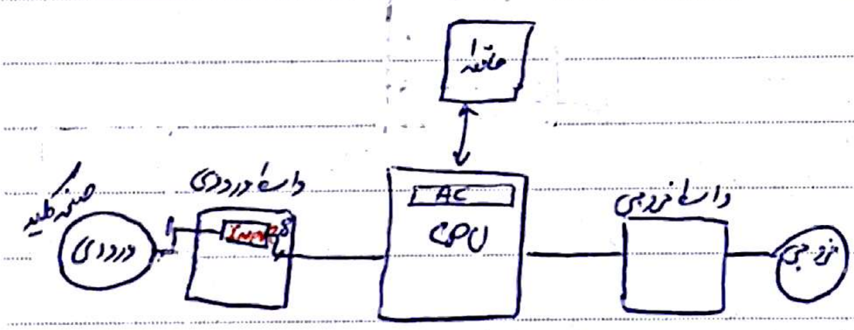
Ariyan

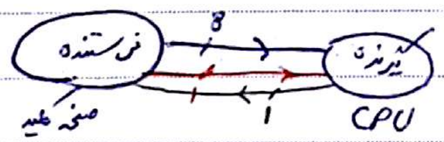


Subject :
Year . Month . Date . ( )

روش های I/O \ I/O	چک آمادگی: ورودی / خروجی	خواندن (ورودی) / نوشتن (خروجی)	انتقال به صف ورودی	انتقال از صف خروجی	پردازش: ورودی / خروجی	ترشوت در طول و کد سری یا موازی
Prog I/O ← دستور (CPU بار ورودی)	IFG, OFG (CPU), SKI, SKO, BUN, BUN	INPR, OUT/TR (CPU), INP, OUT	CPU, STA	CPU, LDA	تبدیل ورودی / تبدیل خروجی، دستورات لازم (CPU)، ADD ... AND	سری به موازی و بالعکس، Pack و unpack → CPU
INT driven	سیگنال سیکل وقفه، واسط، IOF و ION	"	"	"	"	"
DMA (Direct memory Access)	واسط	واسط	واسط	واسط	"	واسط
I/O Processer	"	"	"	"	واسط	"

در مدار واسط توسعه کارهای CPU هم کم می‌شود.





چجوری اینا باید باشن تبادل مطمئنی داشته باشه چیزی گم نشه.

چگونه به شکل Hand shake باشه bit یه که یعنی گذاشتم در دار (وتریز)
گیرنده هم باید با 1bit بگه که برداشتم. (امداد)

با یه راه دیگه Access time که بعد یه تایی ممکن اطلاعات هست باشن DRAM باشه
با سیگنال wait هر وقت wait صفر شد آماده است اما اگه بدونیم چقد طول می‌کشه دستور آماده شه ورودی داریم بعد از زمان

اما برای همه ورودی خروجیا نمی‌شه بیت Hand shake تعریف کرد چون این بیت های زیادی به واحد کنترل. برای همین استفاده نمی‌شه. از یه قرار دادی استفاده می‌کنیم که کار Hand shake رو بکنه

Ariyan

IFG : یه فلگ توی مدار واسط که ۱ شدنش به معنای آماده بودن ورودی است.

فرض کنیم که CPU ورودی داره IFG رو صفر می‌کنه پس هم ورودی هم خروجی باید به IFG دسترسی داشته باشن

پس IFG اگه صفر بود ورودی می‌ذاره و اگه ۱ بود CPU برمی‌داره.

فرستنده : اگر IFG صفر بود می‌فرسته و IF را یک می‌کنه. (ورودی)

گیرنده : اگر IFG یک بود برمی‌دارد و IF را صفر می‌کند. (CPU)

می‌تونه ۰ و ۱ رو برعکس کنیم ما ۱ بودن رو از دید CPU می‌بینیم.

OFG هم مشابه IFG

فرستنده : اگر OFG یک بود می‌فرسته و OFG را صفر می‌کنه. (CPU)

گیرنده : اگر OFG صفر بود برمی‌دارد و OFG را یک می‌کنه. (خروجی)

```
L1, SKI   / if (IFG = 1)  PC ← PC+1
    BUN L1          →   صبر می‌کنی تا کلید بزنه
    INP / AC(0-7) ← INPR , IFG ← 0
    STA 200   / می‌ذاره تو بافر (صف) ورودی /
      ⋮
L2, SKI
    BUN L2
    INP
    STA 201
```

قرارداد مون رو اجرا می‌کنه.

می‌شه بیوفته توی loop تا یه کاراکتر جدید وارد شه.

نه ادامه بدی (به برنامه‌نویسی می‌سازیم که آدرس ادرس رو تونه می‌کنه I کنی)

برای خروجی هم به شکل مشابه

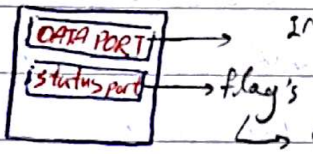
```
L3, SKO  / if (OFG = 1)  PC ← PC+1
    BUN L3
    LDA 300
    OUT / OUTR ← AC(0-7) , OFG ← 0
```

اما باز هم چند تا ورودی باشه باید Register و flag هر کدوم برن به CPU و لازم باید یه کار دیگه انجام بدیم.

رجیسترهای مدار واسط ـ Port

رجیسترهای پردازشی | در CPU

رجیسترهای کمکی



یه PE هم میذارن که ↳ Parity error

یه bit دیگش OW ← over write

بیت های status port باید قابل چک کردن باشه.

نوع سوم که اینجا هست بهش میگن Control port عملکرد دستگاه واسط رو عوض می‌کنه.

(مدار واسط قابل برنامه پذیر باشد. ورودی باشه خروجی باشه با توجه به نرخ تبادل کنترل کرد. جدول داده.

مثه بیت های حافظه روی شه چک کرد اگه بتونی بخونی و بنویسی اشکال نداره تو AC میذاریش یه جای حافظه می‌تونی هر کلمه‌ای رو بخونی بنویسی)

حل مسائل I/O ← قابلیت خواندن و نوشتن Port ها
- دستورات INP و OUT ← I/O ایزوله
- دستورات LD و ST : memory mapped I/O

سیگنال دفعه کار IF6 و OF6 رو انجام میده

مثل مهمون میاد وایمیستیم دم در ببینیم کی میاد ← IF6 و OF6

یا یه زنگ میذاریم به کار خودمون میرسیم هر وقت زنگ زد می‌فهمیم اومده و اتلاف وقت حذف میشه ← INT

دوره کار وقفه انجام میشه به ورودی/خروجی هم سرویس میده.

DMA ، مثال رئیس دانشکده

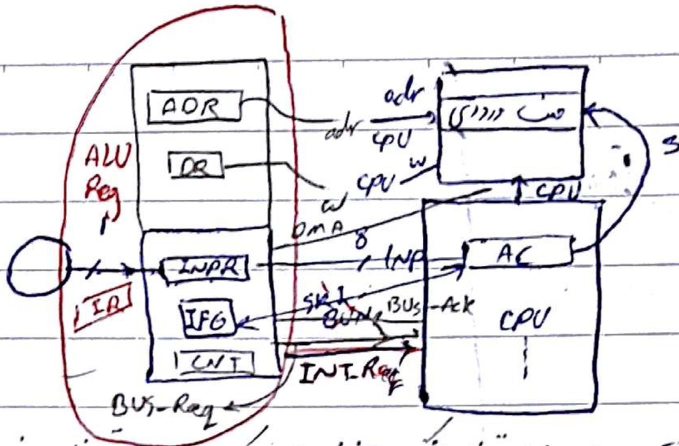
↳ اول مدار واسط مثل منشی هست که کارها رو مرتب کنه برای رئیس

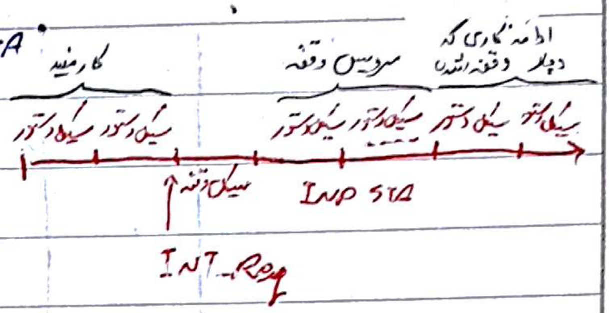
چون مراجعات زیادی داره یه منشی میذاره اون این کارا رو بکنه

I/O processor

مثال رئیس جمهور

↳ خلاصه هم می‌کنه یعنی قسمتی از پردازش هم انجام میده.





وقفه‌های پشت هم مثل call تو در تو مثل داری کتاب می‌خونه تلفن زنگ می‌زنه وقفه تو کتاب داری تلفن حرف می‌زنی زنگ می‌خوره وقفه توی تلفن بعد که در رو باز کردی می‌ری تلفن بعدش می‌ری سر کتاب دوباره.

اما فرقش با call اینه نمی‌دونیم کی call اتفاق می‌افته سیکل وقفه همون call ه.

که اینجا دیگه SKI BUN نداریم.

برای حالت DMA تمام کارهای برداشتن اطلاعات و فرستادن اینا رو مدار واسط انجام می‌ده.

(مدار واسط توسعه پیدا می‌کنه.) اعداد INPR رو OR میکنه می‌چینه توی OR بعد باید بفرسته تو حافظه.

اما نمی‌شه هم CPU وصل باشه هم DMA به ADR

یه سیگنال می‌فرسته BUS_Req که CPU از حافظه بیره بیرون CPU هم یه BUS_Ack می‌فرسته که یعنی درخواست قبول شد من رفتم کنار (به اندازه یه نوشتن تو حافظه باید از حافظه بره کنار.

واسه اینکه اگه وسط نوشتن یا خوندن بود از بین نره کارش.

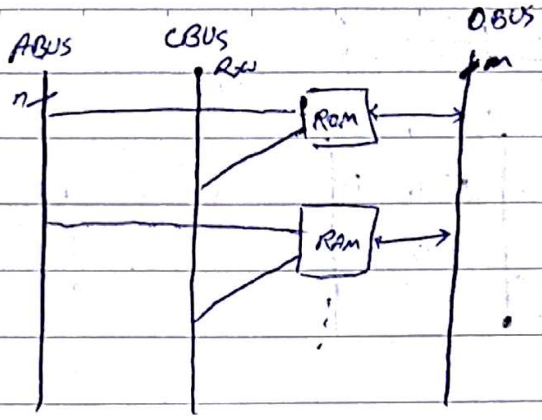
	CPU	CPU	DMA	
BUS_Req	0	1	1	0
BUS_Ak	0	0	1	1

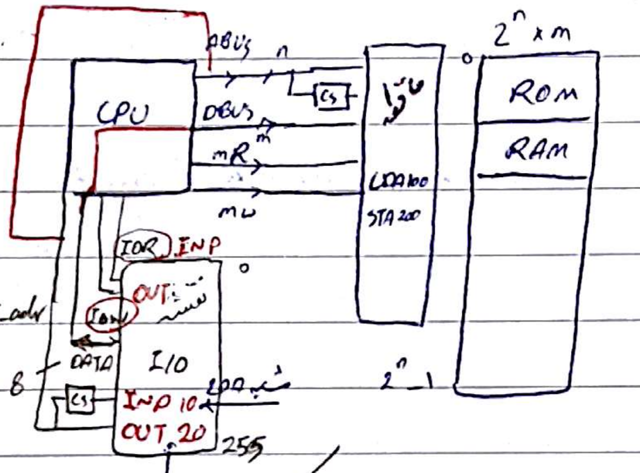
هیچ کدام اینا به حافظه دسترسی ندارن

I/O proc باز هم بیشتر توسعه پیدا می‌کنه ورودی می‌شه یه پروسسور (قرمز)

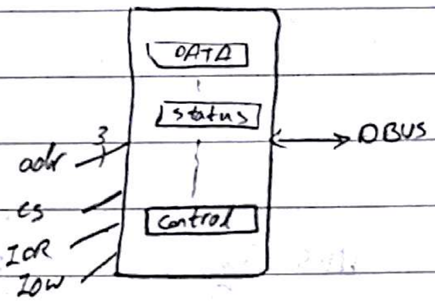
Isolated I/O
memory mapped I/O

ورودی / خروجی به توانایی خواندن و نوشتن پورت‌ها ساده می‌شود.





ساختار مدار واسط مثل همین mem map. ، تعداد کمتر.



همون بحث mapping با تعداد بیت کمتر. مشابه STA

در مثال ما یه Port خروجی یه Port ورودی داریم که آدرس نمی‌خواد. فقط INP داره و یه OUT

فضای ورودی/خروجی و حافظه جدا هستن

I/O map و memory map

Isolated I/O { INP port-adr / OUT Port-adr }

mem mapped I/O — پردازنده INP ندارد، OUT ندارد، IOR و IOW ندارد

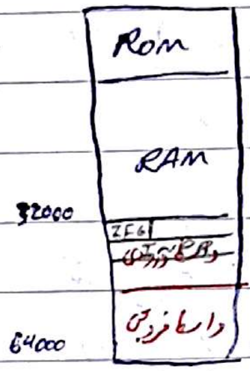
می‌تونه Adr و DATA یکی باشه و IOR و IOW جدا و MW و MR هستند. (قرار) اما یه آدرس 100 بدیم توی جدولش آدرس 100 می‌آید با IOR از ورودی می‌گیریم.

اگه منطقه و اینا نباشه مثلا IFG بیت سمت چپ آدرس 20 باشه OFG بیت راست آدرس 40

خروجی هم ست به

```
L1, SKI                 zer, INP 20
    BUN L1     =>            AND Mask / mask=10...
                             SZA
                             BUN one
                             BUN zer
```

بخشی از حافظه مربوط به واسط ورودی و واسط خروجی



همینا هم ورودی
نه نقشش

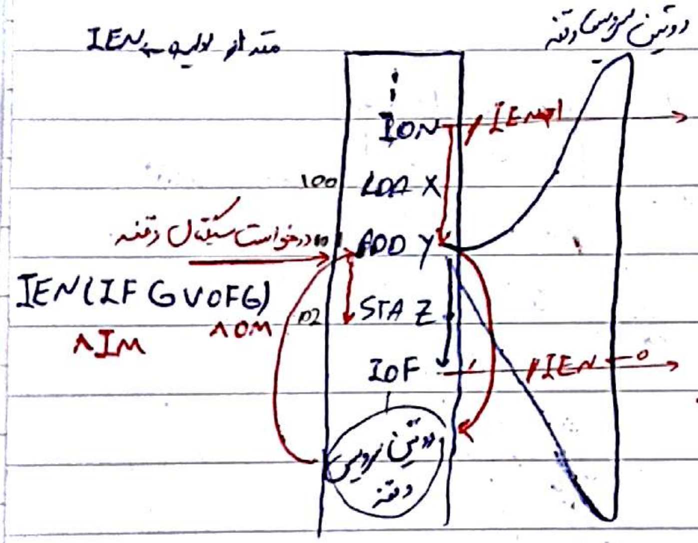
می‌شه

```
ZER, LDA 32000
     AND MASK
     SZA
     BUN one
     BUN ZER
     LDA 32000
```

سیکل وقفه

IFG یا OFG یک شه درخواست وقفه — یک کردن Interrupt enable

می‌ده به CPU

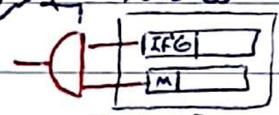


یک کردن Interrupt enable

ادامه صفحه بعد

صفر کردن Interrupt enable

mask توی خودشه — کنترل عوامل وقفه

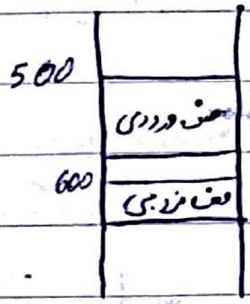


mask می‌تونیم یه INT رو انتخاب کنیم

$(INT_1 \cdot M_1 \wedge INT_2 \vee M_2 \wedge INT_3) M_3$

وقفه‌ها اولویت دو تا کار هست کدوم مهم‌تره

اگر IEN یک باشه ION هم یک باشه وقفه فعال می‌شه. (اونایی که اولویت ورودی خروجی بیشتره ION بکنه ممکنه بین دوتا وقفه هم اولویت قائل شیم)

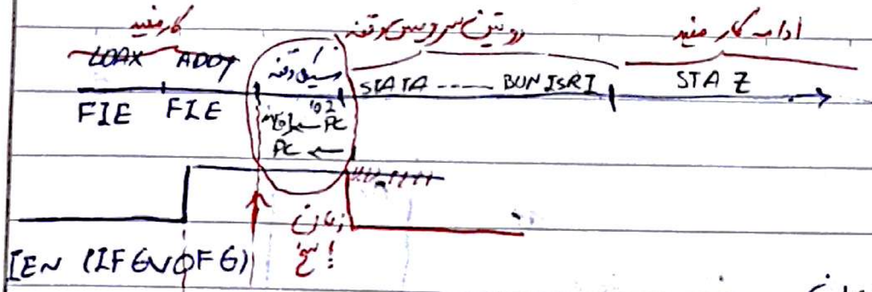


ورودی	خروجی
L1, SKI	L2, SKO
BUN L1	BUN L2
INP	LDA OBP I
STA IBP I	OUT
ISZ IBP	ISZ OBP

سرویس ورودی در مثال ما

سرویس خروجی در مثال ما

IBP, Hex 500   OBP, Hex 600



زمان درخواست!
زمان پاسخ!
سیکل وقفه

- ضبط وضعیت
- تشخیص عامل
- اعمال اولویت سرویس
- بازیابی وضعیت و برگشت

هر موقعی می‌تونه فعال شه پس زمان اومدن سیگنال معلوم نیست اما زمان پاسخ بعد از اتمام یه سیکل دستوریه که توش بودیم. نمیذاریم طول نمی‌کشه ورودی کننده طولانی‌ترین دستور هم هفت تا تایمینگ.

اطلاعات وضعیت
- فاتقلا ← با تخصیص حل می‌شود.
- رجیستر AC ← اگر ورودی AC باشه اینم می‌شه با تخصیص حل کرد.
- فلگ‌ها E ← ضبط و بازیابی (به شرطی که عوض نشوند) (PC که قطعاً عوض می‌شود)
- ترتیب اجرا PC
- رجیسترهای کمکی IR, AR, DR, TC ← چون قرارداد کردیم بعد از سیکل دستور پاسخ بدیم لازم نیست ضبط بشه.

با مشکلات روزم افزاری حل می‌کنیم (به مدار اضافه نمی‌کنیم مگر اینکه مجبور باشیم) که نشه دک کاریش کرد.

ایا تو سرویس چون معلوم نیست وقفه می‌یاد.

بعد از سیکل رفته 102 توش

روتین سرویس وقفه

معمولاً ورودی اولویت بیشتر

بین دو ورودی اونی که تبادل اطلاعات بیشتری داره.

اما تو این مثال می‌تونه E رو ضبط نکنیم.

```
ORG 0
ISR, Hex 0 / 102
STA  TA   200
CIR       201
STA  TE
SKI
BUN L1
INP BSA ISR
STA IBP I
ISZ IBP
L1, SKO
BUN L2
LDA OBP I
OUT
ISZ OBP
L2, LDA TE
CIL
LDA TA
BUN ISR I
```

- برای اینکه ببینیم IFG یا OFG کدوم یک دونه دونه چک می‌کنیم اونی که اولویت بیشتری داره اول چک می‌کنیم.
- ضبط وضعیت: STA TA / CIR / STA TE
- در حالت کلی اینه اما ما اینکار رو نمی‌کنیم خودش روتین ورودی داریم.
- سرویس ورودی: STA IBP I / ISZ IBP
- اگر چیزی جواب نیاز داریم می‌ریم سراغ خروجی
- سرویس خروجی: LDA OBP I / OUT / ISZ OBP
- بازیابی: LDA TE / CIL / LDA TA / BUN ISR I

STA	M[EA] ← AC
BSA	M[EA] ← PC PC ← EA+1

یعنی آخرین دستور قبل وقفه و اولین دستور وقفه PC عوض نشه

آدرس برگشتم حفظ شه

که در سیکل وقفه ← یه کاری کرد کار ما رو می‌کنه اینجا مجبوریم سخت افزاری حل کنیم ← سیکل وقفه (ابتدا)

می‌شه کارای دیگه هم توی سیکل وقفه انجام داد.

سیکل وقفه

$M[\ ]? \leftarrow PC$

$PC \leftarrow ?[\ ]+1$

$0 \leftarrow$ constant

M[constant] یه خونه حافظه رو تخصیص می‌دیم به سیکل وقفه.

BUS

سیکل وقفه:

$M[0] \leftarrow PC$ و $IEN \leftarrow 0$

$PC \leftarrow 0+1$

معادل BSA 0

اولش همیشه از یه ولی می‌شه بهینه‌ش. از 200 شروع شه توی 1 می‌ذاریم BUN 200

اما این یه اشکال داره ما گفتیم نقطه شروع صفره واسه حل این مشکل فرض می‌کنیم نقطه شروع رو دیگه 100 در نظر می‌گیریم.

برای اینکه هیچ ربطی به D و I و E نداشته باشه فرض می‌کنیم تو T_8 و T_9 و T_{10} اجرا شه.

سیکل وقفه: $T_8 \quad T_9 \quad T_{10}$

سیکل وقفه با اثر فوق

$T_8: AR \leftarrow 0$

$T_9: M[AR] \leftarrow PC$ و $PC \leftarrow 0$

$T_{10}: PC \leftarrow PC+1$ و $SC \leftarrow 0$ و $IEN \leftarrow 0$

اما مشکل اینجاست که: ما بعد T_5 سر T_8 ما فرض می‌کردیم دستور بعدی پشت دستور باشه. اما وقتی وقفه داریم بعدی یا دستور یا سیکل وقفه دیگه توی T_0، SC صفر نمی‌شه

$T_4D_1: DR \leftarrow M[AR]$

$T_5D_1: AC \leftarrow DR+AC$ ~~$SC \leftarrow 0$~~ if $(IEN \wedge (IF \vee OF))$ $SC \leftarrow 8$ else $SC \leftarrow 0$ — خود بخش به سیکل وقفه

برای همه دستورات (استثناء ION و IOF)

چون توی تایمینگ آخر گذاشتیم وسط دستور نمی‌پره

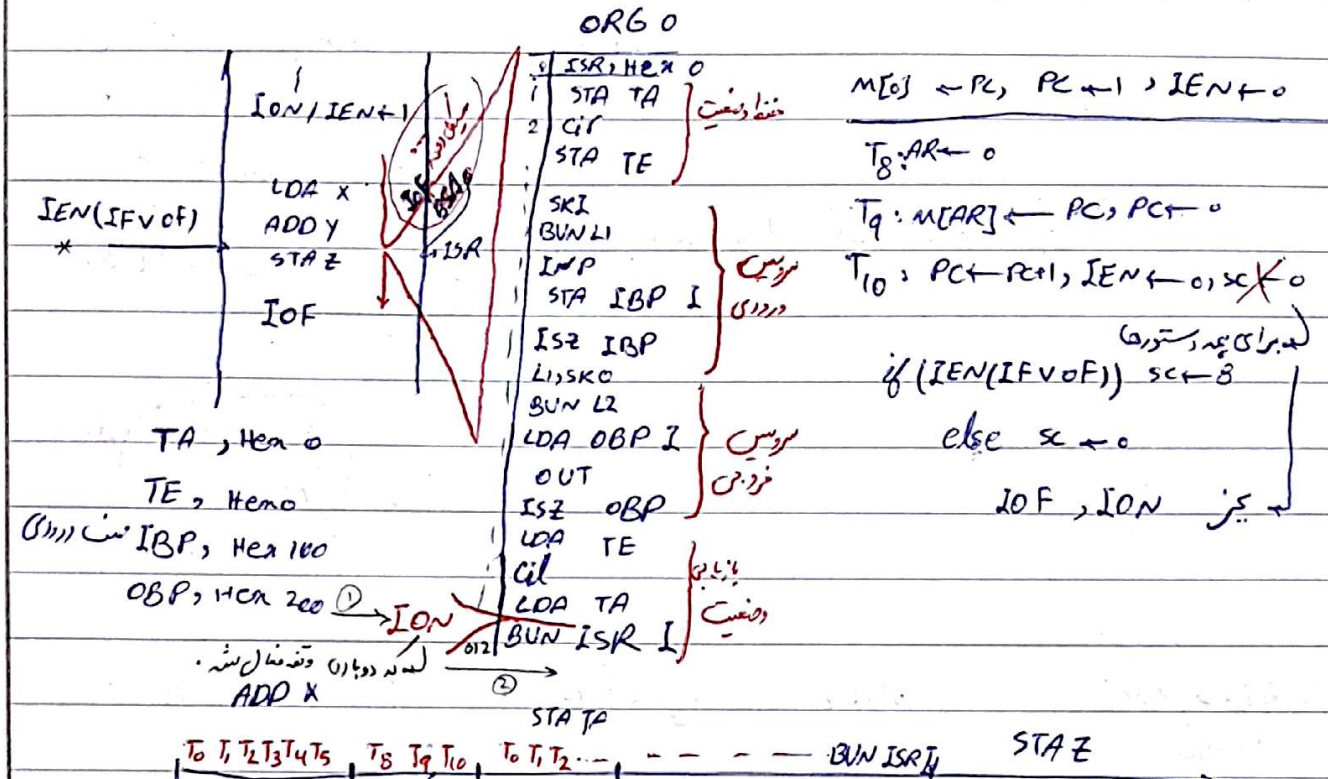
اما باز یه مشکل دیگه هست اون سیگنال $IEN(IFG \vee OFG)$ چک می‌شه تا INP هم / $IFG=0$

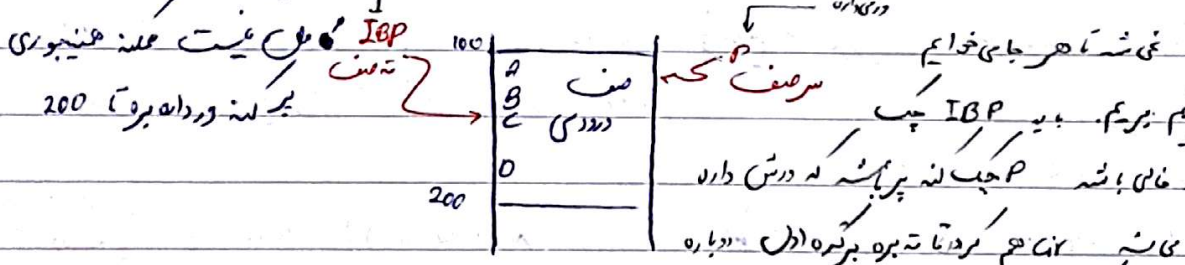
برای همین توی سیکل وقفه یه کار اضافه دیگه هم انجام می‌شه. (تمرین)

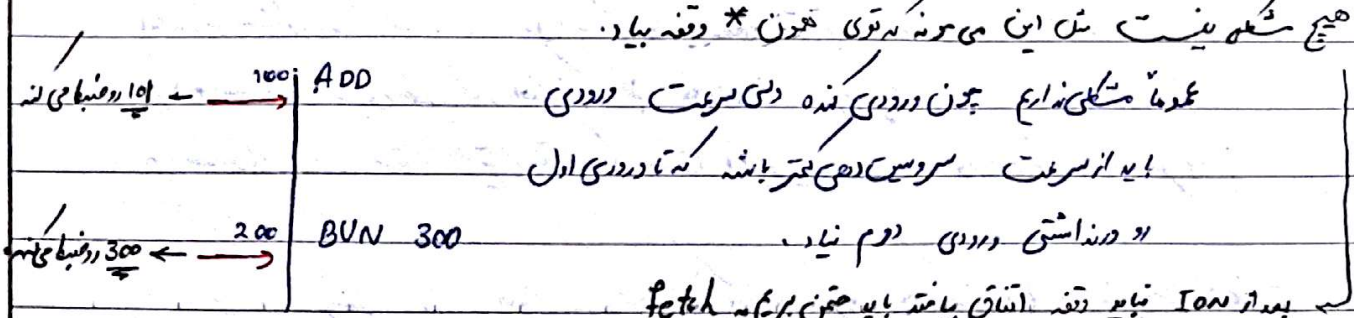
می‌شه توی اولین دستور روتین وقفه هم گذاشت اما چون باید توی خود سیکل وقفه صفر می‌کنیم.

اولویت دارن هم با همون ION و IOF

اما وقتی دوباره برگشتیم وقفه باید فعال شه دوباره. پس اگه وقفه‌پذیر بود بعد برگشت هم وقفه‌پذیر باشه. (وضعیت فعلی هم باید تغییر کنه)







IOF

حالا می‌خوایم بعد IOF وقفه نیاد اگه سیکل

$T_3 D_7 IR_6 : IEN \leftarrow 0 , SC \leftarrow 0$ — اگه این نباشه

مثل ترنزه بازنه اگه بلافاصله

if (IEN ....) $SC \leftarrow 8$

بعدش وقفه بیاد چون IEN توی اینک

else $SC \leftarrow 0$

بعد 0 می‌شه ممکنه SC بشه 8 بره توی وقفه

بعد نه وقفه دوباره ION داریم انگار اون در نظر نگرفتیم IOF رو می‌تونیم مثل آبیه بنویسیم یا ترنزه توی T4 بنویسیم.

پرچم‌ها دیگه اولیه هم مهمه

IFG ← 0 یعنی روشن داریم — روشن نه می‌شه چیزی توی ورودی

OFG ← 1 — روشن نمی‌شه چیزی توش نیست که نفرسته

تا اینکه روشن نشه به خاطر OFG و وقفه روشن می‌شه.

کنترل عوامل وقفه ←
- IEN
- یه mask اختصاصی برای IF
- اختصاصی برای OF (مثل بیت اختصاصی)

اگه یه چیز دیگم بود

$IEN ((INT1 \wedge M1) \vee (INT2 \wedge M2) \vee (M3))$

باید بتواند مقداردهی شود.

باید بتواند چک شود

$IEN (IF \vee OF)$ — IM , OM

الان پردازنده‌ها ندارند الان IM و OM رو

اگه فقط بخوام به ورودی سرویس بدم OM رو صفر می‌کنیم.

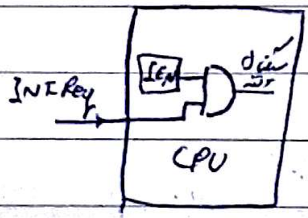
در نتیجه الان اول برنامه خروجی نمی‌تونه باعث وقفه بشه..

اگه ما اول IM رو یک کردیم OM رو صفر بعد وقفه هم باید همین بمونن

الان باید سرویس روتین رو هم تغییر بدیم که اگه OM مثلا صفر بود بهش سرویس نده. اگه mask و flag یک بود سرویس بدیم.

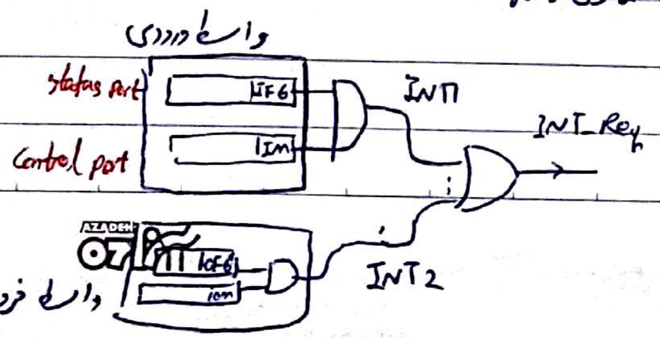
باید همچین چیزی باشه:

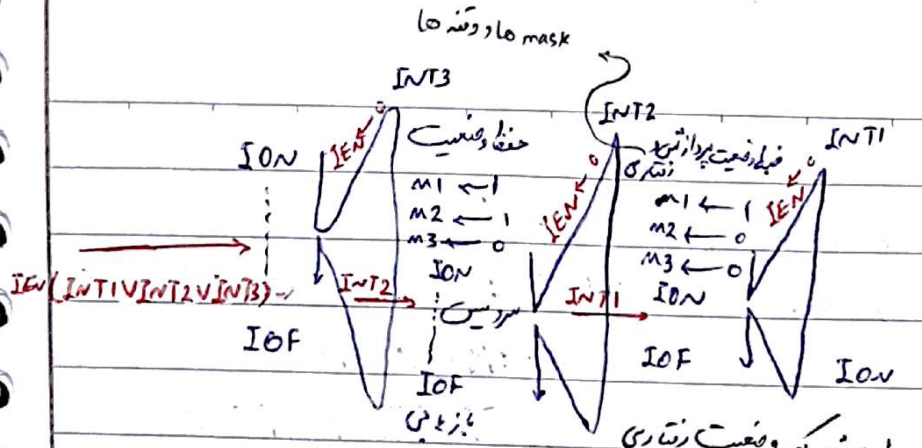
```
SKIM
BUN L1
SKI
BUN L2
```



اینجوری که دستور جدید نمی‌خوایم

IM توی control port

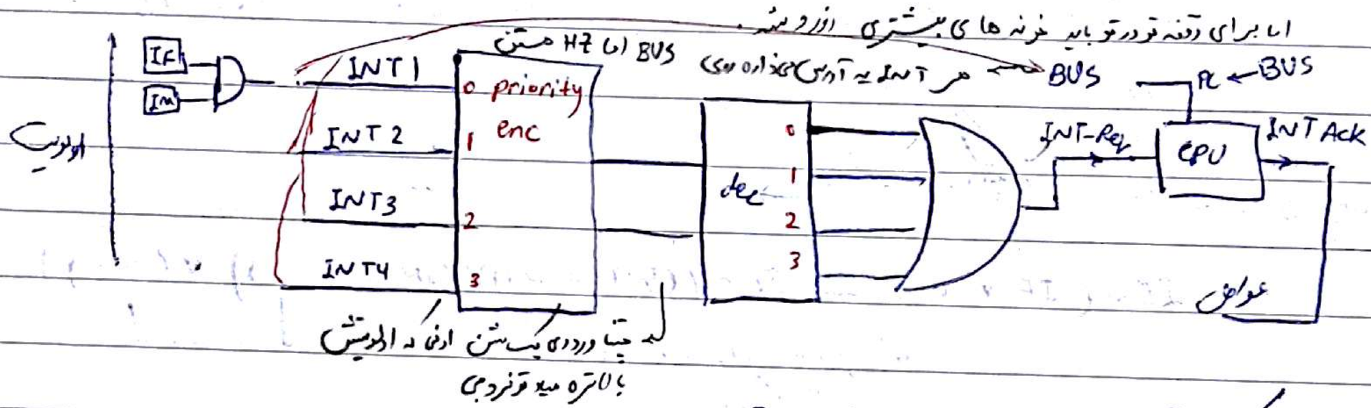




حالا برای وقفه تودرتو
باید اجازه وقفه به اولویت
های بالاتر بدیم

وقتی از INT3 به INT2 می‌ریم باید M1 ، 1 باشه که وضعیت رفتاری
حفظ بشه.

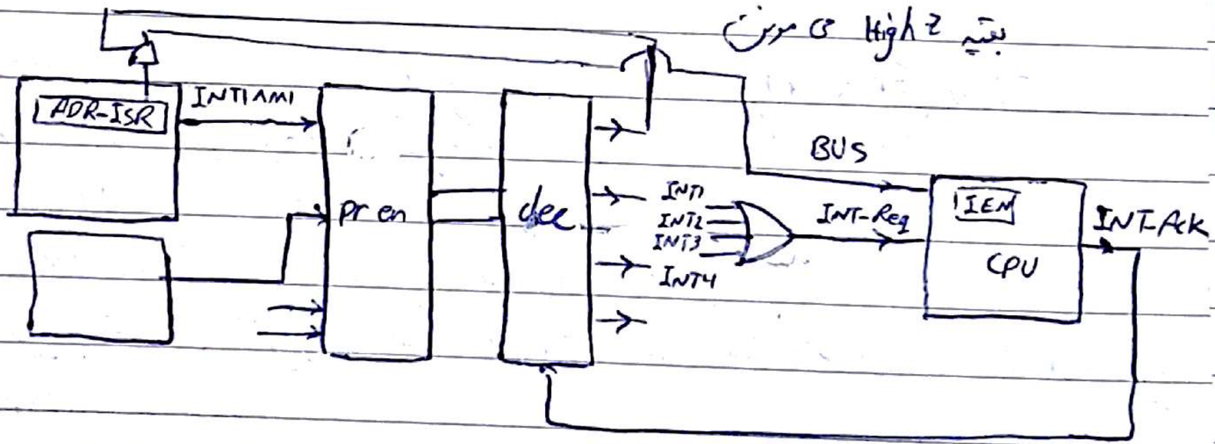
اما برای وقفه تودرتو باید خونه های بیشتری اورد بشه



همه ی چیتا ورودی یک بشن اونی که اولویتش
بالاتره میره تو خروجی

اونی که انتخاب میشه load میشه تو رجیستر PC

بقیه High Z میرن



$M[SP] \leftarrow PC \ , \ SP \leftarrow SP-1$

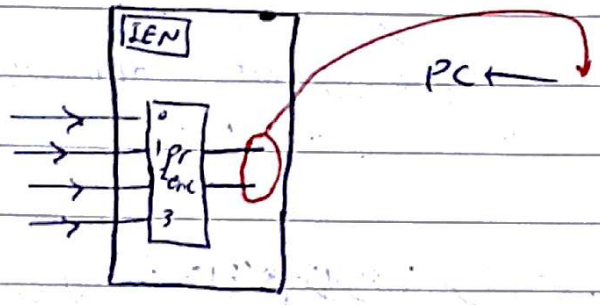
$INT\_Ack \leftarrow 1$

$PC \leftarrow ADR\text{-}ISR$

$INT\_Ack \leftarrow 0 \ , \ IEN \leftarrow 0$

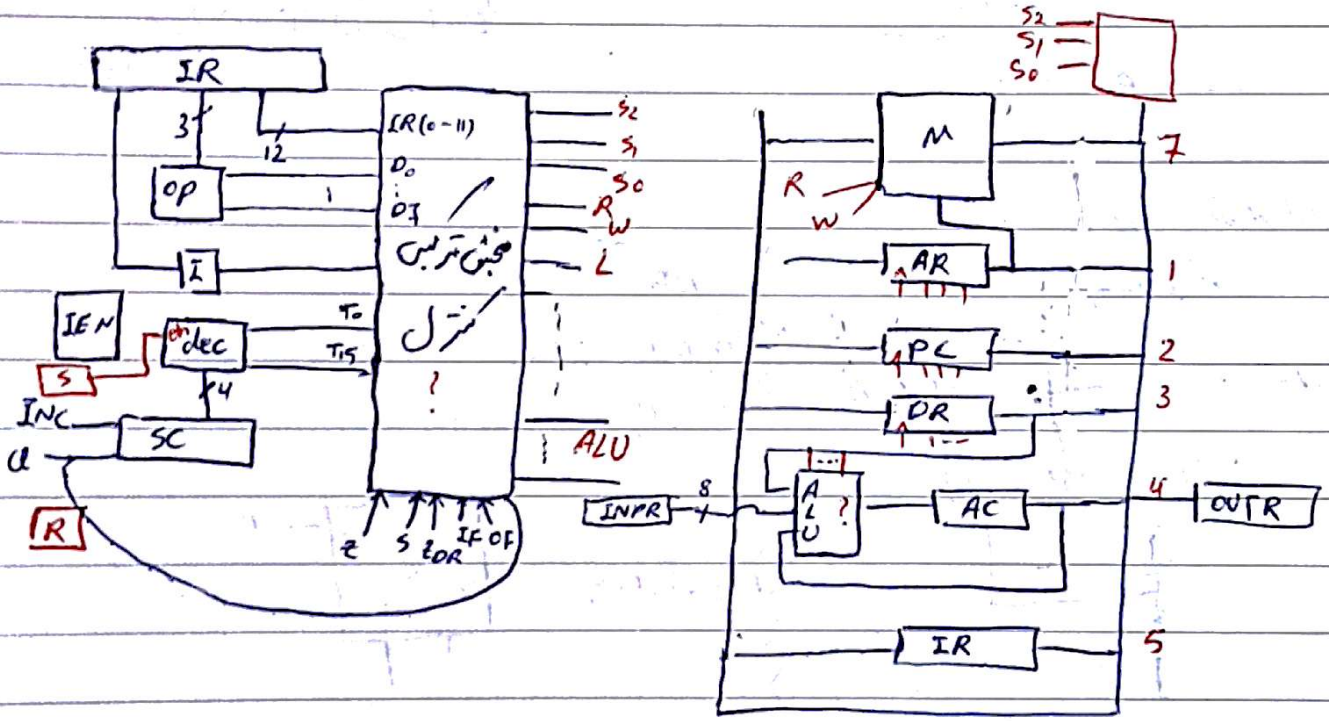
سیگنال وقفه

$IEN \wedge ((I_1 \wedge M1) \vee (I_2 \wedge M2) \dots)$



حال اینجا توی T4 ما داریم BUN به جایی که در تابع وقفه انتخاب شده است.

طرح کامپیوتر ما



Fetch

$R' T_0$: $AR \leftarrow PC$, $PC \leftarrow PC+1$, $sc \leftarrow sc+1$

$R' T_1$: $IR \leftarrow M[AR]$, $sc \leftarrow sc+1$

$R' T_2$: $I \leftarrow IR_{15}$, $AR \leftarrow IR(0-11)$, $sc \leftarrow sc+1$

مقادیر اولیه

$T_2 T_1 T_0 IEN (IF \vee OF) \vee R \leftarrow 1$ توی T_3 و T_4 و ... R وابسته

$SC \leftarrow 0$, $R \leftarrow 1$ ($R' T_0$)

$IEN \leftarrow 0$

$IF \leftarrow 0$

$OF \leftarrow 1$

سیگنال وقفه

$R T_0$: $AR \leftarrow 0$

$R T_1$: $M[AR] \leftarrow PC$, $PC \leftarrow 0$

$R T_2$: $PC \leftarrow PC+1$, $IEN \leftarrow 0$, $SC \leftarrow 0$, $R \leftarrow 0$

$D_1 T_4$: $DR \leftarrow M[AR]$

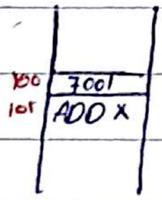
$D_1 T_5$: $AC \leftarrow DR + AC$, $SC \leftarrow 0$

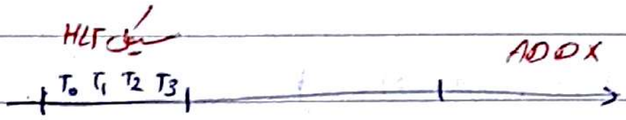
برای اینکه اینها فاصله نشن به بیت R تعریف می‌کنیم.

اگر تو fetch هستیم نباید برم تو وقفه چون fetch به R وابستگی پیدا می‌کنه.

سیگنال وقفه و HLT نرم‌افزاری قبل عمل غیرقابل‌پیش‌بینی

HLT





برای HLT یه bit تعریف می‌کنیم (S) ، enable ، dec

$I'D_7T_3IR_5$: $S \leftarrow 0$, $SC \leftarrow 0$

خروجی از HLT ، هم نرم‌افزاری میسر نیست باید یه سیگنال خارجی بیاد که

سیگنال‌های کلاک میان شون Reset چون sc ها داشته شده $S \leftarrow 1$, $SC \leftarrow 0$ →

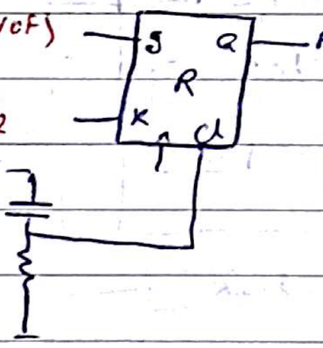
→ Reset

مقدار اولیه $R \leftarrow 0$

$RT_2 \leftarrow 0$

$T_0'T_1'T_2'IEN(IF \vee OF)$ → J

RT_2 → K



$IEN \leftarrow 0$

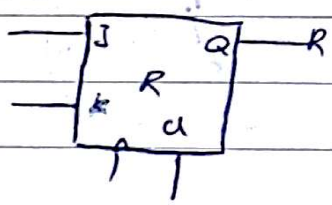
$RT_2 : IEN \leftarrow 0$

$ID_7T_3IR_6 : IEN \leftarrow 0$

$ID_7T_3IR_7 : IEN \leftarrow 1$

$T_3ID_7IR_7$ → J

$RT_2 + T_3ID_7IR_6$ → K



$cl(SC) = D_7T_3 + D_0T_5 + D_1T_5 + D_6T_6 + RT_2 + \ldots$

$L(AR) = R'T_0 + R'T_2$ $\quad$ $cl(PC) = RT_1$

$INC(PC) = R'T_0 + RT_2$

برای fetch و سیگنال وقفه

$S_2 = R'T_2 + R'T_1$

$S_1 = R'T_0 + R'T_1 + RT_1$

$S_0 = R'T_2 + R'T_1$

$R = R'T_1$

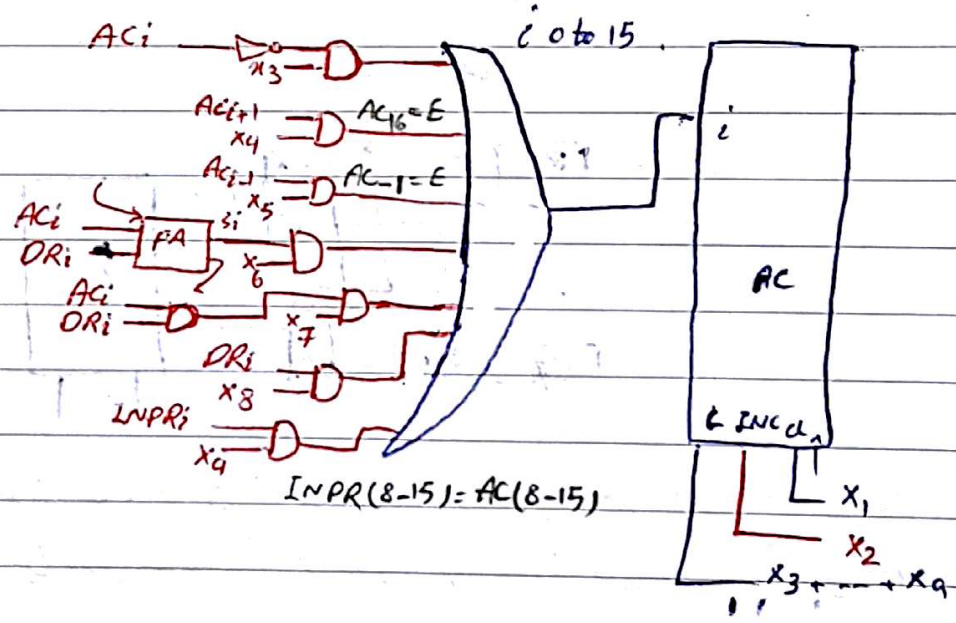
$W = RT_1$

$L(IR) = R'T_1$

CLA	$I'D_7T_3IR_{11}$	x_1: AC ← 0	$CL(AC) = x_1$
Inc		x_2: AC ← AC+1	$INC(AC) = x_2$
CMA		x_3: AC ← $\overline{AC}$	$CL(AC) = x_3 + x_4 + \dots + x_q$
Cir		x_4: E, AC ← cir E, AC	
Cil		x_5: E, AC ← cil E, AC	
ADD	D_1T_5	x_6: AC ← AC + DR	
AND	D_0T_5	x_7: AC ← AC ∧ DR	
LDA	D_2T_5	x_8: AC ← DR	
INP	$D_7IT_3IR_{11}$	x_9: AC(0-7) ← INPR	

cil، cir، CMA از طریق ALU

مدار یک بیت ALU



مدار CPU از دید کاربر

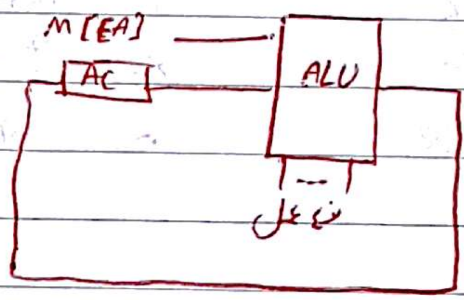
دستور
- تعداد AF ها در دستورات پیوانی بیشتر
- سرهای آدرس دهی
- op کدها
- نکته های دستور

تعداد AF=1 سازمان AC
قوعاً" آدرس حافظه

general Register مرجع

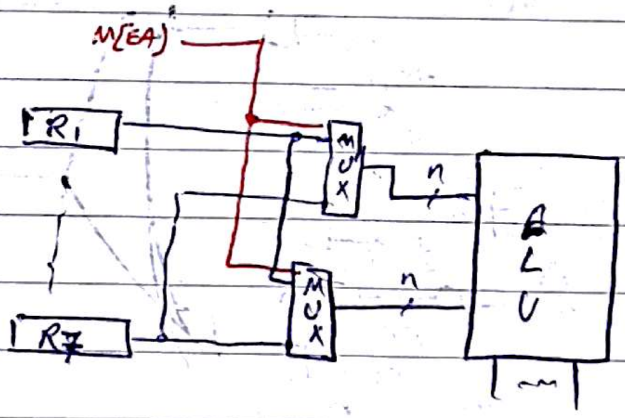
تعداد AF= 2/3 {
سازمان GR ← عملیات با رجیستر سریعتره ← میایم رجیستر ها رو بیشتر می کنیم. مراجعه به حافظه کم می شه.
از load و store دائماً اجتناب می کند.
که حداکثر یک آدرس داشته
سازمان RISC ← همون GR بدون آدرس حافظه
}

برای ما



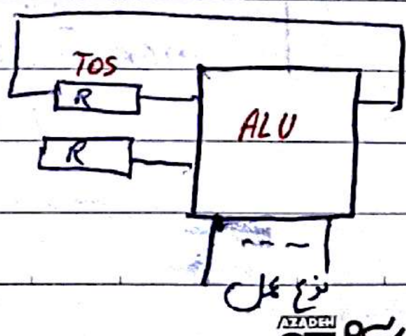
سخت افزارمون ارزان شد
هی رجیستر اضافه نشد.

در GR



در GR می شه از سه تا آدرس حافظه استفاده کرد مثلن مثلاً 100 رو با 200 جمع کن بذار تو 300 رو عموماً این کار نمی شه به خاطر کم شدن سرعت و ارزان ← حداکثر یک آدرس حافظه

تعداد آدرس صفر (AF=0) ← فقط نوع دستور گفته می شه.
سازمان پشته



پردازش ها

$X = (A+B) * (C-D)$

ترفیق که ضرب و تفریق هم داریم

سازمان AC	سازمان GR	سازمان RISC
LDA A	LD R1, A	LD R1, A
ADD B	ADD R1, B, R2	LD R2, B
STA T	LD R3, C	ADD R1, R2, R3
LDA C	SUB R3, D, R4	LD R4, C
SUB D	MUL R2, R4, X	LD R5, D
MUL T		SUB R4, R5, R6
STA X		MUL R6, R3, R7
		ST R7 X

LD R1, A → A را در R1 LDA

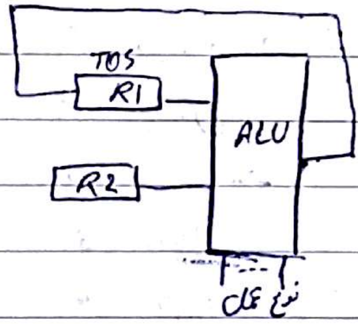
ADD R1, B, R2 → $R2 \leftarrow R1 + B$

ADD R1, R2, R3 → آدرس حافظه ندارن دیگه پردازشیا

سازمان AC
سازمان GR
سازمان RISC
سازمان پشته

تعداد AF در دستورات
مدهای آدرس دهی
تنوع op
فلگ ها و تفسیر آنها

CPU



سازمان پشته

$X = (A+B) * (C-D)$ infix notation

$A+B$ " "

$AB+$ Post fix "

$+AB$ Pre fix "

$AB+\ CD-\ *$

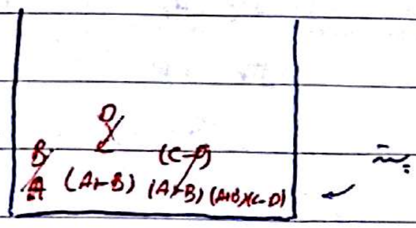
$X = A+B*C-D$ ⟶ $ABC*+D-$

توی سازمان پشته ورودی ها رو می دادیم بالای پشته به علاوه می رسید حساب می کنه حاصل رو می ذاره بالای پشته.

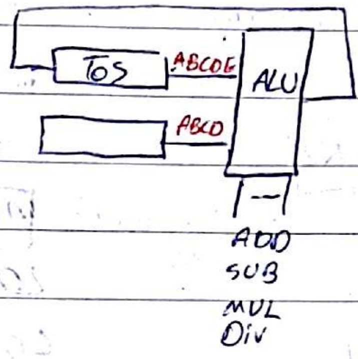
$(A+B)*(C-D)$ ⟶ $AB+CD-*$

$ABCDE+-*/$

$A/(B*(C-(D+E)))$



گذاشتن در پشته: Push ، برداشتن پشته: pop



$D+E \rightarrow C-(D+E) \rightarrow B*(C-(D+E))$

C ، B ، A

$\rightarrow A/(B*(C-(D+E)))$

$X=(A+B)*(C-D) = AB+CD-*$

دستورهای بدون آدرس.

Push A
Push B
ADD
Push C
Push D
SUB
MUL
Pop X

انواع مدهای آدرس دهی (AF نحوه تفسیر) راندمان و تسهیلات

(عملوند؟ EA چیست؟ AF حاوی چیست؟)

۱- مد ضمنی ← AF نداریم. محل عملوند در مدار مشخص و مستتر است. ← مثل توی AC ه

2- بلافصل ، ← خود AF عملوند قرار دارد. ... $ADD \mid 100 \Rightarrow AC \leftarrow AC+100$ (دیگر احتیاج به حافظه نداریم دستور دارای عملوند هست)

مد	AF	EA	عملوند
3- حافظه‌ای مستقیم	آدرس حافظه	EA خود AF	M[AF]
4- حافظه‌ای غیر مستقیم	"	M[AF]	M[M[AF]]
5- رجیستری مستقیم	آدرس رجیستر	-	(R) محتوای رجیستر آدرس داده شده
6- رجیستری غیر مستقیم	آدرس رجیستر	(R)	M[(R)]
7- Auto inc	آدرس رجیستر	(R)	M[(R)] , R←R+1
8- Auto dec	آدرس رجیستر	(R)	M[(R)] , R←R-1
9- نسبی	عدد صحیح با علامت	آدرس PC+AF	M[PC+AF] → عملوند نسبت به 2 تا بعد 2 یا 3 تا OPC
10- اندیسی	آدرس حافظه	IX + AF (مرجع بردار)	M[IX+AF]
11- بیسی	آدرس حافظه	BR+AF (AI) بیس رجیستر	M[BR+AF]

(مدهای 5 تا 8: راندمان ↑)

op

ADD	R1	R2	R3	
4	4	4	4	

00 مستقیم
01 غیرمستقیم
10 Auto inc
11 Auto dec

اگه یکی از اینا رو بخوایم استفاده کنیم اگه بیدونی که بخوایم از هر ۴ مد رجیستری استفاده کنیم دو بیت می‌خوایم (قرمز)

6 بیت آخر:

0011 00 → R1 ← R2 + R3 ADD R1, R2, R3

0011 01 → R1 ← R2 + M[R3] ADD R1, R2, [R3]

0011 10 → R1 ← R2 + M[R3], R3 ← R3 + 1 ADD R1, R2, [R3]+

0011 11 → R1 ← R2 + M[R3], R3 ← R3 − 1 ADD R1, R2, [R3]−

	آدرس‌ها = مراجعه به حافظه			مراجعه به حافظه
ORG 100 H	2	LDA AA	LD R1, # 100 H	1
A, :	2	STA Pt	LD R2, # −20 D	1
AA, Hex 100	2	LDA LA	CL R3	1
LA, DEC 20	1	CMA	lop, ADD R3, [R1]+, R3	2 (f_3 M[R1])
	1	INC	~~ISZ R1~~	1
Pt, Hex 100	2	STA CNT	ISZ R2	1
	1	CLA	BUN lop	1
CNT, DEC −20	3	lop, ADD Pt I		
	3	ISZ Pt		
	3	ISZ CNT		
	1	BUN lop		
	2	STA SUM		

مراجعه به حافظه خیلی زیاده.

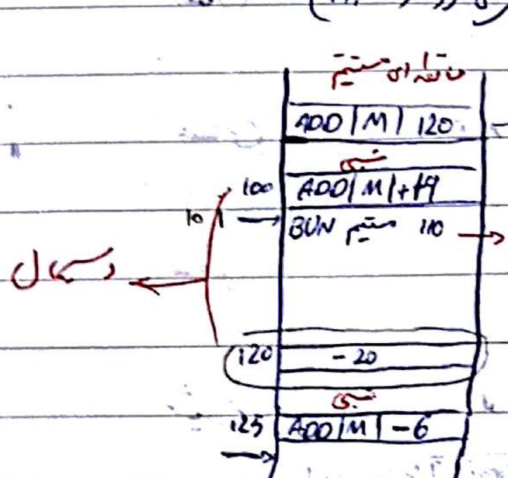
هر حلقه 10

هر حلقه 5

طول دستورات کم می‌شه چون آدرس رجیسترها کوتاهه.

اگه Auto inc بشه قرمز

مد رجیستری نسبی به پی سی ← جابجایی پذیری + طول دستور کوتاه‌تر می‌شود (AF)



AC ← AC + M(120)

AC ← AC + M[PC + 19]

→ BUN (نسبی) +8 → کوتاهتری شه

عملوند

هرجی مد نسبی اینه که جابجایی پذیره

مثلن این الان از 100 می‌خوایم بشه از 200 (ORG 200) باید کدرو عوض کنیم چون آدرسا عوض می‌شه

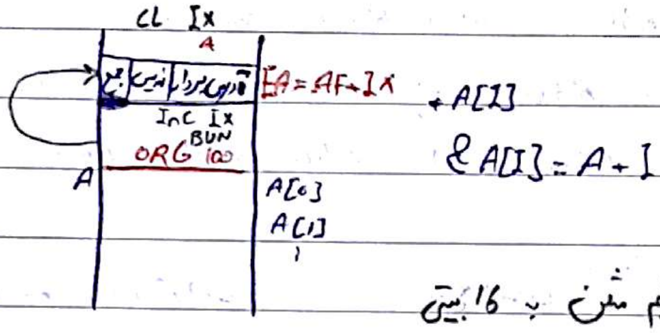
ولی تو نسبی لازم نیست عوض بشن هیچ‌وقت از 200 می‌خوایم.

← جابجایی پذیره تو ی رجیستری‌ها هم عوض نمی‌شه.

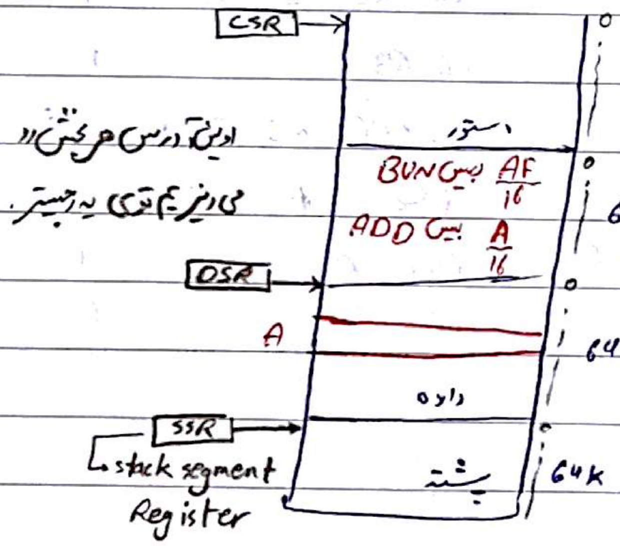
که مگه اینکه از آدرس استفاده کرده باشه.

عملوند ؟   EA ؟   AF ؟   تعداد AF

CPU:
- مدهای آدرس دهی — مد نسبی
- تنوع op — مد اندیس — آدرس حافظه — AF+IX — M[AF+IX] — برای جدول اندیس
- تغییر تکه ها — مد بیس ستینم — آدرس حافظه — AF+BR — M[AF+BR] — جابجایی پذیری، دکامش طول AF



اگه آدرس 32 بیت بود تقسیم می‌کنیم مثل به 16 بیتی



آدرس 16 بیتی رو با آدرس بیس جمع می‌کنیم. 64k ← 16 بیت

AF	EA	عملوند
AF	AF	M[AF]
AF	BR+AF	M[BR+AF]
AF	M[AF]	M[M[AF]]
AF	M[BR+AF]	M[M[BR+AF]]

M[BR+M[AF+BR]]

تنوع op-code: MOV R1,R2 : R2←R1 (انتقال بین رجیسترها) ، MOV R1,[R2] (به آدرس محتوای R2) — نگه داشتن آدرس stack — نگه داشتن stack ها

انتقال: LD, ST, MOV, xch, push, PoP, Push all, PoP all ← جابجایی دو رجیستر

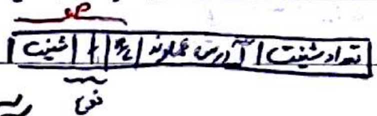
حسابی: ADD, Inc, CL, SUB, MUL, DIV, ADC, SBB, DEC, NEG — برای هر کدام چند نوع داریم:

INT, BCD, float → ADDI, ADDBCD, ADDFL, ADDDP

double precision ← با دقت 64

منطقی: CMA, NOT, AND, OR, XOR, Set, clear

شیفت: shr, shl, cir, cil, Ashr, Ashl, عملوند، تعداد شیفت



یه نوع دیگه circular | circular through

Subject:
Date

کنترل برنامه

I/O

← بخاطر معادلات مثل I/O یا mem map یا Isole ← INPOUT داره. LD داره

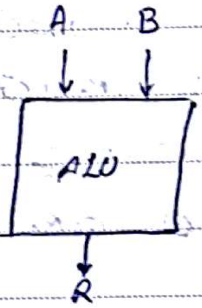
E, S = AC(15), Z = NOR AC(0-15) بقیه تعریف شده رجیسترهاست از خودش تک هارو داشتیم که بیشتر در رجیسترها بود.
برای جایی که در رجیسترها از داینک تک ها داریم یه status رجیستر داریم.

if (CD) PC ← EA

CD			
1	BUN adr	P	B podd adr ← parity
C	BC adr	P'	BPEV adr
C'	BNC adr	ov	Bov ← over flow
S'	BPos adr	ov'	BNOV adr
S	BNEG adr		
Z	BZ adr		
Z'	BNZ adr		

Branch ← اگر کری هست.

| S | C | Z | ov | P | AXC |



if(CD) { M[SP] ← PC; SP ← SP-1; PC ← EA }

if(CD) { SP ← SP+1; PC ← M[SP] }

CD			
1	Call adr	P	Cpodd adr
C	CC adr	P'	CPEV adr
C'	CNC adr	ov	COV adr
S'	CPos adr	ov'	CNov adr
S	CNEG adr		
Z	CZ adr		
Z'	CNZ adr		

CD			
1	RET	P	Rpodd
C	RC	P'	RPEV
C'	RNC	ov	Rov
S'	RPos	ov'	RNov
S	RNEG		
Z	RZ		
Z'	RZ'		

RET آدرس نداره

if(CD) PC ← EA

CD	CMP A,B / SUB A,B,C	CMP A,B / SUB A,B,C
>	BGT	BHT
≥	BGE	BHE
<	BLT	:
≤	BLE	:
==	BEQ	:
!=	BNEQ	:

← با علامت — ترکیبی از تک ها — ← بدون علامت

$A>B \Rightarrow (A-B)>0$

مقایسه‌ها براساس تفریق (مدار مقایسه کننده جدا نداریم)

برای این مقایسه ها سه تا آدرس می‌خوان:

چیزهایی مقایسه کننده نگه می‌داره: اما لزوماً

SUB A,B,C ← حاصل تفریق تو C و فقط به تک ها مقدار می‌ده

CMP A,B ← تفریق می‌کنه تک ها رو تغییر می‌ده ولی دیگه جایی نتیجه رو نگه نمی‌داره

اگه یه پروسسور CMP داشت ازش استفاده می‌کنیم جای SUB

PAPCO

Subject:
Date

AND A, $\overset{mask}{\tilde{B}}$, C / C = A ∧ B — می‌خوایم بیت‌های A رو چک کنیم.

TEST A, Mask — چون کار AND رو می‌کنه ولی فلگ‌ها رو تغییر می‌ده حاصل رو ذخیره نمی‌کنه.

فلگ‌ها: مقدارگیری — نوعاً برای پردازش دستورات پردازش مقدار می‌گیرند. و در جدول دستورات مشخص شده. کنترل برنامه از فلگ‌ها استفاده می‌کنن ولی عوضشون نمی‌کنن.

تفسیر

$Z = (R_0 + R_1 + \dots + R_{n-1})'$

$S = R_{n-1}$ بیت علامت

$P = (R_0 \oplus R_1 \oplus \dots \oplus R_{n-1})$

$C/B = (ADD) \cdot C_n + (SUB) \cdot b_n + (Cir) \cdot A_0 + (Cil) A_{n-1} \dots$

(ADC: در دستور جمع — کری خروجی؛ SBB: در تفریق — قرض خروجی)

$OV = (ADD) \cdot (C_n \oplus C_{n-1}) + (SUB)(b_n \oplus b_{n-1}) + (Ashl)(A_{n-1} \oplus A_{n-2}) + (DIV) \dots$

با علامت

که بر صفر تقسیم شده یا n بیتی بر n بیتی تقسیم شده که بیت اولش بزرگتر شد.

تفسیر فلگ‌ها

int A, B — جمع و تفریق با علامت → OV → اعداد با علامت
که برای بی‌علامت معنی نداره.

unsigned int — برای جمع و تفریق بدون علامت → b_n و C_n → اعداد بدون علامت
که برای با علامت بی‌معنی هستن.

A — عدد با علامت مثبت: [01 - - ———] C — عدد با بیت با ارزش صفر

D — عدد با علامت منفی: [11 ———] D — عدد با بیت با ارزش یک

تفسیر عوض می‌شه — بیت علامت نیست

در جمع با علامت کری خروجی یعنی سرریز نمی‌ده اما $C_n \oplus C_{n-1}$: OV اما در جمع بدون علامت کری خروجی یعنی سرریز می‌ده.
$C = C_n$

در با علامت کری نداریم OV داریم.

```
B   1 1 1 1
    0 1 1 0 1
A   1 1 0 1 1 +
   ----------
    0 1 0 0 0
```

status

C	S	Z	OV	P
1	0	0	0	1

```
  0 1 1 0 1
  0 1 1 0 1
+ 0 0 1 0 1
 ----------
  1 0 0 1 0
```

C = b'

```
  1 0 0 1 0
  0 1 1 0 1
  1 1 0 1 1 -
 ----------
  1 0 0 1 0
```

status

C	S	Z	OV	P
1	1	0	1	0

$C_n = b_n'$ اگر تفریق با جمع و مکمل باشد

بارو در بدون علامت یعنی عدد اول از دوم کوچکتر بوده.

P4PCO

5<7 یعنی ① 5 − 7 = 8

+5 − +7 = −2