جلسه پنجم

طراحی کامپیوتر پایه (ادامه)

تاریخ میان ترم: سه شنبه ۱۶ اردیبهشت

مهم: هر دانشجو، فقط در ساعت ثبت نامی خود سر امتحان حاظر شود

میزان نمره: ۴ نمره از ۲۰ نمره

- در صورت تغییر از طریق وب سایت به اطلاع خواهد رسید

>> salar.seyedebrahimi.ir <<

کنترل و زمان بندی

- زمان بندی تمام ثبات ها در کامپیوتر توسط یک تولید کننده پالس ساعت کنترل می شود
- پالس های ساعت به تمام ثبات ها و فلیپ فلاپ ها اعمال می شوند
- پالس های ساعت مقدار یک ثبات را تغییر نمی دهند بلکه زمان انجام یک ریز عمل روی آن ثبات را تعیین می کنند. همچنین ثبات باید توسط یکی از سیگنال های کنترلی فعال شده باشد.
- سیگنال های کنترلی در واحد کنترل تولید می شوند و شامل ورودی های کنترل مالتی پلکسر در باس، ورودی های کنترل ثبات ها در پردازنده و ریز عملیات مربوط به اکومولاتور هستند
- دو نوع ساختار عمده کنترلی وجود دارد.

کنترل و زمان بندی

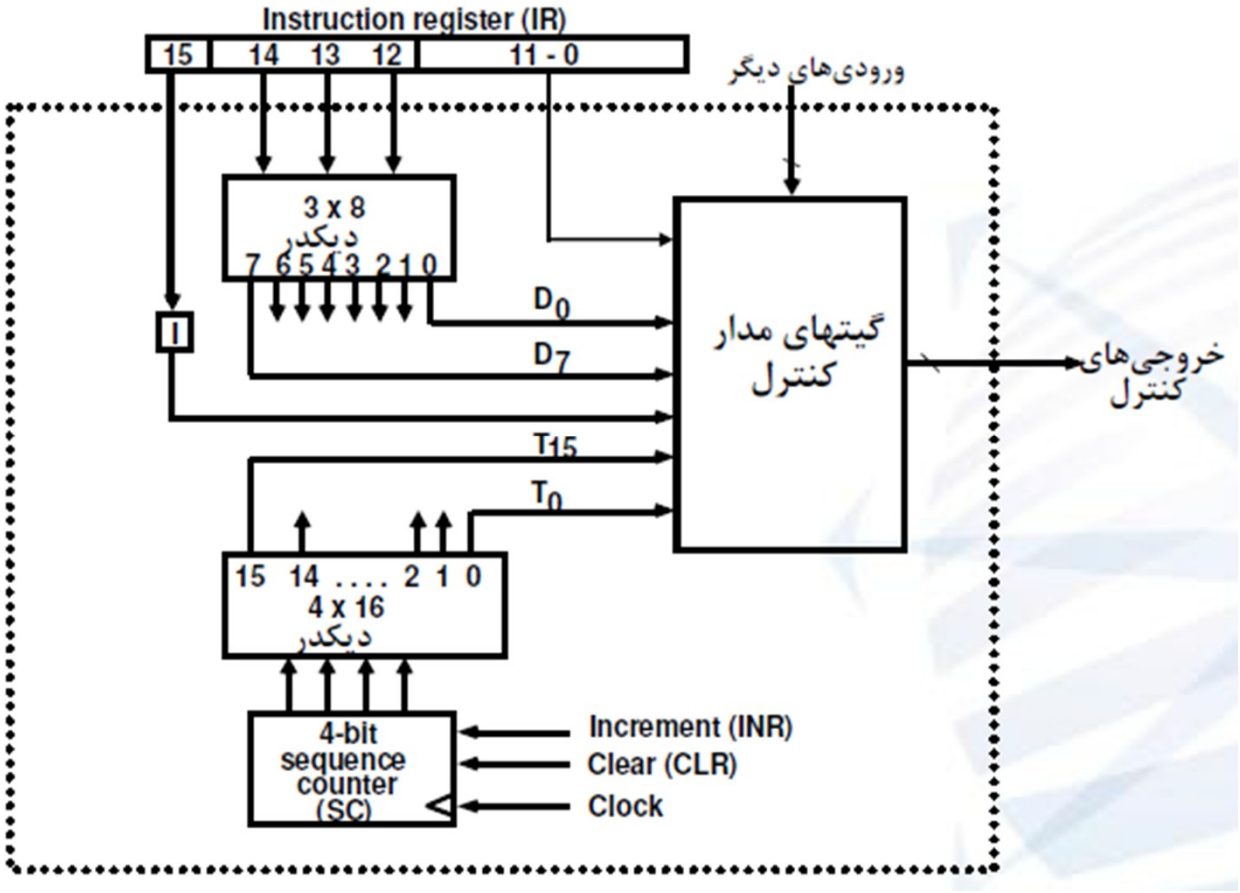
- دو نوع ساختار کنترلی عمده وجود دارد
- کنترل سیم بندی شده (Hard-wired Control Unit)
  - منطق کنترل توسط گیتها، فلیپ فلاپ ها، رمز گشاها و سایر مدارات دجیتال پیاده سازی می شود. مزیت این کنترل سرعت بالای آن است

- کنترل ریز برنامه ریزی شده (Micro-programmed Control Unit)
  - اطلاعات کنترلی، در یک حافظه کنترلی ذخیره می شوند. این حافظه طوری برنامه ریزی می شود که دنباله ای از ریزعملیات را تولید کند. مزیت این نوع کنترل، انعطاف پذیری بالای آن است. هر تغییر احتمالی یا به روز کردن اطلاعات حافظه کنترلی قابل اعمال است.

در این بخش نوع اول بررسی می شود

کنترل و زمان بندی

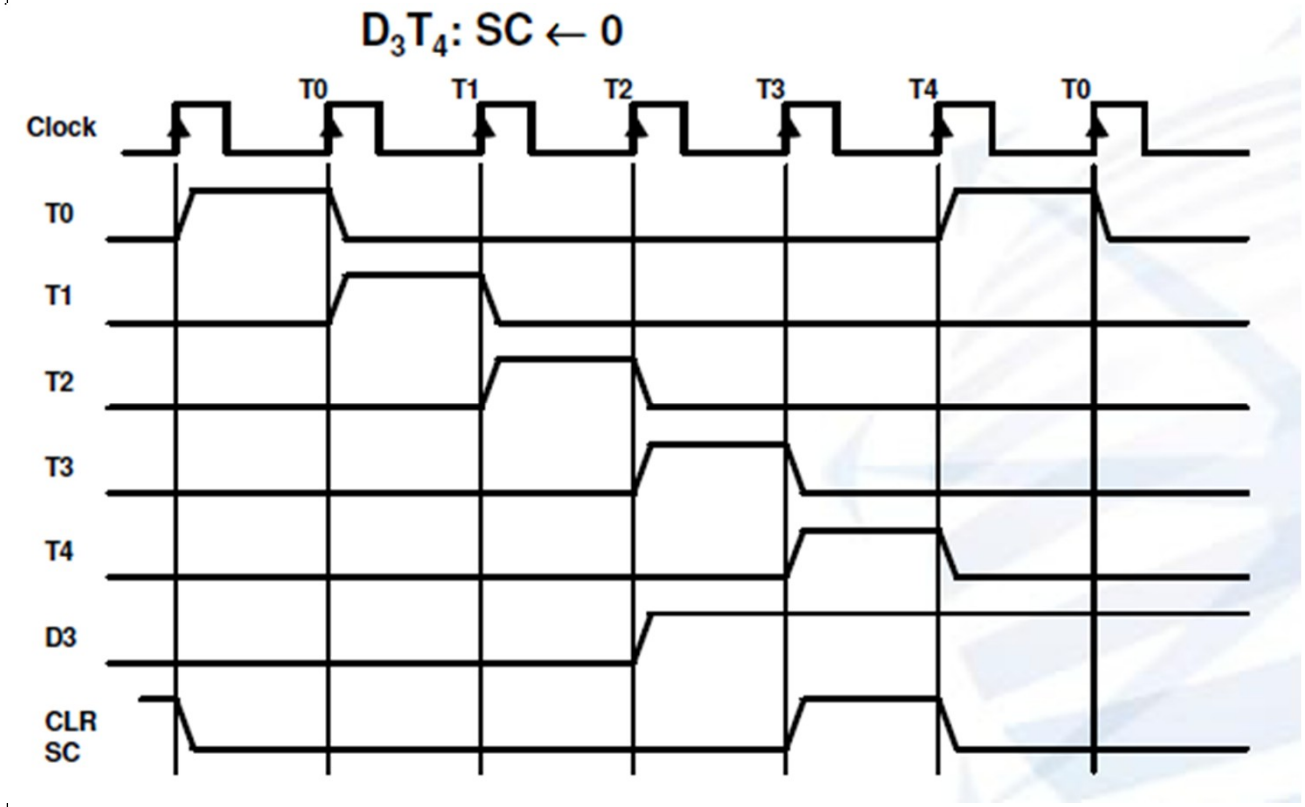
- بلوک دیاگرام واحد کنترل سیم بندی شده



- دستوری که از حافظه خوانده می شود، در ثبات IR قرار می گیرد، کد عملیاتی یعنی بیت های ۱۲ تا ۱۴ توسط یک رمزگشای 3x8 ترجمه شده و D0 تا D7 را تولید می کنند. بیت ۱۵ ثبات به فلیپ فلاپ I منتقل می شود.
- بیت های ۰ تا ۱۱ نیز به واحد کنترل منتقل می شوند. شمارنده متوالی از ۰ تا ۱۵ شمرده و رمز گشای 4x16 سیگنال های زمان بندی T0تا T15 را تولید می کند. اغلب اوقات شمارنده یکی یکی می شمارد ولی گاه نیاز است تا با ریست کردن آن، شمارش را از ابتدا شروع کند

کنترل و زمان بندی

- سیگنال های زمان بندی



- در لبه مثبت هر کلاک، شمارنده یک واحد می شمارد. اگر شمارنده ریست نشود، تا T15 شمرده و مجددا به T0 باز می کردد. ولی با فعال شدن D3 و T4، CLR فعال شده و شمارنده از صفر شروع به شمارش می کند که این امر موجب تولید T0 می شود.

دستورات کامپیوتر پایه

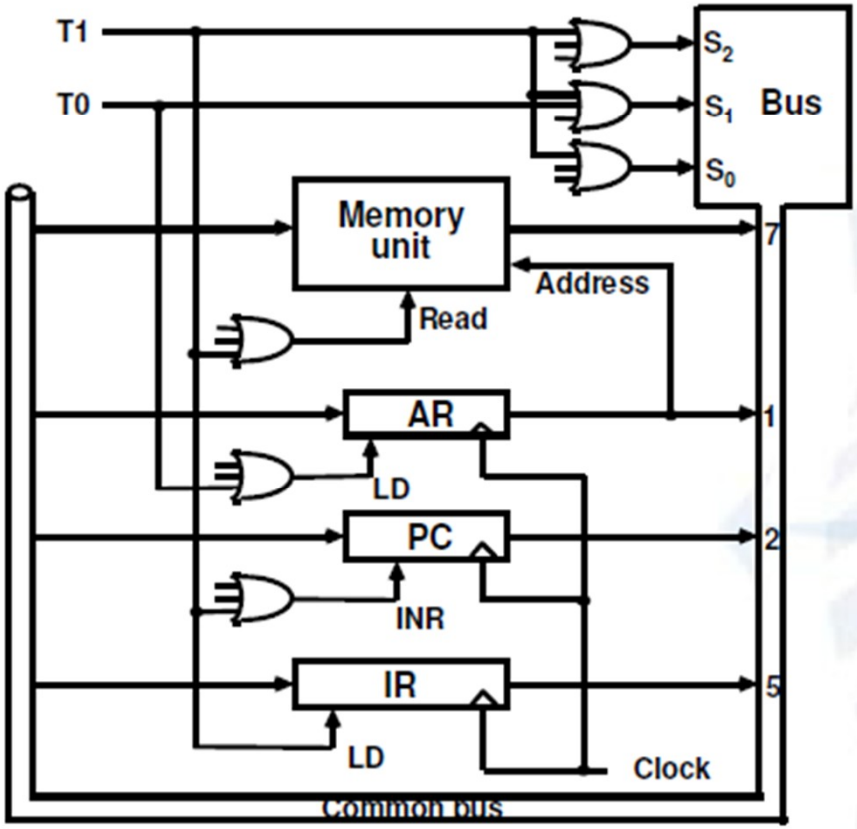
- سیکل دستور العمل طبق الگوریتم ون نیومن به ترتیب زیر است:
  - ۱. واکشی دستور از حافظه (Instruction Fetch)
  - ۲. رمز گشایی دستور (Instruction Decode)
  - ۳. پیدا کردن آدرس موثر دستور اینکه I=1 (Finding Effective Address)
  - ۴. اجرای دستور

- ریز عملیات واکشی و رمز گشایی عبارتند از:
  - واکشی یا fetch — **T0: AR ← PC**
  - واکشی یا fetch — **T1: IR ← M [AR], PC ← PC + 1**
  - ترجمه و دیکود کردن — **T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)**

واکشی وکدگشایی

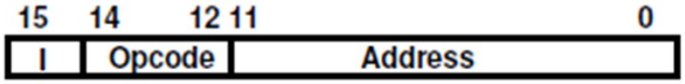
- **Fetch and Decode**

T0: AR ← PC ($S_0S_1S_2$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)



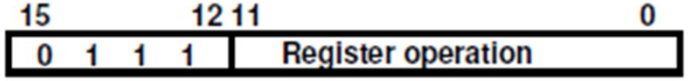
دستورات کامپیوتر پایه

- تعداد بیت های اختصاص یافته به Opcode در فرمت دستورالعمل کامپیوتر پایه ، **3** بیت می باشد.
- اگر سه بیت opcode مخالف ۱۱۱ باشند، دستور حافظه ای است.



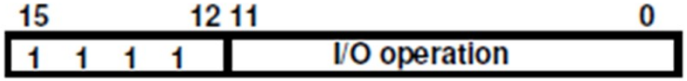
دستورالعمل های مراجعه به حافظه
(OP-code = 000 ~ 110)

- اگر سه بیت opcode مساوی ۱۱۱ باشند و I=0 دستور ثباتی است.



دستورالعمل های مراجعه به ثبات
(OP-code = 111, I = 0)

- اگر سه بیت opcode مساوی ۱۱۱ باشند و I=1 دستور ورودی خروجی است.



دستورالعمل های ورودی خروجی
(OP-code =111, I = 1)

دستورات کامپیوتر پایه

- با توجه به اینکه در کلاک های T0و T1 و T2 ، واکشی و رمز گشایی صورت می گیرد، در T3 یکی از عملیات زیر صورت می گیرد

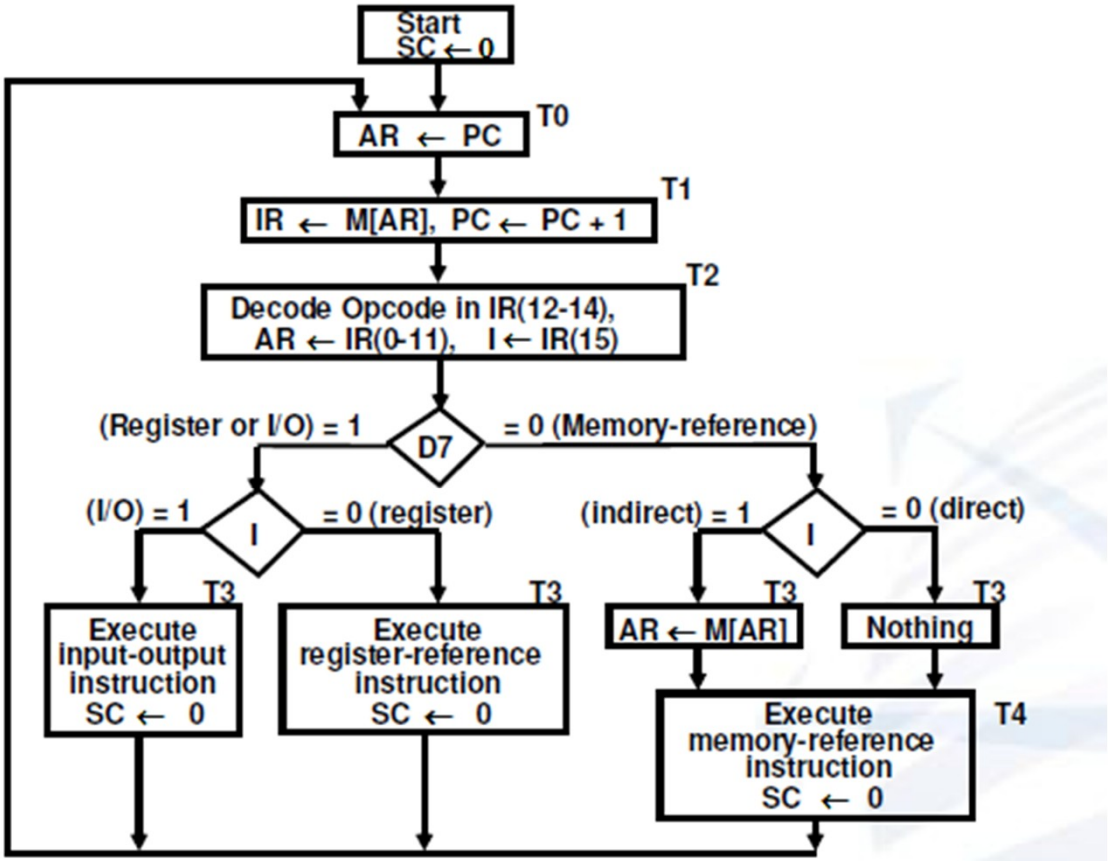
۱. اگر دستور حافظه ای مستقیم باشد (I=0)، هیچ کاری صورت نمی گیرد

۲. اگر دستور حافظه ای غیر مستقیم باشد (I=1)، آدرس موثر به AR منتقل می شود

۳. اگر دستور ثباتی باشد، دستور اجرا می شود

۴. اگر دستور ورودی/خروجی باشد، اجرا می شود.

تعیین نوع دستورالعمل



D'_7IT_3: $AR \leftarrow M[AR]$

$D'_7I'T_3$: Nothing

$D_7I'T_3$: Execute a register-reference instr.

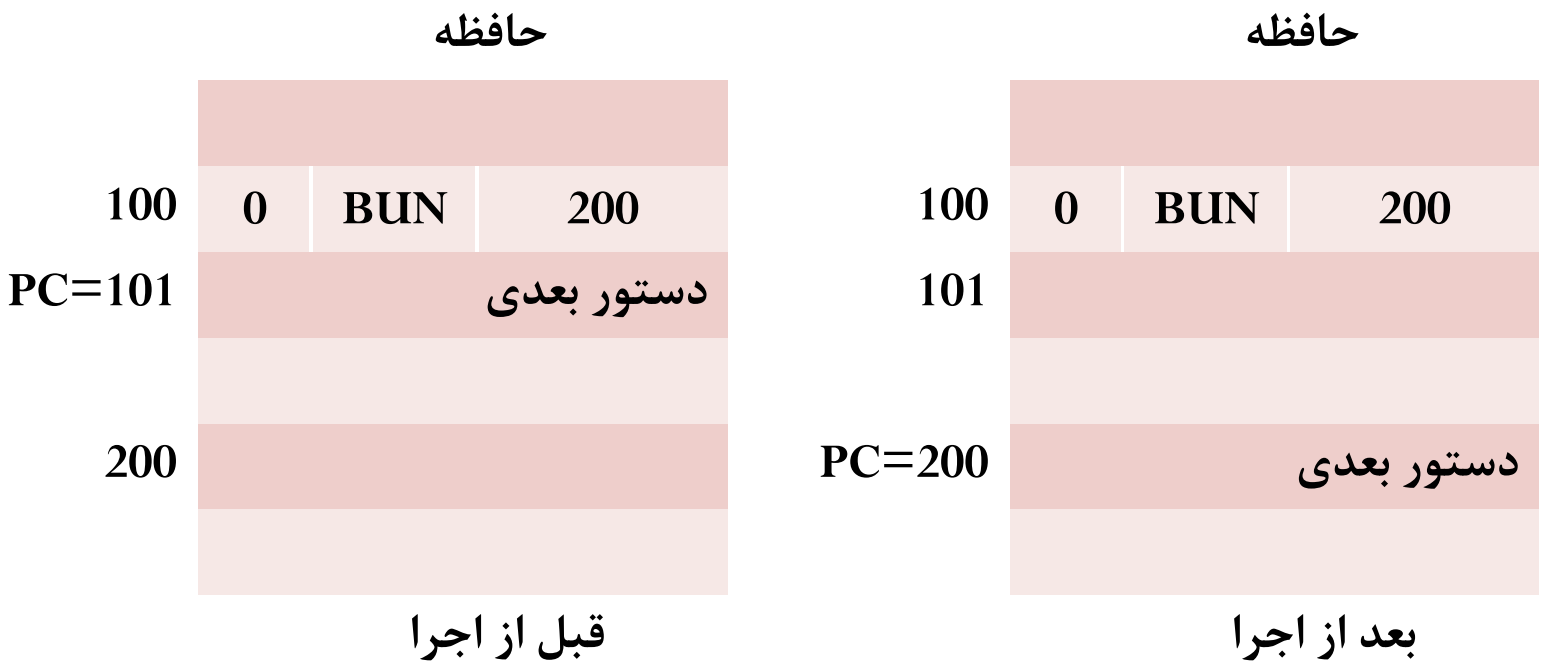
D_7IT_3: Execute an input-output instr.

دستورات حافظه ای

دستور	کد RTL
AND	D_0T_4:DR ← M[AR] D_0T_5:AC← AC∧DR , SC← 0
ADD	D_1T_4:DR ← M[AR] D_1T_5:AC← AC+DR, E← Cout, SC← 0
LDA	D_2T_4:DR ← M[AR] D_2T_5:AC← DR , SC← 0
STA	D_3T_4:M[AR]← AC, SC← 0
BUN	D_4T_4:PC← AR, SC← 0
BSA	D_5T_4:M[AR]← PC, AR← AR+1 D_5T_5:PC← AR, SC← 0
ISZ	D_6T_4:DR ← M[AR] D_6T_5:DR← DR+1 D_6T_6:M[AR]← DR, IF(DR==0) THEN (PC← PC+1), SC← 0

الف- مثال دستورات حافظه ای

- دستور BUN یک دستور پرش غیر شرطی است و آدرس موثر را در PC قرار می دهد



BUN مستقیم

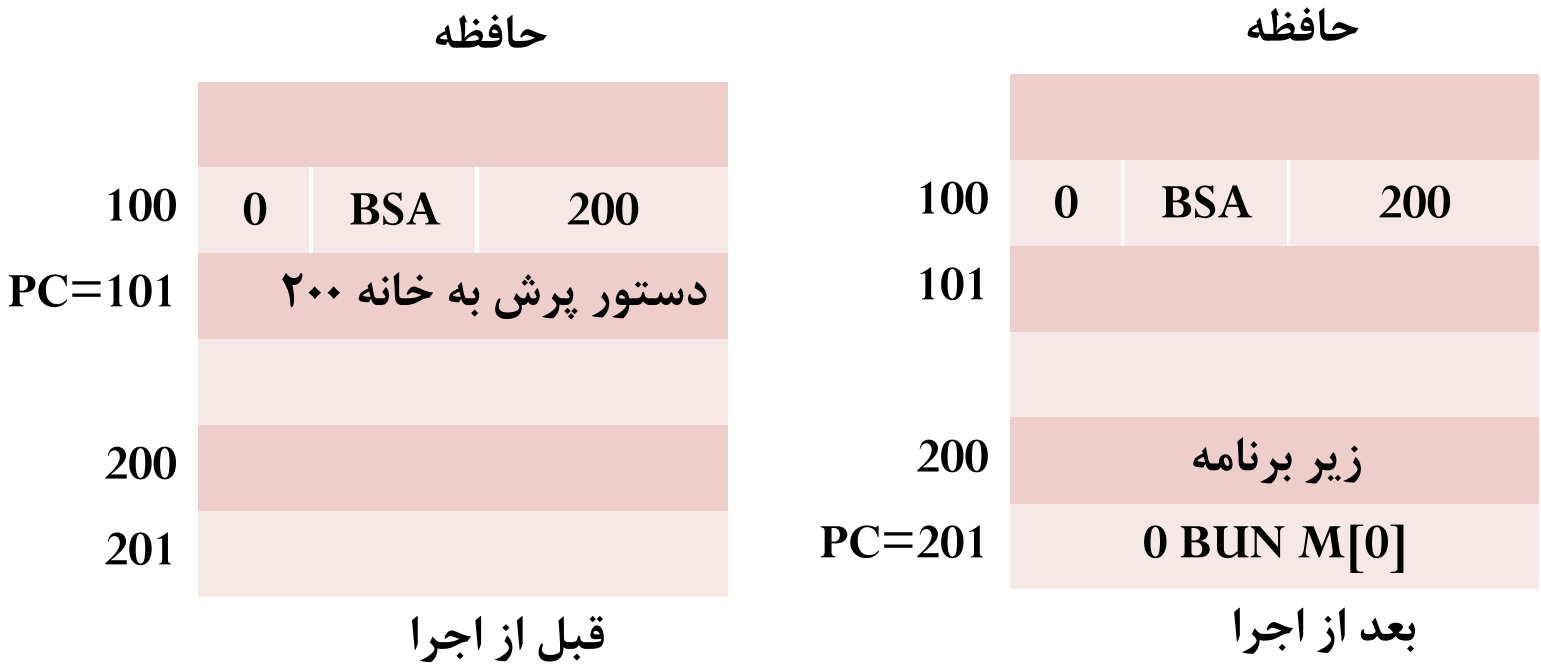
الف- مثال دستورات حافظه ای

- دستور BUN یک دستور پرش غیر شرطی است و آدرس موثر را در PC قرار می دهد

BUN غیر مستقیم

الف- مثال دستورات حافظه ای:

- دستور BSA ابتدا PC را در M[AR=0] ذخیره می کند و سپس به خانه بعدی آن پرش می کند، یعنی آدرس برگشت در ابتدای برنامه ذخیره می شود. پس در انتهای برنامه، برنامه نویس باید با کمک دستور BUN به نقطه اولیه بازگشت کند



BSA مستقیم

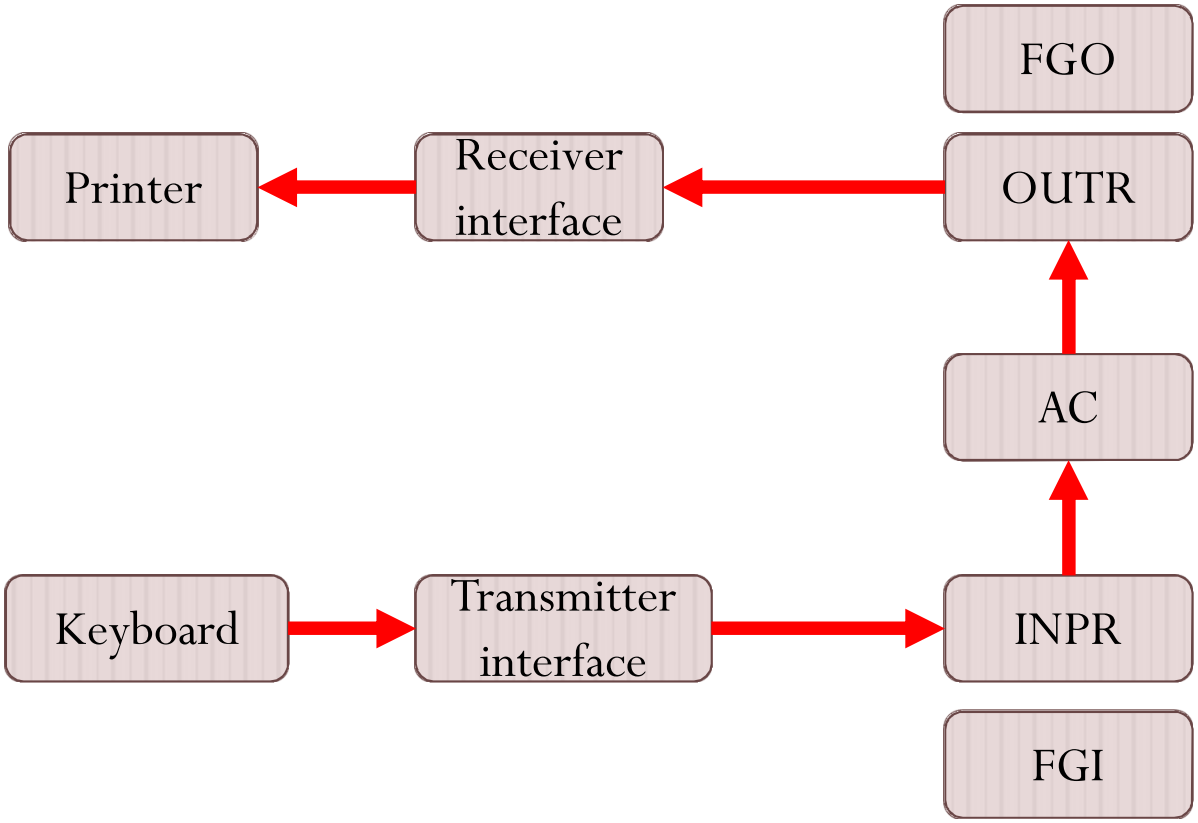
لیست کلی ریز عملیات

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D0, ..., D7 \leftarrow$ Decode $IR(12 \sim 14)$, $AR \leftarrow IR(0 \sim 11), I \leftarrow IR(15)$
Indirect	$D_7'IT_3$:	$AR \leftarrow M[AR]$
Interrupt		
	$T_0'T_1'T_2'(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Memory-Reference		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if(DR=0) then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

لیست کلی ریز عملیات

Register-Reference		
	$D_7I'T_3 = r$	(Common to all register-reference instr)
	$IR(i) = B_i$	(i = 0,1,2, ..., 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow AC'$
CME	rB_8:	$E \leftarrow E'$
CIR	rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	$If(AC(15) = 0)\ then\ (PC \leftarrow PC + 1)$
SNA	rB_3:	$If(AC(15) = 1)\ then\ (PC \leftarrow PC + 1)$
SZA	rB_2:	$If(AC = 0)\ then\ (PC \leftarrow PC + 1)$
SZE	rB_1:	$If(E = 0)\ then\ (PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Input-Output	$D_7IT_3 = p$	(Common to all input-output instructions)
	$IR(i) = B_i$	(i = 6,7,8,9,10,11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7),\ FGO \leftarrow 0$
SKI	pB_9:	$If(FGI = 1)\ then\ (PC \leftarrow PC + 1)$
SKO	pB_8:	$If(FGO = 1)\ then\ (PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

ج- دستورات I/O:



- پرچم یک بیتی ورودی FGI (flag input) یک فلیپ فلاپ کنترلی است.
- وقتی اطلاعات جدیدی در ورودی حاضر است، FGI=1 می شود و زمانیکه اطلاعات توسط کامپیوتر پذیرفته می شود FGI=0 می شود.
- پرچم یک بیتی خروجی FGO (flag Output) یک فلیپ فلاپ کنترلی است.
- کامپیوتر FGO را چک می کند اگر یک بود اطلاعات از AC به طور موازی به OUTR منتقل شده و FGO=0 می شود. وسیله خروجی اطلاعات را می پذیرد و کاراکتر معادل را چاپ می کند و زمانیکه عملیات کامل شد FGO=1 می شود. زمانی که FGO=0 کامپیوتر هیچ کاراکتر جدیدی به OUTR بار نمی کند.

ج- دستورات I/O:

- به منظور انتقال اطلاعات AC ، چک کردن بیت های پرچم و برای کنترل امکان وقفه، از دستورات ورودی- خروجی استفاده می شود.
- این دستورات در طی زمان T_3 اجرا می شوند و اجرای هر دستور ورودی- خروجی، تابع کنترلی D_7IT_3 را نیاز دارد که با حرف P نمایش می دهیم و علاوه بر آن، با توجه به بیت های ۶ تا ۱۱ از هر دستور چون تنها یک بیت از این ۶ بیت فعال است، همان بیت به تابع کنترلی وارد می شود.

	p:	SC ← 0	Clear SC
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0	Input character
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0	Output character
SKI	pB_9:	If(FGI=1) then (PC ← PC + 1)	Skip on input flag
SKO	pB_8:	If(FGO=1) then (PC ← PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6:	IEN ← 0	Interrupt enable off

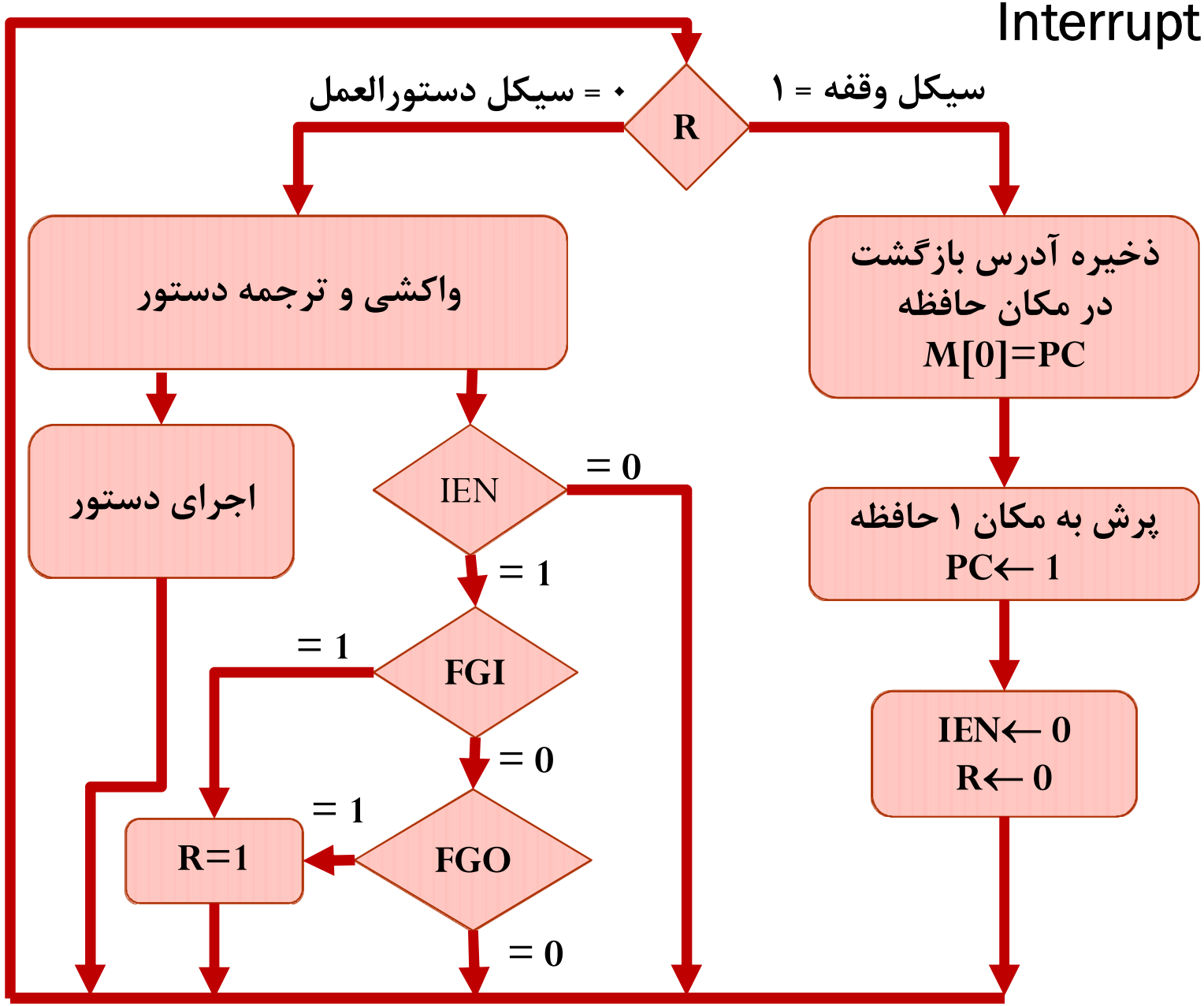
- وقفه یا INTERRUPT :

- کامپیوتر پایه فعلی باید دائما بیت پرچم را چک کند و اگر بیت مورد نظر ست بود، انتقال اطلاعات را آغاز می کند. این روش وقت کامپیوتر را هدر می دهد.

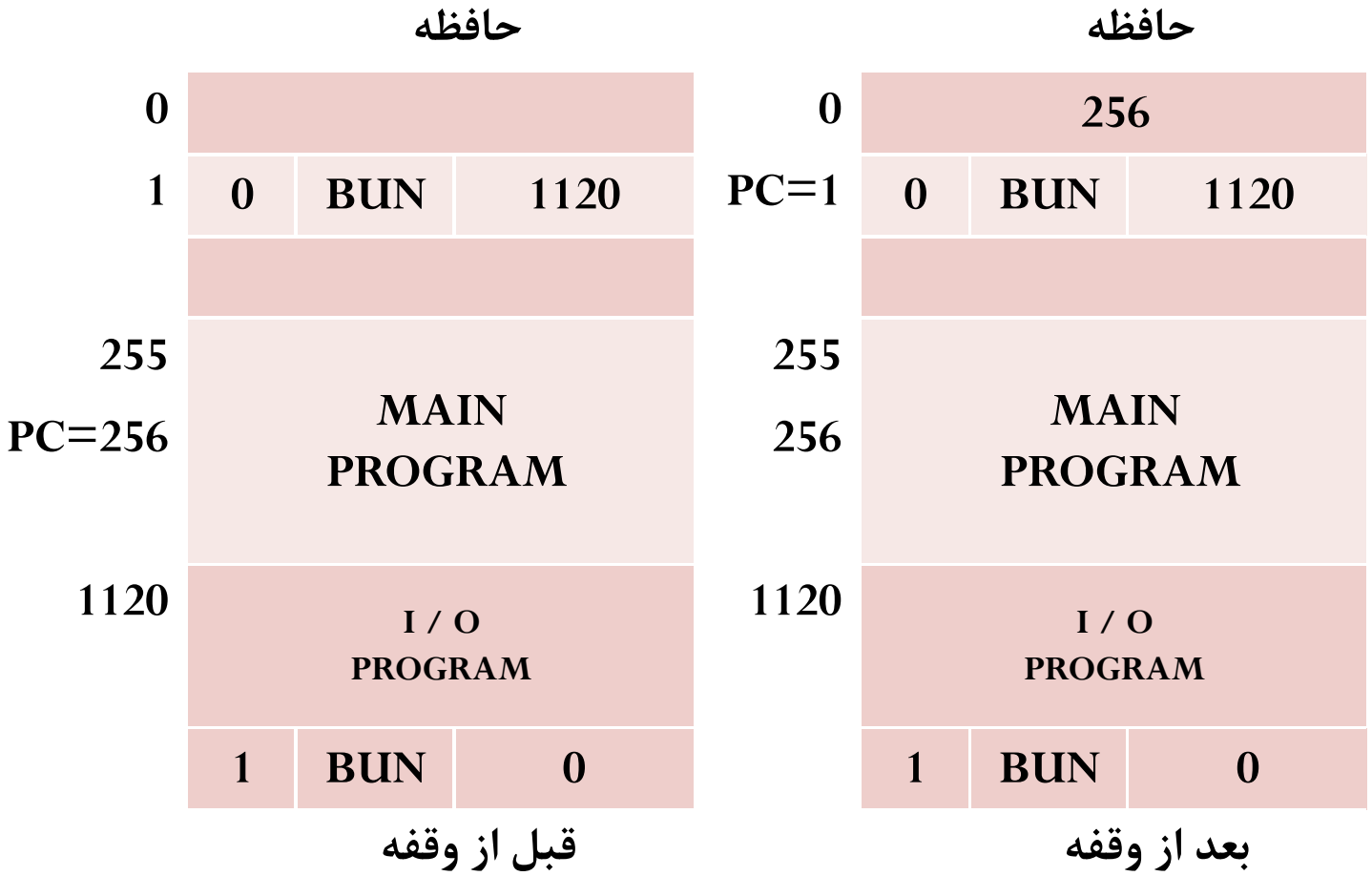
که به آن انتقال تحت کنترل برنامه گویند.

- روش دیگر آنستکه وسایل جانبی زمانیکه برای انتقال اطلاعات آماده هستند، اعلام کنند. در عین حال کامپیوتر می تواند به کار دیگری مشغول باشد و این روش وقفه است.
- زمانیکه کامپیوتر در حال اجرای یک برنامه است، پرچم ها را چک نمی کند. هرگاه پرچمی ست شود، به کامپیوتر توسط وسیله جانبی وقفه داده می شود، کامپیوتر کار فعلی خود را قطع کرده و انتقال ورودی/خروجی (وقفه) را آغاز می کند. پس از پایان سرویس دهی به وقفه، به برنامه قبلی خود باز می گردد.
- سیکل وقفه، پیاده سازی سخت افزاری دستور پرش و ذخیره آدرس بازگشت است. آدرس بازگشت که در PC است، در مکان بخصوصی ذخیره می شود که بتوان پس از بازگشت، دستوری که در آن وقفه رخ داده است را یافت. این مکان می تواند، ثباتی از پردازنده باشد یا پشته یا مکانی از حافظه.
- در کامپیوتر پایه ، PC در آدرس 0 حافظه ذخیره می شود و سپس PC← 1 شده و IEN و R هر دو صفر می شوند.

- وقفه یا Interrupt



- وقفه یا INTERRUPT



- در شکل بالا هنگامیکه کامپیوتر در حال اجرای دستور ۲۵۵ است از خارج از کامپیوتر به آن وقفه اعمال می شود. محتویات PC (۲۵۶) در مکان 0 ذخیره شده و 1 ←PC و 0 ←R می شود.
- در ابتدای سیکل بعدی، دستوری که خوانده می شوداز مکان 1 حافظه است. دستور پرش واقع در آدرس 1 موجب می شود برنامه به روال ورودی / خروجی واقع در آدرس ۱۱۲۰ منتقل شود. و در انتها به مکان قبل از وقوع وقفه باز می گردد.

- سیکل وقفه:

- فلیپ فلاپ R زمانی ۱ می شود که IEN=1 و یکی از پرچم های FGI یا FGO برابر ۱ باشند همچنین سیگنال های زمانی T_0 و T_1 ، T_2 فعال نباشند. یعنی:

- $T_0' T_1' T_2' (IEN)(FGI+FGO) : R \leftarrow 1$

- دنباله ریزعملیات زیر، سیکل وقفه را نشان می دهد:

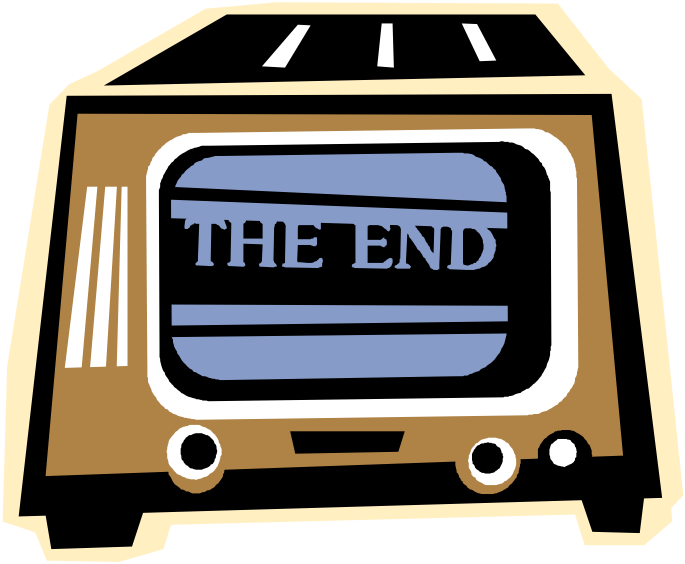
RT_0: AR ← 0, TR ← PC
RT_1: M[AR] ← TR, PC ← 0
RT_2: PC ← PC+1, IEN ← 0, R← 0, SC ← 0

- اکنون به مطالب گذشته مراجعه کرده و به طور مثال مجددا فاز های واکشی و ترجمه را اصلاح می کنیم.

- نکته:

- به جای استفاده فقط از سیگنال های T_0 و T_1 ، T_2 این سیگنال ها را با R' ،AND می کنیم.

R'T0: AR ← PC (S$_0$S$_1$S$_2$=010, T0=1)	واکشی یا fetch
R'T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)	واکشی یا fetch
R'T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)	ترجمه و دیکود کردن



- جلسه بعد: طراحی کنترل سیم بندی شده
- خسته نباشید
- سوال؟