عنوان | صفحه

پایپلاین محاسباتی ۱۰۹
پایپلاین دستورالعمل ۱۱۰
تست‌ها ۱۱۲
پاسخنامه تست‌ها ۱۱۵

■ فصل ششم ـ سازمان ورودی - خروجی ۱۱۶
واسط ورودی - خروجی ۱۱۶
گذرگاه I/O ۱۱۶
فرمان I/O ۱۱۶
گذرگاه I/O و گذرگاه حافظه ۱۱۷
انتقال نادهٔ نامهگام ۱۱۸
کنترل Strobe ۱۱۸
Handshaking ۱۱۹
انتقال سریال نامهگام ۱۲۰
حالت‌های انتقال ۱۲۱
تست‌ها ۱۳۱
پاسخنامه تست‌ها ۱۳۳

■ فصل هفتم - حافظه ۱۳۵
سلسله مراتب حافظه ۱۳۵
حافظه اصلی ۱۳۵
ساختار بیرونی ROM و RAM ۱۳۶
نقشهٔ آدرس حافظه ۱۳۷
اتصال حافظه‌ها به CPU ۱۳۷
پهنای باند حافظه ۱۴۰
حافظه انجمنی (Associative Memory) ۱۴۰
حافظه نهان (Cache memory) ۱۴۲
نگاشت (Mapping) ۱۴۳
نوشتن در حافظه نهان (کش) ۱۴۷
فرمول‌های زمان‌های متوسط دسترسی ۱۴۷
تست‌ها ۱۵۲
پاسخنامه تست‌ها ۱۵۸

فصل اول

معماری کامپیوتر

معماری کامپیوتر[1]

معماری کامپیوتر، طراحی کامپیوتر شامل مجموعه دستورالعمل‌ها، قطعات سخت‌افزاری، و ساختار سیستم است. در بخش اساسی در معماری کامپیوتر مطرح است: معماری مجموعه دستور[2] (ISA) و معماری سخت‌افزار سیستم[3] (HSA). در واقع ISA دید برنامه‌نویس است از کامپیوتر، شامل مجموعه دستورات، مجموعه ثباتها و فرمت دستورات و HSA که به آن دید ریزمعماری[4] نیز می‌گویند شامل اجزاء سخت‌افزاری پردازنده مثل حافظه نهان، چگونگی اجرای دستورات و... است.

دو کامپیوتر با ISA یکسان ممکن است دستورات مشابهی اجرا کنند ولی لزومی ندارد سخت‌افزار داخلی آنها و چگونگی اجرای دستورات مشابه باشد. به عنوان مثال پردازنده‌های خانواده Intel و AMD دستورات مشابهی دارند ولی سخت‌افزار داخلی آنها متفاوت است.

معیارهای اندازه‌گیری کارایی پردازنده

معمولترین معیار اندازه‌گیری کارایی یک CPU[5]، MIPS[6] (تعداد میلیون دستوری که CPU در ثانیه می‌تواند اجرا کند) می‌باشد. برای کامپیوترهایی که محاسبات مهندسی و علمی انجام می‌دهند MIPS مفیدترین معیار سنجش کارایی نیست بلکه از واحد MFlops یا Megaflops استفاده می‌شود؛ یعنی چند میلیون عملیات ممیز شناور[7] در ثانیه می‌تواند انجام دهد. یا از واحد GFlops یا GigaFlops یعنی چند میلیارد عملیات ممیز شناور می‌تواند انجام دهد.

معیارهایی برای سنجش کیفیت معماری کامپیوتر نیز وجود دارد که عبارتند از:

عمومیت[8]: برای چه گستره‌ای از کاربردها، معماری ما مناسب است.

بازدهی[9]: میزان متوسط قطعات یک کامپیوتر که در طی استفاده، مشغول هستند.

و معیارهای دیگری که بحث نمی‌شود.

طبقه‌بندی معماری‌های کامپیوتر

۱ـ ماشین‌های فن‌نیومن[10]:

تقریباً اکثر کامپیوترهایی که با آن آشنا هستید در این دسته قرار می‌گیرند. به هر حال منظور ما از کامپیوتر همان ماشین فن‌نیومن است. ماشین فن‌نیومن دارای خاصیت‌های زیر است:

1- Computer Architecture
2- Instruction - Set Architecture
3- Hardware - System Architecture
4- Microarchitecture
5- Central Processing Unit
6- Million Instructions per second
7- Floating point
8- Generality
9- Efficiency
10- Von Neumann

	r:	$SC \longleftarrow ۰$
CLA	$rB_{۱۱}$:	$AC \longleftarrow ۰$
CLE	$rB_{۱۰}$:	$E \longleftarrow ۰$
CMA	$rB_{۹}$:	$AC \longleftarrow \overline{AC}$
CME	$rB_{۸}$:	$E \longleftarrow \overline{E}$
CIR	$rB_{۷}$:	$AC \longleftarrow shr\ AC, AC(۱۵) \longleftarrow E, E \longleftarrow AC(۰)$
CIL	$rB_{۶}$:	$AC \longleftarrow shl\ AC, AC(۰) \longleftarrow E, E \longleftarrow AC(۱۵)$
INC	$rB_{۵}$:	$AC \longleftarrow AC+۱$
SPA	$rB_{۴}$:	$If(AC(۱۵)=۰)\ then\ (PC \longleftarrow PC+۱)$
SNA	$rB_{۳}$:	$If(AC(۱۵)=۱)\ then\ (PC \longleftarrow PC+۱)$
SZA	$rB_{۲}$:	$If(AC=۰)\ then\ (PC \longleftarrow PC+۱)$
SZE	$rB_{۱}$:	$If(E=۰)\ then\ (PC \longleftarrow PC+۱)$
HLT	$rB_{۰}$:	$S \longleftarrow ۰$

Input-output:

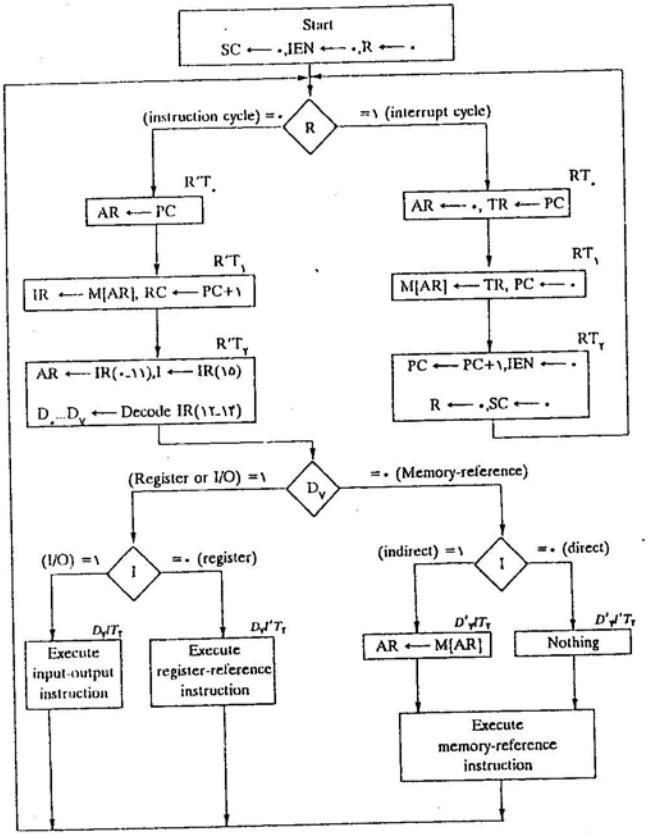
$D_7IT_3 = P$ (common to all input-output instructions)

$IR(i) = B_i (i = ۶, ۷, ۸, ۹, ۱۰, ۱۱)$

	P:	$SC \longleftarrow ۰$
INP	$pB_{۱۱}$:	$AC(۰\text{-}۷) \longleftarrow INPR, FGI \longleftarrow ۰$
OUT	$pB_{۱۰}$:	$OUTR \longleftarrow AC(۰\text{-}۷), FGO \longleftarrow ۰$
SKI	$pB_{۹}$:	$If(FGI=۱)\ then\ (PC \longleftarrow PC+۱)$
SKO	$pB_{۸}$:	$If(FGO=۱)\ then\ (PC \longleftarrow PC+۱)$
ION	$pB_{۷}$:	$IEN \longleftarrow ۱$
IOF	$pB_{۶}$:	$IEN \longleftarrow ۰$

$(A \oplus B) \oplus A \leq B$

$(A \oplus B) \oplus B \leq A$



شکل (۱-۱۲)

طراحی کامپیوتر پایه

کامپیوتر پایه دارای اجزاء زیر است:

۱) یک واحد حافظه ۲۰۹۶ کلمه‌ای و هر کلمه ۱۶ بیت

۲) ۸ ثبات: AR, PC, DR, AC, IR, TR, OUTR, INPR و شمارنده SC

۳) هفت فلیپ فلاپ: I, S, E, R, IEN, FGI و FGO

۴) دو رمزگشا: یک رمزگشای ۸ × ۳ برای کد دستورات و یک رمزگشا برای عملیات زمانبندی ۱۶×۴

۵) یک گذرگاه مشترک ۱۶ بیتی

۶) گیتهای منطقی کنترلی

۷) مدار منطقی و جمع‌کنندهٔ متصل به ورودی AC

دریچه‌های منطقی واحد کنترل

با توجه به شکل ۱.۷، بلاکی به نام (control logic gates) وجود دارد که ورودیهای آن بیتهای ۰ تا ۱۱ از IR، خروجیهای رمزگشای ۸ × ۳ دستور، خروجیهای رمزگشای ۱۶ × ۴ زمانبندی و بیت شماره ۱۵، IR می‌باشند. علاوه بر این ورودیها، ورودیهای دیگری نیز وجود دارند، مثل بیتهای AC برای تشخیص صفر بودن AC یا منفی بودن AC؛ بیتهای DR و مقدار هفت فلیپ فلاپ. خروجی‌های مدار کنترلی عبارتند از:

۱ـ سیگنالهای کنترل ورودی‌های ۹ ثبات

۲ـ سیگنالهایی برای خواندن و نوشتن از حافظه

۳ـ سیگنالهایی برای فعال نمودن[1] و پاک کردن یا مکمل کردن فلیپ فلاپ‌ها

۴ـ سیگنالهای مربوط به S_0 و S_1 و S_2 برای انتخاب یک ثبات برای گذرگاه

۵ـ سیگنالهای کنترل مدار منطقی و جمع‌کننده مربوط به AC

کنترل ثباتها و حافظه

ورودیهای کنترلی ثباتها عبارتند از LD (load)، INC (increment) و CLR (clear).

فرض کنید که می‌خواهیم ورودیهای کنترل AR را طراحی می‌کنیم. با جستجو در جدول ۶-۱، می‌توان جملاتی را که موجب تغییر در محتویات AR می‌شوند را پیدا کرد:

$R'T_0$: AR ⟵ PC

$R'T_2$: AR ⟵ IR(۰-۱۱)

D'_7IT_3: AR ⟵ M[AR]

RT_0: AR ⟵ ۰

D_5T_3: AR ⟵ AR+۱

1- Set

سه جمله اول مشخص‌کنندهٔ انتقال اطلاعات از یک ثبات یا حافظه به AR هستند. جمله چهارم AR را پاک می‌کند و جمله پنجم AR را با ۱ جمع می‌کند پس:

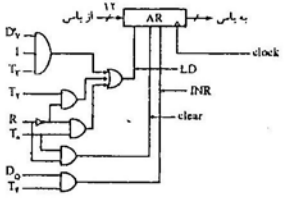
$LD(AR)=R'T_0+R'T_2+D'_7IT_3$

$CLR(AR)=RT_0$

$INR(AR)=D_5T_4$

یعنی به AR زمانی اطلاعات منتقل می‌شود (load) که یکی از توابع کنترلی، $R'T_0$، $R'T_2$ یا D'_7IT_3 فعال باشند.

شکل ۱.۱۴ گیتهای کنترلی مربوط به AR را نشان می‌دهد.



شکل (۱.۱۴)

به همین ترتیب می‌توان جملاتی که در سمت راست آنها عبارت M[AR] ⟵ وجود دارد را استخراج کرده، ورودی Read حافظه را طراحی کرد:

$Read = R'T_1+D'_7IT_3+(D_0+D_1+D_4+D_6)\,T_4$

کنترل فلیپ فلاپ‌ها

با توجه به جدول ۶-۱ مشاهده می‌شود که مثلاً محتویات فلیپ‌فلاپ IEN با دو دستور IOF و ION تغییر می‌کند:

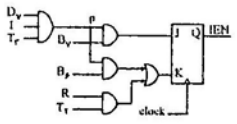
pB_7: IEN ⟵ ۱

pB_6: IEN ⟵ ۰

که $p=D_7IT_3$ و B_6 و B_7 بیتهای شماره ۶ و ۷ ثبات IR هستند. همچنین در انتهای سیکل وقفه نیز IEN پاک می‌شود:

RT_4: IEN ⟵ ۰

اگر از فلیپ‌فلاپ JK برای IEN استفاده کنیم، گیت کنترلی در شکل ۱۵-۱ نشان داده شد.



شکل (۱.۱۵)

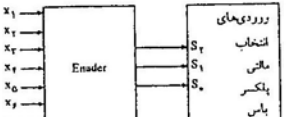
کنترل گذرگاه مشترک

گذرگاه مشترک با ورودیهای انتخاب S_0 و S_1 و S_2 کنترل می‌شود. جدول ۱.۷ مقادیر دودویی $S_2S_1S_0$ که هر ثبات را انتخاب می‌کند را نشان می‌دهد. هر عدد دودویی به یک متغیر بولی X_1 تا X_7 نسبت داده شده است. به عنوان مثال وقتی

$X_1=1$ مقدار $S_2S_1S_0$ باید ۰۰۱ باشد و AR روی باس قرار گیرد. جدول ۷ـ۱ در واقع جدول درستی یک اِنکدر است. شکل ۱.۱۶ نحوه قرار دادن انکدر برای تولید انتخاب‌های باس را نشان می‌دهد.

جدول (۸ـ۱)

ثباتی که برای باس انتخاب می‌شود.	خروجی‌ها			ورودیها						
	S_0	S_1	S_2	X_7	X_6	X_5	X_4	X_3	X_2	X_1
None	۰	۰	۰	۰	۰	۰	۰	۰	۰	۰
AR	۱	۰	۰	۰	۰	۰	۰	۰	۰	۱
PC	۰	۱	۰	۰	۰	۰	۰	۰	۱	۰
DR	۱	۱	۰	۰	۰	۰	۱	۰	۰	۰
AC	۰	۰	۱	۰	۰	۱	۰	۰	۰	۰
IR	۱	۰	۱	۰	۱	۰	۰	۰	۰	۰
TR	۰	۱	۱	۰	۱	۰	۰	۰	۰	۰
Memory	۱	۱	۱	۱	۰	۰	۰	۰	۰	۰



شکل (۱۶ـ۱)

توابع بولی برای انکدر عبارتند از:

$$S_2=X_1+X_3+X_5+X_7$$
$$S_1=X_2+X_3+X_6+X_7$$
$$S_0=X_4+X_5+X_6+X_7$$

برای یافتن تابع منطقی برای X_iها باید توابع کنترلی که ثبات مربوطه را روی باس قرار می‌دهند را تعیین کرد. مثلاً تابع منطقی که $X_1=1$ عبارت از $D_2T_2+D_5T_5$ چون ریزعملیات‌های $X_1=D_2T_2$: PC ⟵ AR و D_5T_5: PC ⟵ AR، AR را روی باس قرار می‌دهند. و یا

$$X_7=\bar{R}T_1+\bar{I}DIT_3+(D_0+D_1+D_2+D_6)T_2$$

تست‌ها

۱ـ یک کامپیوتر دیجیتال دارای واحد حافظه‌ای به ظرفیت ۸۱۹۲ کلمه ۵۴ بیتی می‌باشد. فرمت دستورات این کامپیوتر شامل ۵ بیت برای OP-CODE و ۱۳ بیت برای آدرس می‌باشد. جهت صرفه‌جوئی در سیکل‌های کامپیوتر هر سه عدد از این دستورات را بصورت یک کلمه حافظه پر می‌آوریم. اگر F=۰ مشخص‌کننده سیکل fetch و F=۱ مشخص‌کننده سیکل اجراء باشد و IR=Instruction Register، ۵۴ بیتی باشد و هدف صرفه‌جوئی در سیکل‌های fetch باشد توابع کنترل در قسمت‌های نقطه‌چین کدامیک از عبارت زیر می‌باشند؟ (مهندسی کامپیوتر ۷۱)

توجه کنید که R و I دو فلیپ‌فلاپ جهت انتخاب دستورات می‌باشند.

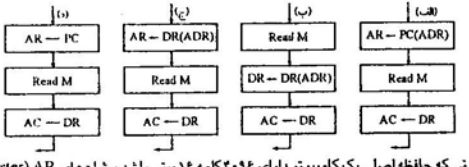
الف) $R'IFt_2$ ب) $R'I'Ft_2$ ج) $R'I'F't_2$ د) هیچکدام

$RI'Ft_2$ $RI'F't_2$ $R'IFt_2$

$RIFt_2$ $R'IF't_2$

$F't_0$: MAR ← PC
$F't_1$: MBR ← M , PC ← PC + ۱
$F't_2$: IR ← MBR
$F't_3$: F ← ۱ , R ← ۰ , I ← ۰
Ft_0 : MAR ← IR (AD۱) , OP ← IR (op۱)
Ft_1 : MBR ← M
Ft_2 : execute instruction
..... : IR (۱ـ۱۸) ← IR (۱۰ـ۲۶) , I ← ۱
..... : IR (۱ـ۱۸) ← IR (۲۷ـ۵۴) , R ← ۱
..... : F ← ۰ , R ← ۰ , I ← ۰

۲ـ با توجه به شمای زیر که مربوط به سیکل اجراء (Execute cycle) برخی از دستورات یک کامپیوتر می‌باشد. دستورالعمل‌های LOAD کدام یک از موارد زیر می‌باشد؟ (مهندسی کامپیوتر ۷۵)



۳ـ در صورتی که حافظه اصلی یک کامپیوتر دارای ۴۰۹۶ کلمه ۱۶ بیتی باشد و ثبات‌های AR (Address Register) و PC (Program Counter) و DR(Data Register) از جمله ثبات‌های این کامپیوتر باشند، کدام یک از گزینه‌های زیر اولاً نشان‌دهندهٔ تعداد بیت ثبات‌های DR, PC, AR بوده و ثانیاً نشان‌دهنده اجرای دستور مراجعه به حافظه BSA(Branch and Save Return Address) می‌باشد: (مهندسی کامپیوتر ۷۶)

۱۸ـ گزینهٔ (د) صحیح است.

next state= $\bar{x}\,\bar{y}+\bar{x}\,y\,\bar{Q}+x\bar{y}Q$

$\Rightarrow$ next state $=\bar{x}\,\bar{y}+\bar{x}\,\bar{Q}+x\bar{y}Q$

$=\bar{x}\,\bar{Q}+\bar{y}\,Q=(\bar{x}\wedge\bar{Q})\vee(\bar{y}\wedge Q)$

۱۹ـ گزینهٔ (ج) صحیح است. چون دستور انشعابی شرطی با توجه به اینکه پرش صورت می‌گیرد یا خیر زمان اجرا متفاوت است.

۲۰ـ گزینهٔ (ب) صحیح است.

۲۱ـ گزینهٔ (ب) صحیح است. سیکل اول واکشی دستور است که خواندن از حافظه است.

۲۲ـ گزینهٔ (ب) صحیح است.

$AX \leftarrow AX \oplus BX$

$BX \leftarrow (AX \oplus BX) \oplus BX = AX$

$AX \leftarrow (AX \oplus BX) \oplus BX = (AX \oplus BX) \oplus AX = 0 \oplus BX = BX$

۲۳ـ گزینهٔ (ج) صحیح است. مدار از ۰ تا ۹۹۹۹ می‌شمارد و چون فرکانس ۱۰۰۰ هرتز است مدت شمارش ۱۰ ثانیه می‌باشد که در هر ثانیه ۱۰۰۰ بار کلاک آمده است.

۲۴ـ گزینهٔ (ج) صحیح است. به اندازهٔ R_1، R_2 با خودش جمع شده و داخل R_2 قرار می‌گیرد.

فصل دوم

کنترل ریز برنامه‌ریزی شده

کنترل ریزبرنامه‌ریزی شده[1]

وظیفه واحد کنترل در یک کامپیوتر تولید ریزعملیات است. اگر سیگنالهای کنترلی توسط سخت‌افزار و با استفاده از تکنیکهای طراحی منطقی متداول، تولید شوند، گفته می‌شود واحد کنترل سیم‌بندی شده[2] است. طراحی واحد کنترل سیم‌بندی شده در فصل گذشته بررسی شد. ریزبرنامه‌ریزی روش دیگری برای طراحی واحد کنترل یک کامپیوتر است. این روش، روشی سیستماتیک و زیبا برای کنترل دنبالهٔ ریزعملیات می‌باشد.

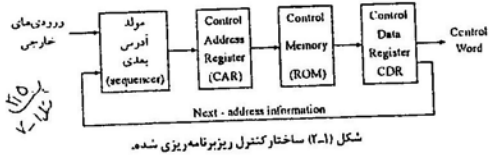
کلمه کنترلی[3]

متغیرهای کنترلی که توسط واحد کنترل تولید می‌شوند، دنباله‌ای از ۰ها و ۱ها هستند، که به آنها، کلمه کنترلی می‌گوییم. واحد کنترلی که متغیرهای کنترلی آن در یک حافظه ذخیره شده باشد، واحد کنترل ریزبرنامه ریزی شده می‌نامیم. هر کلمه در یک حافظه کنترلی شامل یک ریزدستور[4] می‌باشد. هر ریزدستور یک یا چند ریزعمل را برای سیستم مشخص می‌کند. دنباله‌ای از این ریزدستورات، ریزبرنامه[5] را تشکیل می‌دهد. از آنجایی که نیازی نخواهد بود که ریزبرنامه‌ها در زمانی که واحد کنترل ساخته می‌شود، عوض شوند، واحد کنترل می‌تواند یک حافظه فقط با قابلیت خواندن باشد (ROM).

روش ریزبرنامه‌ریزی دینامیک، اجازه می‌دهد که یک ریزبرنامه از یک حافظه جانبی به حافظه کنترلی بار شود. واحد کنترلی که از ریزبرنامه‌ریزی دینامیک استفاده می‌کند از یک حافظه کنترل قابل نوشتن استفاده می‌کند. این حافظه کنترل را می‌توان تغییر داد و ریزبرنامه‌ها را مجدداً نوشت.

توجه داشته باشید که حافظه کنترلی با حافظه اصلی سیستم متفاوت است. هر دستور که در حافظهٔ اصلی ذخیره شده است، موجب اجرای دنباله‌ای از ریزدستورات در حافظه کنترلی می‌شود، که این ریزدستورات، ریزعملیات مربوط به واکشی، ترجمه و اجرای دستور را تولید می‌کنند.

ساختار کلی یک واحد کنترل ریزبرنامه‌ریزی شده در شکل ۱ـ۲ آمده است.



شکل (۱ـ۲) ساختار کنترل ریزبرنامه‌ریزی شده.

1- Micro-programmed control
2- hardwired
3- Control word
4- microinstruction
5- microprogram

ثبات آدرس کنترل[1] (CAR)، آدرس ریزدستور را مشخص می‌کند و ثبات داده کنترل[2] (CDR) ریزدستوری که از حافظه کنترل خوانده می‌شود را در خود ذخیره می‌کند. CDR ریزدستور فعلی را در خود ذخیره می‌کند در حالیکه همزمان آدرس بعدی ریزدستور محاسبه می‌شود. به CDR گاهی ثبات پایپلاین[3] می‌گویند. این ثبات اجازه می‌دهد، ریزعملیات مشخص‌شده توسط کلمه کنترلی، همزمان با تولید ریزدستور بعدی اجرا شوند. در صورت استفاده از CDR، به یک کلاک با دو فاز احتیاج خواهیم داشت، یک کلاک برای CAR و یکی برای CDR.

مولد آدرس بعدی یا sequencer، آدرس ریزدستور بعدی در حافظه کنترل را تولید می‌نماید. آدرس ریزدستور بعدی می‌تواند آدرس ریزدستور فعلی بعلاوه یک باشد ولی این آدرس به طریق دیگر نیز محاسبه می‌شود. سیستم می‌تواند بدون استفاده از CDR با کلاک تک فاز که فقط به CAR اعمال می‌شود، کار کند. در این فصل، CDR را حذف می‌کنیم بنابراین فقط CAR احتیاج به کلاک خواهد داشت.

مزیت اصلی کنترل ریزبرنامه‌ریزی شده، انعطاف‌پذیری آن است، یعنی اگر بخواهیم کنترل بر سیستم را تغییر دهیم، تنها کافیست محتویات حافظه کنترل را تغییر دهیم. ولی در روش سیم‌بندی باید کل سخت‌افزار تغییر کند. البته سرعت روش سیم‌بندی از ریزبرنامه‌ریزی بیشتر است.

وقتی CDR داریم ولی ترانیم دیگر نگذاریم فاز اطلاعات در سیم‌ها از تولید آدرس بعدی آن CDR را بکشیم بگیریم CDR ما

تولید آدرس ریزدستور در سیستم‌های ابتدائی CDR داریم ولی ترانیم از تاخیر مولد آدرس بعدی جلوگیری می‌کند

ریزدستورات در حافظه کنترلی، بصورت گروهی ذخیره شده‌اند و هر گروه مربوط به یک دستور است یا مربوط به سیکل مثلاً واکشی دستور، یا سیکل پیدا کردن آدرس موثر دستور. به هر گروه، روتین[4] گفته می‌شود. هر دستور کامپیوتر، یک روتین مشخص در حافظه کنترل دارد که این روتین شامل ریزدستوراتی است که این ریزدستورات، ریزعملیات‌های مربوط به آن دستور را تولید می‌کنند.

اولین روتینی که از حافظه کنترل اجرا می‌شود، روتین واکشی دستور است. پس از اتمام واکشی، حافظه کنترل، باید روتینی که آدرس موثر عملوند را می‌یابد اجرا کند، و در پایان روتین مربوط به اجرای دستور اجرا می‌شود. هر دستور، ریزبرنامه مربوط به خود را در حافظه کنترل دارد و با توجه به کد عملیاتی دستور آن ریزبرنامه اجرا می‌شود. تبدیل کد عملیاتی دستور به یک آدرس در حافظه کنترل را عمل نگاشت[5] گویند.

در کل، آدرس ریزدستور بعدی در حافظه کنترل به یکی از طرق زیر بدست می‌آید:

۱ـ جمع CAR با یک

۲ـ پرش شرطی یا غیرشرطی درون حافظه کنترل (مثلاً اگر دستور حافظه‌ای غیرمستقیم باشد، باید به رونین یافتن آدرس موثر پرش کرد.)

۳ـ نگاشت

۴ـ صدا کردن و بازگشتن از زیرروال‌های داخل حافظه کنترل

1- Control Address Register
2- Control Data Register
3- Pipeline
4- Routine
5- Mapping

ساختار کامپیوتر

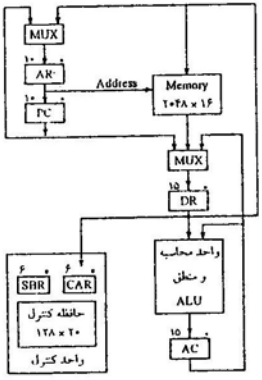
بلوک دیاگرام کامپیوتر مورد بحث در شکل ۲ـ۲ نشان داده شده است. این کامپیوتر دارای دو واحد حافظه است: یک حافظه اصلی برای ذخیره دستورات و داده، و یک حافظه کنترل برای ذخیره ریزبرنامه‌ها. چهار ثبات به واحد پردازنده و دو ثبات به واحد کنترل مربوط هستند. ثباتهای پردازنده عبارتند از: شمارنده برنامه (PC)، ثبات آدرس (AR)، ثبات داده (DR)، و اکومولاتور AC. واحد کنترل دارای دو ثبات است که عبارتند از: (CAR) و (SLR)[1]. ثبات SBR، ثبات زیرروال است و آدرس بازگشت در آن ذخیره می‌شود.

انتقال اطلاعات در این کامپیوتر از طریق باس مشترک صورت نمی‌گیرد. با توجه به شکل DR می‌تواند از AC، PC، یا حافظه اطلاعات دریافت کند. AR می‌تواند از PC یا DR اطلاعات بگیرد و PC از AR. واحد محاسبه و منطق (ALU) نیز ریزعملیات مربوطه را روی داده‌های AC و DR انجام داده و نتیجه را به AC منتقل می‌کند. ثبات AR، آدرس را برای حافظه آماده می‌کند. فقط محتویات ثبات DR را می‌توان به حافظه نوشت و حافظه نیز فقط به DR داده منتقل می‌کند.

حافظه اصلی دارای 2^{11} کلمه ۱۶ بیتی است پس ثبات‌هایی که آدرس را ذخیره می‌کنند (PC,AR) ۱۱ بیتی و ثباتهایی که با داده سروکار دارند ۱۶ بیتی هستند.

حافظه کنترل نیز دارای 2^{7} کلمه می‌باشد که احتیاج به ۷ خط آدرس دارد و ثبات‌های CAR و SBR که با آدرس حافظه کنترل سر و کار دارند ۷ بیتی هستند.

ساختار واحد کنترل در شکل ۲ـ۲ آمده است.



شکل (۲ـ۲)

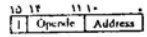
فرمت دستورالعمل

فرمت دستور در این ماشین مطابق شکل (۳ ـ ۲) می‌باشد. دستورات در این ماشین دارای ۲ فیلد هستند. فیلد آدرس که ۱۱ بیتی است، فیلد کد عملیاتی ۴ بیتی و فیلد مُد آدرس‌دهی که یک بیتی است. شکل (۳ ـ ۲)

1- Subroutine Register

symbol	opcode	توصیف
ADD	•••	AC ← AC+M[EA]
BRANCH	•••۱	IF (AC<•) then (PC ← EA)
STORE	••۱•	M[EA] ← AC
EXCHANGE	••۱۱	AC ← M[EA], M[EA] ← AC

(b) چهار دستور کامپیوتر

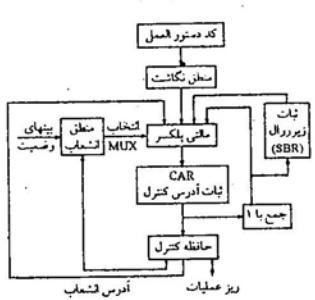


(a) فرمت دستور

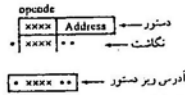
شکل (۳ـ۲)

ساختار واحد کنترل

با توجه به شکل (۳ـ۲)، ثبات CAR از طریق مالتی‌پلکسر از ۴ طریق بار می‌شود. یکی از راههای یافتن آدرس ریزدستور در حافظه کنترل، عمل نگاشت[1] است. نگاشت یعنی تبدیل کد عملیاتی دستور به یک آدرس در حافظه کنترل. از آنجایی که کد عملیاتی دستور در این ماشین ۴بیتی است و آدرس برای حافظه کنترل باید ۷ بیتی باشد، سه بیت باید به کد عملیاتی افزود تا آدرس حافظه کنترل بدست آید. شکل (۵ ـ ۲) عمل نگاشت را نشان می‌دهد.



شکل (۴ـ۲)



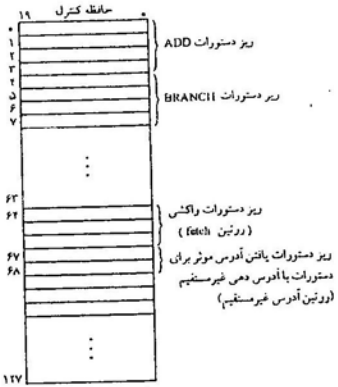
شکل (۵ـ۲) نگاشت از کد دستور به آدرس ریزدستور

با توجه به شکل b-۳-۲، دستور ADD دارای کد عملیاتی •••• است که با عمل نگاشت به ••••••• تبدیل می‌شود که آدرس اولین ریزدستور مربوط به ADD در حافظه کنترل است یا دستور STORE دارای کد عملیاتی ••۱• می‌باشد که پس از نگاشت •••۱••• شده و آدرس اولین ریزدستور مربوط به STORE یعنی آدرس روتین STORE در حافظه کنترل را بدست می‌دهد.

با توجه به فرمول نگاشت هر دستور دارای ۴ ریزدستور می‌باشد. که مطابق شکل (۶ ـ ۲) در حافظه کنترل قرار گرفته‌اند. از آنجایی که کد عملیاتی در این ماشین ۴ بیتی است در کل می‌توان 2^4 یعنی ۱۶ دستور برای این ماشین تعریف

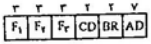
1- mapping

کرد، و با توجه به اینکه هر دستور دارای ۴ ریزدستور است در کل ۶۴=۱۶×۴ ریزدستور مورد نیاز است یعنی آدرسهای ۰ تا ۶۳ از حافظه کنترل، می‌دانیم حافظه کنترل ۱۲۸ کلمه دارد که ۶۴ کلمه آن استفاده شده است. اما ۶۴ کلمه باقی‌مانده، برای روتین‌های مشترک برای همهٔ دستورات مثل روتین واکشی یا یافتن آدرس موثر مورد استفاده قرار می‌گیرند و با این حال مقداری از حافظه کنترل بلا استفاده می‌ماند که می‌توان از آن به منظور افزودن دستورات دیگر به ماشین بهره جست.



شکل (۶ـ۲)

هر ریزدستور یک کلمه ۲۰ بیتی است. فرمت ریزدستور در شکل (۷ ـ ۲) آمده است.



شکل (۷ـ۲) فرمت کد ریزدستور

فیلدهای ریز عملیات : F_1, F_2, F_3

CD : Condition for branching

BR : Branch field

AD : Address field

فیلدهای F_1, F_2, F_3 مشخص‌کننده ریزعملیات برای کامپیوتر هستند. فیلد CD، تعیین‌کننده بیت وضعیت می‌باشد. فیلد BR مشخص می‌کند چگونه می‌توان داخل حافظه کنترل انشعاب داشت. فیلد AD شامل آدرس انشعاب است.

هر ریزدستور دارای ۳ فیلد ریزعمل است. هر فیلد ریزعمل ۳ بیتی است که می‌تواند ۷ ریزعمل را مشخص کند (حالت ••• هیچ ریزعملی را مشخص نمی‌کند) و جمعاً ۲۱ ریزعمل قابل تعریف است. هر ریزدستور حداکثر می‌تواند ۳ ریزعمل را اجرا کند. اگر تعداد کمتری ریزعمل نیاز داشت، یک یا بیشتر از فیلدهای ریزعمل دارای کد ••• خواهند بود. جدول (۱ـ۲) انواع ریزعملها را همراه سایر فیلدها نشان می‌دهد. به عنوان مثال، یک ریزدستور که F_1=••• و F_2=۱۰۰ و F_3=۱۰۱ می‌باشد ریزعملهای M[AR] ← DR و PC ← PC+1 را مشخص کند (F_1 هیچ ریزعملی را مشخص نمی‌کند) باید توجه داشت که ریزعملها متناقض نباشند مثلاً ۰۱۰۰۰۱۰۰۰ هیچ مفهومی ندارد زیرا مشخص می‌کند که AC ← • و در عین حال AC-DR ← AC که دو ریزعمل متناقض هستند. در جدول ۱ـ۲، سمبولهایی برای

ریزعملیاتها مشخص شده است. مثلاً ریزعمل DR ⟵ AC (F_1=۱۰۰) با سمبل DRTAC نشان داده شده است.

فیلد CD (شرط) شامل ۲ بیت است که ۴ نوع وضعیت را مشخص می‌کند. اگر CD=۰۰ باشد شرط همواره ۱ است یعنی با ۰۰ شدن CD شرط صحیح است و با صحیح بودن شرط و با توجه به فیلد BR یک عمل انشعاب غیرشرطی را مشخص می‌کند. فیلد BR (انشعاب) شامل ۲ بیت است. اگر BR=۰۰ باشد، واحد کنترل یک پرش به آدرسی که AD مشخص می‌کند، انجام می‌دهد. وقتی BR=۰۱ یک عمل صدازدن (CALL) زیر روال در حافظه کنترل صورت می‌گیرد. صداکردن زیرروال موجب ذخیره شدن آدرس بازگشت در SBR می‌شود ولی در پرش، آدرس بازگشت ذخیره نمی‌شود. پرش و صدازدن زیرروال بستگی به شرط مشخص‌شده توسط CD دارد. اگر شرط ۱ باشد، AD به CAR منتقل می‌شود ولی اگر شرط ۰ باشد CAR با یک جمع می‌شود و آدرس بعدی بدست می‌آید.

روتین واکشی

بهترین مکان برای روتین واکشی در حافظه کنترل آدرس ۶۴است. ریزدستوراتی که برای روتین واکشی نیاز هستند عبارتند از:

AR ⟵ PC

DR ⟵ M[AR], PC ⟵ PC+۱

AR ⟵ DR(۰-۱۰), CAR(۲-۵) ⟵ DR(۱۱-۱۴), CAR(۰-۱-۶) ⟵ ۰

آدرس دستور، ابتدا از PC به AR منتقل شده و دستور از حافظه خوانده شده و به DR منتقل می‌شود و سپس PC به دستور بعدی اشاره می‌کند. پس از آن فیلد آدرس DR به AR منتقل شده و سپس فیلد کد عملیاتی به یک آدرس در حافظه کنترل نگاشته می‌شود تا ریزدستورات مربوط به دستور موردنظر اجرا شوند.

روتین واکشی احتیاج به ۳ ریزدستور دارد که مطابق شکل (۸-۲) از آدرس ۶۴ در حافظه کنترل قرار گرفته‌اند. در قسمت b شکل ۸-۲ ریزبرنامه واکشی را با اطلاعات سمبولیک می‌بینید.

آدرس دودویی	F_1	F_2	F_3	CD	BR	AD
۱۰۰۰۰۰۰	۱۱۰	۰۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۰۰۱
۱۰۰۰۰۰۱	۰۰۰	۱۰۰	۱۰۱	۰۰	۰۰	۱۰۰۰۰۱۰
۱۰۰۰۰۱۰	۱۰۱	۰۰۰	۰۰۰	۰۰	۱۱	۰۰۰۰۰۰۰

شکل (۸-۲a) ریزدستورات fetch در حافظه کنترل

```
        ORG    ۶۴
FETCH:  PCTAR U JMP NEXT
        READ, INCPC U JMP NEXT
        DRTAR U MAP
```

شکل (۸-۲b) ریزبرنامه واکشی با علائم سمبولیک

اجرای ریزدستور سوم در روتین fetch (یعنی عمل نگاشت MAP) موجب یک پرش به آدرس ۰۰xxxx۰۰ می‌شود، که xxxx چهار بیت کد عملیاتی هستند. مثلاً اگر دستور ADD باشد که کد عملیاتی آن ۰۰۰۰ است، آدرسی که به CAR منتقل می‌شود ۰۰۰۰۰۰۰ می‌باشد. که این آدرس، شروع روتین ADD در حافظه کنترل است.

جدول (۱-۲) سمبولها وکدهای باینری برای فیلدهای ریزدستور

F_1	Microoperation	Symbol
۰۰۰	None	NOP
۰۰۱	AC ⟵ AC+DR	ADD
۰۱۰	AC ⟵ ۰	CLRAC
۰۱۱	AC ⟵ AC+۱	INCAC
۱۰۰	AC ⟵ DR	DRTAC
۱۰۱	AR ⟵ DR(۰-۱۰)	DRTAR
۱۱۰	AR ⟵ PC	PCTAR
۱۱۱	M[AR] ⟵ DR	WRITE

F_2	Microoperation	Symbol
۰۰۰	None	NOP
۰۰۱	AC ⟵ AC-DR	SUB
۰۱۰	AC ⟵ AC∨DR	OR
۰۱۱	AC ⟵ AC∧DR	AND
۱۰۰	DR ⟵ M[AR]	READ
۱۰۱	DR ⟵ AC	ACTDR
۱۱۰	DR ⟵ DR+۱	INCDR
۱۱۱	DR(۰-۱۰) ⟵ PC	PCTDR

F_3	Microoperation	Symbol
۰۰۰	None	NOP
۰۰۱	AC ⟵ AC⊕DR	XOR
۰۱۰	AC ⟵ $\overline{AC}$	COM
۰۱۱	AC ⟵ shl AC	SHL
۱۰۰	AC ⟵ shr AC	SHR
۱۰۱	PC ⟵ PC+۱	INCPC
۱۱۰	PC ⟵ AR	ARTPC
۱۱۱	Reserved	

CD	Condition	Symbol	Comments
۰۰	Always=۱	U	Unconditional branch
۰۱	DR(۱۵)	I	Indirect address bit
۱۰	AC(۱۵)	S	Sign bit of AC
۱۱	AC=۰	Z	Zero value in AC

BR	Symbol	Function
۰۰	JMP	CAR ⟵ AD if condition=۱
		CAR ⟵ CAR+۱ if condition=۰
۰۱	CALL	CAR ⟵ AD,SBR ⟵ CAR+۱ if condition=۱
		CAR ⟵ CAR+۱ if condition=۰
۱۰	RET	CAR ⟵ SBR(Return from subroutione)
۱۱	MAP	CAR(۲-۵) ⟵ DR(۱۱-۱۴), CAR(۰-۱-۶) ⟵ ۰

آدرس شروع روتین BRANCH در حافظه کنترل ۰۰۰۰۱۰۰ می‌باشد چون کد عملیاتی این دستور ۰۰۱ است. و روتین‌های سایر دستورات به ترتیب در آدرسهای ۸،۱۲،۱۶،۲۰...و ۶۰ قرار دارند.

در هر روتین باید ریزدستوراتی داشت که بتوان آدرس موثر را محاسبه کرد. مد آدرس‌دهی غیرمستقیم در تمام دستورات مراجعه به حافظه وجود دارد. اگر ریزدستورات مربوط به یافتن آدرس موثر را به عنوان یک زیرروال در حافظه کنترل قرار دهیم، در کلمات حافظه کنترل صرفه‌جویی کرده‌ایم. روتین محاسبه آدرس موثر، پس از روتین fetch و در آدرس ۶۷ حافظه کنترل قرار دارد. این روتین به نام INDRCT است که در جدول (۲-۲) آمده است.

برای آنکه نحوه پرش و بازگشت از زیرروال مربوط به یافتن آدرس موثر (INDRCT) را ببینیم، فرض کنید که ریزدستور MAP در انتهای روتین fetch موجب یک پرش به آدرس ۰، یعنی جایی که روتین ADD ذخیره شده است، شده است. اولین دستور روتین ADD، با توجه به بیت وضعیت I به زیرروال INDRCT پرش می‌کند. اگر I=۱، یک پرش به INDRCT اتفاق می‌افتد و آدرس بازگشت (آدرس ۱ در این حالت) در ثبات SBR[1] ذخیره می‌گردد. زیرروال INDRCT دو ریزدستور دارد:

INDRCT: READ U JMP NEXT

DRTAR U RET

جدول (۲-۲)

Label	Microoperations	CD	BR	AD
	ORG ۰			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG ۴			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG ۸			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG ۱۲			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR,DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG ۴۴			
FETCH:	PCTAR	U	JMP	NEXT
	READ,INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

1- SUBroutine Register

با توجه به این ریزدستورات، آدرس موثر پیدا شده و به AR منتقل می‌شود. و سپس ریزدستور RET موجب می‌شود SBR به CAR منتقل شود که این موجب بازگشت به آدرس ۱ حافظه کنترل یعنی ریزدستور دوم دستور ADD می‌شود. پس از آن، ریزدستورات ADD اجرا می‌شوند و در پایان ریزدستورات ADD پرشی به روتین fetch صورت می‌گیرد تا دستور بعدی از حافظه، fetch شده و همین روال برای آن تکرار شود.

دستور BRANCH در صورتی که AC<۰ باشد به آدرس موثر پرش می‌کند. دستور BRANCH بیت وضعیت S را چک می‌کند و اگر S=۱ ریزدستور JMP مشخص شده در جدول ۲-۲، موجب پرش به محل OVER می‌شود که ریزدستور واقع در این مکان در صورتی که I=۱ باشد پرشی به روتین INDRCT انجام می‌دهد. سپس آدرس موثر از AR به PC منتقل شده و ریزبرنامه به روتین fetch بازمی‌گردد.

ریزبرنامه‌ها بر حسب مقادیر دودویی

معادل باینری ریزبرنامه‌ها در جدول (۲-۲) لیست شده است. در جدول (۲-۲) دقت کنید که آدرس ۲ هیچ معادلی ندارد یعنی هیچ ریزدستوری را مشخص نمی‌کند. در واقع دستور ADD دارای ۲ ریزدستور بیشتر نیست که این ریزدستورات در آدرسهای ۰ و ۱ و ۲ حافظه کنترل قرار دارند.

اگر چه آدرس ۲ استفاده نمی‌شود، ولی باید مقدار دودویی داشته باشد. می‌توان همهٔ بیتهای آن را صفر کرد. ولی اگر اشتباهاً ثبات CAR شامل مقدار ۲ شود، بهتر است از آن آدرس به روتین fetch پرش کند که در جدول این مسئله در نظر گرفته شده است.

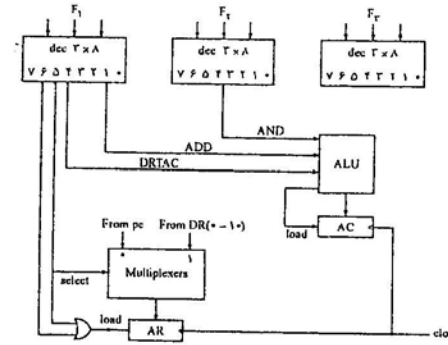
جدول (۳-۲)

	Address		Binary Microinstruction					
Micro Routine	Decimal	Binary	F۱	F۲	F۳	CD	BR	AD
ADD	۰	۰۰۰۰۰۰۰	۰۰۰	۰۰۰	۰۰۰	۰۱	۰۱	۱۰۰۰۰۱۱
	۱	۰۰۰۰۰۰۱	۰۰۰	۱۰۰	۰۰۰	۰۰	۰۰	۰۰۰۰۰۱۰
	۲	۰۰۰۰۰۱۰	۰۰۱	۰۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۰۰۰
	۳	۰۰۰۰۰۱۱	۰۰۰	۰۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۰۰۰
BRANCH	۴	۰۰۰۰۱۰۰	۰۰۰	۰۰۰	۰۰۰	۱۰	۰۰	۰۰۰۰۱۱۰
	۵	۰۰۰۰۱۰۱	۰۰۰	۰۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۰۰۰
	۶	۰۰۰۰۱۱۰	۰۰۰	۰۰۰	۰۰۰	۰۱	۰۱	۱۰۰۰۰۱۱
	۷	۰۰۰۰۱۱۱	۰۰۰	۰۰۰	۱۱۰	۰۰	۰۰	۱۰۰۰۰۰۰
STORE	۸	۰۰۰۱۰۰۰	۰۰۰	۰۰۰	۰۰۰	۰۱	۰۱	۱۰۰۰۰۱۱
	۹	۰۰۰۱۰۰۱	۰۰۰	۱۰۱	۰۰۰	۰۰	۰۰	۰۰۰۱۰۱۰
	۱۰	۰۰۰۱۰۱۰	۱۱۱	۰۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۰۰۰
	۱۱	۰۰۰۱۰۱۱	۰۰۰	۰۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۰۰۰
EXCHANGE	۱۲	۰۰۰۱۱۰۰	۰۰۰	۰۰۰	۰۰۰	۰۱	۰۱	۱۰۰۰۰۱۱
	۱۳	۰۰۰۱۱۰۱	۰۰۱	۰۰۰	۰۰۰	۰۰	۰۰	۰۰۰۱۱۱۰
	۱۴	۰۰۰۱۱۱۰	۱۰۰	۱۰۱	۰۰۰	۰۰	۰۰	۰۰۰۱۱۱۱
	۱۵	۰۰۰۱۱۱۱	۱۱۱	۰۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۰۰۰
FETCH	۴۴	۱۰۰۰۰۰۰	۱۱۰	۰۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۰۰۱
	۴۵	۱۰۰۰۰۰۱	۰۰۰	۱۰۰	۱۰۱	۰۰	۰۰	۱۰۰۰۰۱۰
	۴۶	۱۰۰۰۰۱۰	۱۰۱	۰۰۰	۰۰۰	۰۰	۱۱	۰۰۰۰۰۰۰
INDRCT	۴۷	۱۰۰۰۰۱۱	۰۰۰	۱۰۰	۰۰۰	۰۰	۰۰	۱۰۰۰۱۰۰
	۴۸	۱۰۰۰۱۰۰	۱۰۱	۰۰۰	۰۰۰	۰۰	۱۰	۰۰۰۰۰۰۰

طراحی واحد کنترل

شکل (۹ـ۲) نشان می‌دهد که F_1 و F_2 و F_3 از طریق سه رمزگشا، سیگنالهای کنترلی را تولید می‌کنند. هر کدام از خروجی‌های رمزگشاها باید به مدارات مناسب متصل شوند. به عنوان مثال، وقتی F_1=۱۰۱ باشد، در کلاک بعد، محتویات DR(۰ـ۱۰) به AR منتقل می‌شود و یا اگر F_1=۱۱۰ محتویات PC به AR منتقل می‌شود. مطابق شکل (۹ـ۲) خروجی‌های ۵ و ۶ رمزگشای F_1 به ورودی بار (Load) ثبات AR متصل هستند و سایر اتصالات نیز طوری است که عملیات مورد نظر صورت پذیرد.

مشاهده می‌شود که بجای استفاده از گیتها برای تولید سیگنالهای کنترلی AND و ADD و... مطابق فصل قبل، این ورودیها از خروجی‌های رمزگشاها تولید می‌شوند.



شکل (۹ـ۲) رمزگشایی فیلدهای کد عملیاتی

تست‌ها

۱ـ دلیل استفاده از ریزبرنامه‌سازی (Microprogramming) در طراحی واحد کنترل CPU چیست؟ (مهندسی کامپیوتر ۷۱)

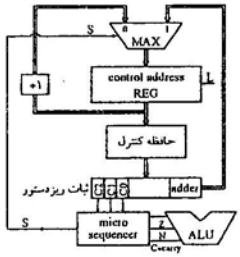
الف) طراحی واحد کنترل سیستماتیک شود.

ب) بتوان به آسانی خطاهای طراحی را برطرف نمود و فهرست دستورهای زبان ماشین را تغییر داد.

ج) بتوان برنامه نوشته‌شده به زبان ماشین کامپیوتر دیگری را اجرا نمود.

د) هر سه مورد

۲ـ شکل زیر قسمتی از مدار کنترل ریزبرنامه ریزی‌شده یک کامپیوتر را نشان می‌دهد. معادله S، خروجی microsequencer عبارت است از: (مهندسی کامپیوتر ۷۱)



الف) $S=C_1C_0+N+Z+carry$

ب) $S=C_1\bar{C}_0Z+\bar{C}_1C_0N+C_2\bar{C}_1\bar{C}_0carry$

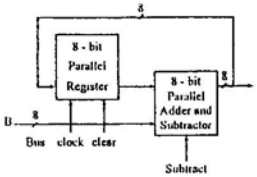
ج) $S=C_1C_0+C_1\bar{C}_0Z+\bar{C}_1C_0N+C_2\bar{C}_1\bar{C}_0carry$

د) $S=(N+Z+carry)(C_1+C_2+C_0)$

فرضهای مسئله:

$C_2C_1C_0$	معنی	
۰۰۰=۰:	ریزدستور بعدی	
۰۰۱=۱:	Jump if	N=۱
۰۱۰=۲:	Jump if	Z=۱
۰۱۱=۳:	Jump	
۱۰۰=۴:	Jump if	C=۱
۱۰۱=۵:	Jump if	N=۱
۱۱۰=۶:	Jump if	Z=۱
۱۱۱=۷:	Jump	

۳ـ نمودار بلوکی مدار و سیگنال‌های کنترل لازم برای انجام عملیات جمع و تفریق شکل زیر نشان داده شده است. در صورتی که در طراحی واحد کنترل آن از روش ریز برنامه‌سازی استفاده کنیم، تعداد ریز دستورالعمل‌های (Microinstruction) لازم برای انجام عمل A-B را به دست آورید. (مهندسی کامپیوتر ۷۳)



الف) یک ریز دستورالعمل

ب) دو ریز دستورالعمل

ج) سه ریز دستورالعمل

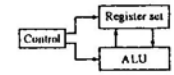
د) چهار ریز دستورالعمل

فصل سوم

Central Processing Unit

مقدمه

واحد پردازش مرکزی (CPU) قسمتی از کامپیوتر است که قسمت اعظم پردازش داده‌ها در آن انجام می‌شود. می‌توان گفت CPU از سه قسمت اصلی تشکیل شده است، که در شکل (۱ـ۳) آمده است.



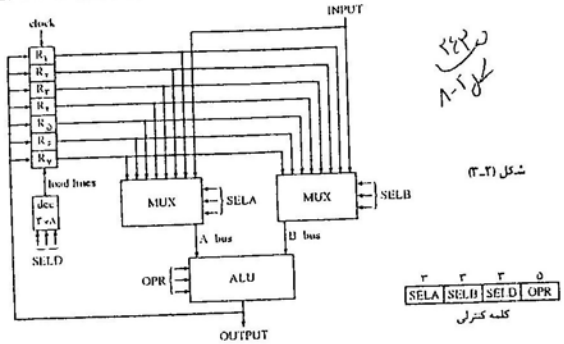
شکل (۱ـ۳) اجزای اصلی CPU

ثباتها داده‌های موقت را که برای اجرای دستورات نیاز هستند، در خود ذخیره می‌کنند. البته می‌توان این داده‌ها را در حافظه اصلی نیز ذخیره کرد، ولی حافظه یکی از کندترین قطعات در کامپیوتر است. در این بخش مباحثی که برنامه‌نویس با آن سروکار دارد (ISA) مطرح می‌شود.

سازمان ثباتهای عمومی

CPU برای انجام اعمال خود، نیاز به ذخیره اطلاعات و بازیابی به موقع آنها دارد. مثلاً برای نگهداری آدرس بازگشت در فراخوانی برنامه‌ها و یا ذخیره نتایج موقت محاسبات و... این حافظه مورد نیاز اگر در خارج از CPU باشد، اتلاف زمان زیادی برای CPU در پی دارد و بهتر این است که در خود CPU قرار گیرد، و البته ظرفیت آن محدود خواهد بود. یک یا چند ثبات درون CPU این حافظه را تشکیل می‌دهند. ثباتهای داخل CPU باید با یکدیگر در ارتباط باشند. شکل (۲ـ۳) ساختار این ارتباط را نشان می‌دهد.

در این شکل، خروجی ثباتها از طریق دو MUX به ALU وارد می‌شود و ALU عملیات خود را روی دو ثبات ورودی انجام می‌دهد. خروجی ALU نیز به ورودی ثباتها متصل است که از طریق یک رمزگشا، یکی از ثباتها، بار می‌شود. کلمه کنترلی که در شکل مشاهده می‌شود، تمام ثباتها و همچنین عملیات داخل ALU را کنترل می‌کند.



شکل (۲ـ۳)

به عنوان نمونه برای انجام ریزعمل $R_1 \longleftarrow R_2+R_3$ در شکل (۲ـ۳)، ابتدا SELA محتوای R_2 و SELB محتوای R_3 را به خروجی MUXها می‌برند. OPR نیز عمل جمع را برای ALU مشخص می‌کند. و SELD نیز ورودی بار ثبات R_1 را فعال می‌کند تا خروجی ALU به R_1 منتقل شود. SELA و SELB که یکی از هفت ثبات را انتخاب می‌کنند می‌توانند ورودی هشتمی را نیز انتخاب کنند. این ورودی حافظه است. OPR، پنج بیت است و می‌تواند ۳۲ ریزعمل را برای ALU مشخص کند. مثلاً برای انجام ریزعمل $R_1 \longleftarrow R_2-R_3$ کافیست کلمه کنترلی به شکل زیر باشد.

SELA	SELB	SELD	OPR
۰۱۰	۰۱۱	۰۰۱	۰۰۱۰۱

که OPR=۰۰۱۰۱ عمل تفریق را مشخص می‌کند. کلمه کنترلی که معرفی شد می‌تواند از حافظه کنترل بیاید.

سازمان پشته[1]

پشته حافظه‌ای است که اطلاعات را به نحوی ذخیره می‌کند که آخرین عنصر ذخیره شده، اولین عنصر قابل برداشت است؛ یعنی دارای یک ساختار آخرین ـ ورودی ـ اولین ـ خروجی[2] (LIFO) می‌باشد. پشته می‌تواند خود حافظه‌ای جداگانه باشد و یا بخشی از حافظه اصلی کامپیوتر را به عنوان پشته تعریف کرد. دو عمل روی پشته قابل تعریف است: عمل PUSH که موجب درج عنصری در بالای پشته می‌شود و عمل POP که موجب حذف عنصر از بالای پشته می‌گردد. ثباتی که آدرس پشته را در خود ذخیره می‌کند، ثبات پشته[3] (SP) نام دارد.

SP می‌تواند به خانه پر بالای پشته اشاره کند یا به خانه خالی، که این بستگی به پیاده‌سازی دارد.

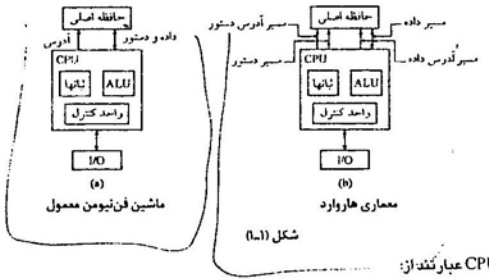
شکل (۳ـ۳)، ساختار یک پشته ۶۴ کلمه‌ای را نشان می‌دهد که به صورت حافظه‌ای مجزا طراحی شده است. به این نوع پیاده‌سازی پشته، پشته ثباتی[4] نیز می‌گویند.

1- Stack
2- Last-In First-Out
3- Stack-Pointer
4- Register stack

- دارای سه زیرسیستم اساسی سخت‌افزاری است:
  - یک CPU
  - یک سیستم حافظه اصلی
  - و سیستم I/O
- این ماشین یک کامپیوتر با برنامه ذخیره شده است. حافظه اصلی شامل برنامه‌ای است که عملیات کامپیوتر را کنترل می‌کند.
- این ماشین دستورات را ترتیبی اجرا می‌کند. CPU در یک لحظه فقط یک عمل را اجرا می‌کند.
- این ماشین یک مسیر منحصر بفرد بین حافظه اصلی و واحد کنترل پردازنده دارد.

ماشین‌های فن‌نیومن معمول (شکل a ۱.۱) یک مسیر برای آدرس و یکی برای داده و دستور دارند.

معماری‌های هاروارد[1] (شکل b ۱.۱) گونه‌ای از ماشین‌های فن‌نیومن هستند که مسیرهای مجزایی برای آدرس داده، داده، آدرس دستور و دستور دارند. در معماری هاروارد می‌توان به داده و دستور همزمان دسترسی پیدا کرد.



شکل (۱-۱)

اجزای اصلی یک CPU عبارتند از:

- واحد کنترل[2] CU، که عملیات کامپیوتر را کنترل می‌کند.
- واحد محاسبه و منطق[3] (ALU)، که عملیات حسابی، منطقی و شیفت[4] را انجام می‌دهد.
- یک مجموعه ثبات، که مقادیر مختلفی را در حین عملیات کامپیوتر ذخیره می‌کنند.
- یک شمارندهٔ برنامه[5] (PC) (گاهی شمارندهٔ دستور[6] می‌گویند) که آدرس یک دستور در حافظهٔ اصلی را در خود دارد. PC قسمتی از مجموعهٔ ثباتها میباشد.

یک ماشین فن‌نیومن یک مدل انتزاعی از کامپیوتر است که دستورات[7] را اجرا می‌کند. هر دستور دارای میدان‌های (فیلدهای)[8] مختلفی است، که محتویات هر فیلد برای واحد کنترل جزئیاتی را فراهم می‌کند، و هر دستور فرمت خاص خود را دارد، که این فرمت مشخص‌کنندهٔ نحوهٔ قرار گرفتن فیلدهاست.

1- Harward
2- Control Unit
3- Arithmatic & Logic unit
4- Shift
5- Program counter
6- Instruction Counter
7- Instructions
8- Field

برای دستوراتی که روی داده کار می‌کنند (مثلاً دستورات حسابی)، داده را عملوند[1] آن عملیات می‌گویند. مجموعه دستور[2] یک کامپیوتر، دستوراتی است که کامپیوتر می‌تواند اجرا کند.

هر دستور دارای یک کد عملیاتی[3] (opcode) است که فیلدی از دستور است و مشخص‌کنندهٔ نوع عمل است. دیگر فیلدهای دستور مشخص می‌کند که عمل روی کدام ثبات‌ها یا مکانهای حافظه باید صورت گیرد. دستورات همچنین به کامپیوتر می‌گویند که کدام بیت‌های وضعیت باید تست شود، یا چه بیتهایی باید ست[4] شود، و همچنین هنگام بروز خطا، پردازنده چه کند. (بیتهای وضعیت پردازنده را که پرچم‌ها[5] نیز می‌گویند ثباتهای یک بیتی داخل پردازنده هستند.)

یک برنامه[6]، دنباله‌ای از دستورات است. هر دستور دارای یک ترتیب منطقی داخل برنامه است که به آن آدرس منطقی[7] می‌گویند. وقتی برنامه داخل حافظهٔ اصلی است، هر دستور یک آدرس فیزیکی[8] دارد. وقتی یک ماشین فن‌نیومن برنامه‌ای را اجرا می‌کند، در هر لحظه یکی از دستورات را به همان ترتیب نوشته شده اجرا می‌کند مگر اینکه یک دستور به کامپیوتر بگوید که دستوری را خارج از ترتیب اجرا کند (مثل دستور پرش[9]).

برای آنکه اجرای دستورات را دنبال کنیم، ماشینهای فن‌نیومن دارای یک PC می‌باشند، PC به دستور بعدی که باید اجرا شود، اشاره می‌کند.

در طی عملیات عادی، واحد کنترل دو عمل اساسی را دنبال هم انجام می‌دهد: واکشی[10] دستور و اجرای[11] دستور. این دنباله را سیکل[12] ماشین فن‌نیومن می‌گویند. در طی واکشی دستور، واحد کنترل، دستور بعدی که آدرس آن در PC است را از حافظهٔ اصلی واکشی کرده و PC را یکی افزایش می‌دهد. پس از واکشی دستور، PC به دستور بعدی اشاره می‌کند.[13] سپس واحد کنترل دستور را اجرا می‌کند. در طی اجرا، ابتدا پردازنده دستور را ترجمه[14] می‌کند و مشخص می‌کند چه عملی باید صورت گیرد و سپس عملیات را انجام می‌دهد. در نهایت، وقتی پردازنده اجرای دستور جاری را به پایان برد سیکل واکشی - اجرا را برای دستور بعد اجرا می‌کند.

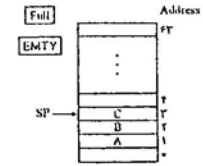
برای افزایش سرعت اجرا، معمولاً معماری فن‌نیومن را با پردازنده‌های پایپلاین[15] پیاده‌سازی می‌کنند. همچنین برای افزایش سرعت می‌توان چندین واحد محاسبه و یا چندین واحد عملیاتی داشت.

ماشین‌های غیر فن‌نیومن

تمام ماشین‌ها، فن‌نیومن نیستند. آقای Flynn بر اساس خواصی مثل تعداد پردازنده، تعداد برنامه‌هایی که می‌توانند اجرا کنند، و ساختار حافظه، معماری‌ها را به چند دسته تقسیم کرده است:

- یک دستور، یک داده[16] (SISD): معماری فن‌نیومن متعلق به این دسته است. کامپیوترهای SISD شامل یک CPU هستند که یک دستور را در یک زمان می‌توانند اجرا کنند و یک داده را در یک زمان می‌توانند از حافظه واکشی کنند یا ذخیره کنند.

1- Operand
2- Instruction Set
3- Operational Code
4- Set
5- Flags
6- Program
7- Logical Address
8- Physical Address
9- Branch
10- Fetch
11- Execute
12- Cycle
۱۳- در عمل ممکن است دستور بیش از یک واحد حافظه را اشغال کرده باشد که نیاز به چند بار افزایش PC است که به دستور بعدی اشاره کند
14- Decode
15- Pipeline
16- Single Instruction Single Data



شکل (۳-۳)

در این شکل، SP همواره آدرس کلمه بالای پشته را در خود دارد. طبق شکل C روی پشته قرار دارد. بنابراین SP=۳ می‌باشد. برای POP کردن، کلمه‌ای که در آدرس ۳ قرار دارد را باید برداشت و SP را یکی کم کرد. برای PUSH کردن ابتدا باید SP را با یک جمع کرد و سپس کلمه را داخل پشته وارد کرد. در این شکل ثبات SP، شش بیتی است.

دو پرچم، برای چک پر یا خالی بودن پشته استفاده می‌شود: EMTY و FULL. اگر پشته خالی شود EMTY ← ۱ و اگر پر شود FULL ← ۱. هر گاه SP شامل آدرس ۶۳ باشد، جمع آن با یک، SP را صفر می‌کند.

عملیات PUSH و POP با توجه به شکل به این صورت است:

PUSH:

```
if (FULL=۰) then
SP ← SP+۱                    (افزایش SP)
M[SP] ← DR                   (نوشتن کلمه در بالای پشته)
IF (SP=۰) then (FULL ← ۱)    (چک پر بودن پشته)
EMTY ← ۰                     (مشخص کردن اینکه پشته خالی نیست)
```

POP:

```
if (EMTY=۰) then
DR ← M[SP]                   (خواندن کلمه از بالای پشته)
SP ← SP-۱                    (کاهش SP)
IF(SP=۰) then EMTY ← ۱       (چک خالی بودن پشته)
FULL ← ۰                     (مشخص کردن اینکه پشته پر نیست)
```

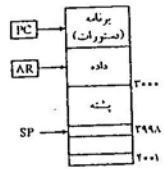
توجه: SP در هر دو حالت پر یا خالی بودن پشته به خانه صفر پشته اشاره می‌کند، ولی بیتهای FULL و EMTY تمایز بین این دو حالت را نشان می‌دهند.

پشته حافظه‌ای

پشته را می‌توان داخل حافظه اصلی نیز در نظر گرفت. کافیست قسمتی از حافظه اصلی را برای عملیات پشته در نظر بگیریم. شکل (۳-۴) قسمتی از حافظه اصلی کامپیوتر را نشان می‌دهد که به سه بخش تقسیم شده است: برنامه، داده و پشته. SP به کلمه بالای پشته اشاره می‌کند. همانطور که می‌دانیم ثبات‌های AR و PC نیز آدرس حافظه را در اختیار می‌گذارند. PC در هنگام عملیات واکشی استفاده می‌شود و به قسمت دستورات اشاره می‌کند، AR در فاز اجرا برای خواندن عملوند مورد نیاز است؛ و SP به منظور PUSH یا POP کردن کلمات استفاده می‌شود.

طبق شکل (۳-۲) اولین کلمه پشته دارای آدرس ۲۰۰۱ می‌باشد و پشته با کاهش آدرس، رشد می‌کند. بنابراین اولین کلمه در آدرس ۲۰۰۰ ذخیره شده و آخرین کلمه در آدرس ۲۰۰۰. می‌توان برای چک پر یا خالی بودن پشته از دو ثبات استفاده کرد که یکی آدرس پایین پشته و دیگری آدرس بالای پشته را در خود دارد. بعد از هر PUSH، SP با ثبات بالای پشته مقایسه می‌شود و پس از POP با ثبات پایین پشته. عملیات PUSH و POP مطابق شکل زیر است:

```
SP ← SP-۱          PUSH
M[SP] ← DR
DR ← M[SP]         POP
SP ← SP+۱
```



شکل (۳-۴)

اینکه ابتدا SP را با یک جمع یا از یک کم کنیم و یا ابتدا عمل PUSH یا POP را انجام دهیم، به ساختار پشته بستگی دارد. به عنوان مثال اگر آدرسها در شکل (۳-۴) به سمت بالا افزایش پیدا کنند، برای PUSH، SP را باید با یک جمع کرد و برای POP، SP را از یک کم کرد.

پشته کاربردهای مختلفی دارد. مثلاً در برنامه‌ها، هنگام صدا کردن یک زیرروال می‌توان آدرس بازگشت را در پشته ذخیره کرد، یکی دیگر از کاربردهای پشته محاسبه عبارات ریاضی است.

کاربرد پشته در محاسبه عبارات ریاضی

به منظور استفاده از پشته برای محاسبه عبارات ریاضی، باید عبارت را به فرم پسوندی[1] یا فرم معکوس لهستانی[2](RPN) نوشت. ما همواره با فرم میانوندی[3] عبارات سروکار داریم. به عنوان مثال عبارت A*B-C*D یک فرم میانوندی است.

مثال: عبارت A+B به فرمهای متفاوت در زیر نوشته شده است:

A+B	فرم میانوندی
+AB	فرم پیشوندی[4] یا لهستانی
AB+	فرم پسوندی یا معکوس لهستانی

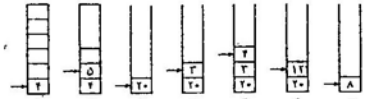
بنابراین فرم RPN عبارت A*B-C*D به شکل AB*CD*- می‌باشد.

مثال: محاسبهٔ عبارت ۲*۳-۵*۴.

ابتدا عبارت را به فرم RPN تبدیل می‌کنیم: ۲۳*۵-۴*. سپس از سمت چپ عبارت را بررسی می‌کنیم، با رسیدن به هر عملوند[5] آن را به پشته PUSH می‌کنیم و با رسیدن به هر عملگر[6]، دو عملوند بالای پشته را، برداشته و با هم عمل می‌کنیم و حاصل را به پشته برمی‌گردانیم. این روند در شکل ۳-۵ توضیح داده شده است. ابتدا ۲ و سپس ۵ به پشته PUSH می‌شوند با رسیدن به *، ۵ و ۲ از پشته حذف شده است و ۲۰ در پشته قرار می‌گیرد سپس ۳ و پس از آن ۲ به پشته

1. Postfix
2. Reverse Polish Notation
3. Infix
4. Prefix
5. Operand
6. Operator

PUSH می‌شوند و با رسیدن به ۴، ۴ و ۳ حذف شده و ۱۲ روی پشته قرار می‌گیرد و سپس با رسیدن به ـ ۲۰ و ۱۲ از پشته حذف شده و ۱۲ـ۲۰ یعنی ۸ در پشته قرار می‌گیرد که حاصل نهایی است.



شکل (۵ـ۳) عملیات پشته برای محاسبه عبارت ۴*۳+۵ـ۴

* **نکات:** ۱ـ به تعداد عملوند، PUSH کرده‌ایم یعنی ۳ عمل PUSH انجام داده‌ایم. ۲ـ تعداد کلمات مورد نیاز در پشته مطابق شکل (۵ـ۳)، ۳ کلمه است.

فرمت‌های دستورالعمل

فرمت دستورات در دو فصل قبل برای دو ماشین بررسی شد. معمولاً هر فرمت دستور شامل فیلدهای زیر است:

۱ـ یک فیلد کد عملیاتی (opcode) که مشخص‌کنندهٔ نوع عملی است که انجام می‌شود.

۲ـ یک فیلد آدرس که یک آدرس حافظه یا یک ثبات پردازنده را مشخص می‌کند.

۳ـ یک فیلد مُد آدرس‌دهی را مشخص می‌کند؛ یعنی روشی که عملوند پیدا می‌شود را تعیین می‌کند.

سایر فیلدها نیز می‌توانند استفاده شوند. مثلاً فیلدی که تعداد شیفت‌ها را در یک دستور شیفت مشخص می‌کند.

اغلب کامپیوترها دارای سه نوع ساختار کلی هستند:

۱ـ ساختار مبتنی بر یک آکومولاتور

۲ـ ساختار چند ثباتی (مبتنی بر ثباتهای عمومی)

۳ـ ساختار پشته‌ای

کامپیوتر معرفی‌شده در فصل ۱، مبتنی بر AC است. فرمت دستور در این ماشین فقط دارای یک فیلد آدرس است و دستوری مثل ADD X، محتویات حافظه X را با AC جمع می‌کند: $AC \longleftarrow AC+M[X]$

مثالی از ساختار کامپیوترهای چند ثباتی در شکل (۲ـ۳) آمده است. در این نوع کامپیوتر احتیاج به ۳ فیلد آدرس دارد. بنابراین دستور ADD R_1, R_2, R_3، مثلاً R_2 را با R_3 جمع و حاصل را به R_1 منتقل می‌کند $R_1 \longleftarrow R_2+R_3$

تعداد فیلدهای آدرس در دستور می‌تواند ۲ تا باشد مثلاً R_2 و ADD R_1، عمل $R_1 \longleftarrow R_1+R_2$ را انجام می‌دهد. در ماشین‌هایی که مبتنی بر ثباتها هستند می‌توان عملوندی از حافظه را با ثباتی جمع کرد و حاصل را به ثبات منتقل کرد: مثلاً ADD R_1,X، عمل $R_1 \longleftarrow R_1+M[X]$ را انجام می‌دهد.

ماشینهای پشته‌ای: از دستورات PUSH و POP استفاده می‌کنند که این دستورات احتیاج به یک فیلد آدرس دارند. بنابراین دستور PUSH X کلمه‌ای که در آدرس X قرار دارد را به بالای پشته منتقل می‌کند. در این ماشین، دستوراتی که روی عملوندها، عمل می‌کنند نیاز به فیلد آدرس ندارند و مثلاً دستور ADD در این نوع ماشین دو عملوند بالای پشته را برداشته و جمع کرده و حاصل را به پشته بازمی‌گرداند.

اکثر کامپیوترها مشخصات هر سه نوع کامپیوتری را که معرفی شد، دارند.

مثال: می‌خواهیم عبارت $M[X]=(M[A]+M[B])*(M[C]+M[D])$ را با دستورات صفر، یک، دو و سه آدرس محاسبه کنیم.

۱ـ دستورات سه آدرسی: در این نوع، هر فیلد هم می‌تواند یک ثبات را مشخص کند و هم یک آدرس حافظه را:

```
ADD   R1,A,B ; R1 ←— M[A]+M[B]
ADD   R2,C,D ; R2 ←— M[C]+M[D]
MUL   X,R1,R2 ; M[X] ←— R1*R2
```

پس از هر دستور، توضیحات آن دستور آمده است مزیت این نوع فرمت، کوتاهی برنامه‌ها است. عیب آن طولانی بودن دستور می‌باشد.

۲ـ دستورات دو آدرسی: این دستورات معمولترین نوع در کامپیوترها هستند:

```
MOV   R1,A ; R1 ←— M[A]
ADD   R1,B ; R1 ←— R1+M[B]
MOV   R2,C ; R2 ←— M[C]
ADD   R2,D ; R2 ←— R2+M[D]
MUL   R1,R2 ; R1 ←— R1*R2
MOV   X,R1 ; M[X] ←— R1
```

۳ـ دستورات تک آدرسی: دستورات تک آدرسی از ثبات (AC) استفاده می‌کنند. دستور LOAD برای منتقل کردن مقداری از حافظه به یکی از ثباتها (در اینجا AC) و دستور STORE برای ذخیره یکی از ثباتها (در اینجا AC) در حافظه استفاده می‌شوند.

```
LOAD  A  ; AC ←— M[A]
ADD   B  ; AC ←— AC+M[B]
STORE T  ; M[T] ←— AC
LOAD  C  ; AC ←— M[C]
ADD   D  ; AC ←— AC+M[D]
MUL   T  ; AC ←— AC*M[T]
STORE X  ; M[X] ←— AC
```

۴ـ دستورات بدون فیلد آدرس: در ماشینهای پشته‌ای فقط PUSH و POP دارای یک فیلد آدرس هستند و سایر دستورات فیلد آدرس ندارند.

```
PUSH A ; TOS ←— A   (TOS: TOP Of Stack)
PUSH B ; TOS ←— B
ADD    ; TOS ←— (A+B)
PUSH C ; TOS ←— C
```

منظور از TOS بالای پشته است.

برای محاسبه عبارت، ابتدا باید آنرا به فرم RPN تبدیل کرد.

```
PUSH  D  ;  TOS ←— D
ADD      ;  TOS ←— (C+D)
MUL      ;  TOS ←— (C+D)*(A+B)
POP   X  ;  M[X] ←— TOS
```

دستورات RISC [1]

ماشینهای RISC بعداً توضیح داده خواهند شد. دستوراتی که در این نوع ماشین با حافظه سروکار دارند، محدود به Load و Store هستند. یعنی این دو دستور می‌توانند ارتباط بین ثباتهای پردازنده و حافظه را فراهم کنند. سایر دستورات در این ماشینها فقط روی ثباتها عمل می‌کنند و هیچ مراجعه‌ای به حافظه ندارند:

```
LOAD    R1,A   ;  R1 ←— M[A]
LOAD    R2,B   ;  R2 ←— M[B]
LOAD    R3,C   ;  R3 ←— M[C]
LOAD    R4,D   ;  R4 ←— M[D]
ADD     R1,R2  ;  R1 ←— R1+R2
ADD     R3,R2  ;  R2 ←— R3+R2
MUL     R1,R3  ;  R1 ←— R1*R3
STORE   X,R1   ;  M[X] ←— R1
```

مدهای آدرس‌دهی

مد آدرس‌دهی مشخص می‌کند که عملوند چگونه انتخاب شود. هدف از بکارگیری تکنیکهای مختلف مدهای آدرس‌دهی یکی از موارد زیر است:

۱ـ برای برنامه‌نویس امکاناتی از قبیل اشاره‌گر به حافظه، شمارنده برای حلقه‌ها و... فراهم کند.

۲ـ کاهش تعداد بیت در فیلد آدرس در دستور است.

برای داشتن چند مد آدرس‌دهی، دستور باید دارای فیلد مد آدرس‌دهی مطابق شکل (۶ـ۳) باشد. مثلاً اگر ماشینی دارای ۱۰ مد آدرس‌دهی مختلف است، فیلد مُد باید حداقل ۴ بیتی باشد.

Opcode	Mode	Address

شکل (۶ـ۳) فرمت دستور با فیلد مُد

اگر چه اغلب مدهای آدرس‌دهی، فیلد آدرس دستور را دستکاری می‌کنند، دو مُد وجود دارد که هیچ نیازی به فیلد آدرس ندارد. این دو نوع عبارتند از: مدهای ضمنی [2] و بلافصل [3].

مد ضمنی Implied: در این مد عملوند به طور ضمنی در خود دستور نهفته است. مثلاً دستور CMA (complement AC) یک

1- Reduced Instruction Set Computer 2- Implied 3- Immediate

دستور در مُد ضمنی است، چون عملوند در ثبات AC می‌باشد و به طور ضمنی در دستور مشخص شده است. تمام دستورات ثباتی که از AC استفاده می‌کنند، در مُد ضمنی هستند. همچنین دستورات بدون فیلد آدرس در ماشین پشته‌ای، دستوراتی در مُد ضمنی هستند.

مد بلافصل Immediate: در این مُد، عملوند در خود دستور نهفته است. به عبارت دیگر، یک دستور در مُد بلافصل دارای فیلد عملوند است نه فیلد آدرس.

مد ثباتی Register mode: در این مُد، عملوند در حافظه نیست بلکه در یکی از ثباتهای پردازنده است.

مد ثباتی غیر مستقیم Register Indirect mode: در این مُد، دستور، ثباتی از CPU را مشخص می‌کند که آدرس عملوند و نه خود عملوند، در آن است. یعنی در این مُد، یک ثبات شامل آدرس عملوند در حافظه است. مزیت این مُد این است که فیلد آدرس بیتهای کمتری نیاز دارد چرا که باید به یکی از ثباتهای CPU اشاره کند و نه به یک آدرس حافظه.

مد افزاینده یا کاهنده خودکار Autoincrement or Autodecrement: این مد شبیه ثباتی غیر مستقیم است، با این تفاوت که ثبات بعد یا قبل از استفاده، یک واحد افزایش یا کاهش می‌یابد.

مد آدرس‌دهی مستقیم یا مطلق direct-absolute: در این مد، آدرس موثر برابر فیلد آدرس دستور است، یعنی عملوند در حافظه است و آدرس آن در فیلد آدرس دستورالعمل.

مد آدرسی‌دهی غیرمستقیم Indirect Address Mode: در این مُد، فیلد آدرس دستور، آدرس موثر نیست، بلکه آدرس آدرس موثر است. یعنی فیلد آدرس، مکانی از حافظه را نشان می‌دهد که آدرس عملوند (نه خود عملوند) آنجاست.

در بعضی از مدهای آدرس‌دهی، فیلد آدرس دستور به یکی از ثباتهای پردازنده افزوده می‌شود تا آدرس موثر بدست آید. یعنی آدرس موثر از رابطهٔ زیر محاسبه می‌شود:

محتویات یک ثبات + فیلد آدرس دستور = آدرس موثر

ثباتی که استفاده می‌شود می‌تواند PC باشد که در این صورت به این مُد، مُد آدرس‌دهی نسبی [1] گویند. اگر یک ثبات ایندکس باشد، مد را آدرس‌دهی ایندکس [2] گویند. و اگر ثبات یک ثبات پایه باشد، مد را آدرس‌دهی پایه [3] گویند.

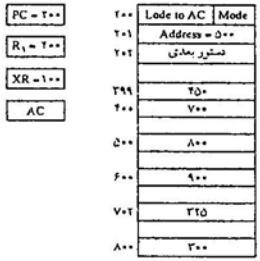
مد آدرس‌دهی نسبی Relative: در این مُد محتویات PC به فیلد آدرس افزوده می‌شود تا آدرس موثر محاسبه شود. قسمت آدرس دستور می‌تواند مثبت یا منفی باشد، که وقتی به PC افزوده شود آدرس موثری پیدا می‌شود که موقعیت آن نسبت به دستور بعدی می‌باشد. فرض کنید PC=۸۲۵ و قسمت آدرس دستور شامل عدد ۲۲ باشد. دستور از آدرس ۸۲۵ خوانده شده و PC یکی افزایش می‌یابد و آدرس موثر برای مد نسبی عبارت خواهد بود از: ۸۵۰=۲۲+۸۲۶. که این آدرس ۲۲ خانهٔ حافظه جلوتر از دستور بعدی است. از این مُد معمولاً در آدرس پرش استفاده می‌شود.

مد آدرس‌دهی ایندکس یا شاخص‌دار Indexed: در این مد محتویات یک ثبات ایندکس به قسمت آدرس دستور افزوده می‌شود تا آدرس موثر بدست آید. ثبات ایندکس، یکی از ثباتهای CPU است. فیلد آدرس دستور، مشخص‌کننده آدرس شروع یک آرایه در حافظه است و با افزایش ثبات ایندکس عناصر آرایه قابل دسترسی هستند. یعنی در این مد فیلد آدرس دستور ثابت است و ثبات ایندکس عوض می‌شود تا عناصر آرایه را یکی‌یکی، آدرس‌دهی کند.

1- Relative 2- Indexed mode 3- Base Addressing mode

مد آدرس‌دهی ثبات پایه Base Register: در این مد، محتویات یک ثبات پایه به قسمت آدرس دستور افزوده می‌شود تا آدرس موثر پیدا شود. این مُد شبیه مُد شاخص‌دار است، با این تفاوت که نام ثبات پایه است. در این مُد ثبات پایه شامل یک آدرس پایه است و فیلد آدرس دستور عوض می‌شود و فاصله‌ای نسبت به محتویات ثبات پایه بوجود می‌آورد. این مُد به این منظور استفاده می‌شود که اگر برنامه از یک بخش حافظه به بخش دیگری منتقل شود، فقط کافیست محتویات ثبات پایه را عوض کرد تا به آدرس شروع آن بخش اشاره کند، و نیازی به تغییر فیلد آدرس دستور نیست.

مثال: در شکل (۷ـ۳)، در آدرسهای ۲۰۰ و ۲۰۱ دستور "Load to AC" قرار دارد که فیلد آدرس آن شامل عدد ۵۰۰ است. فیلد دوم دستور که در کلمه اول دستور واقع است، مُد آدرس‌دهی است. می‌خواهیم با توجه به مقادیر مشخص‌شده در شکل، اثر مُدهای مختلف را بررسی کنیم. جدول (۱ـ۳) اثر مدهای مختلف را نشان می‌دهد.



شکل (۷ـ۳) مثال عددیِ مدهای آدرس‌دهی

جدول (۱ـ۳)

مد آدرس‌دهی	آدرس موثر	محتویات AC	محتویات R_1	محتویات PC
مستقیم	۵۰۰	۸۰۰	۲۰۰	۲۰۲
بلافصل	۲۰۱	۵۰۰	۲۰۰	۲۰۲
غیر مستقیم	۸۰۰	۳۰۰	۲۰۰	۲۰۲
نسبی	۷۰۲	۳۲۵	۲۰۰	۲۰۲
شاخص‌دار	۶۰۰	۹۰۰	۲۰۰	۲۰۲
ثباتی	-	۲۰۰	۲۰۰	۲۰۲
ثباتی غیر مستقیم	۲۰۰	۷۰۰	۲۰۰	۲۰۲
پس‌افزاینده	۲۰۰	۷۰۰	۲۰۱	۲۰۲
پیش‌کاهنده	۳۹۹	۴۵۰	۳۹۹	۲۰۲

دقت شود PC، ۲۰۲ می‌باشد زیرا دستور بعدی در آدرس ۲۰۲ است. در مد پس‌افزاینده ابتدا آدرس موثر پیدا می‌شود، سپس R_1 با یک جمع می‌شود. در مد پیش‌کاهنده برعکس یعنی R_1 یکی کم شده است و سپس آدرس موثر پیدا شده است.

دستورات کامپیوتر

اغلب کامپیوترها دارای دستوراتی هستند که می‌توان آنها را به این شکل دسته‌بندی کرد:

۱ـ دستورات انتقال داده، ۲ـ دستورات دستکاری داده‌ها، ۳ـ دستورات کنترل برنامه، *۴ـ دستورات خاص پردازنده

جدول (۲ـ۳) لیستی از ۸ دستور انتقال داده را که در اغلب کامپیوترها استفاده می‌شود، نشان می‌دهد.

دستورات load و store برای انتقال داده به/از حافظه از/به ثباتها استفاده می‌شوند.

دستور move معمولاً برای انتقال داده داخل ثباتها است. این دستور همچنین برای انتقال داده بین حافظه و ثباتها نیز استفاده می‌شود. دستور exchange محتویات دو ثبات را با هم عوض می‌کند. دستورات Input و Output برای انتقال داده از ورودی/خروجی به ثباتها می‌باشند و دستورات PUSH و POP برای انتقال از ثباتها به حافظه پشته می‌باشند.

جدول (۲ـ۳)

نام دستور	اختصار
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

جدول (۳ـ۳) دستور Load را همراه با مدهای متفاوت نشان می‌دهد.

دقت شود که این قراردادها قابل‌تغییر هستند و هر کامپیوتر از نمادهای متفاوت برای مدهای مختلف استفاده می‌کند.

جدول (۳ـ۳) ۸ مد آدرس‌دهی برای دستور Load

مد	نماد اسمبلی	RTL
مستقیم	LD ADR	$AC \longleftarrow M[ADR]$
غیر مستقیم	LD @ADR	$AC \longleftarrow M[M(ADR)]$
نسبی	LD $ADR	$AC \longleftarrow M[PC+ADR]$
بلافصل	LD #NBR	$AC \longleftarrow NBR$
شاخص‌دار	LD ADR(X)	$AC \longleftarrow M[ADR+XR]$
ثباتی	LD R_1	$AC \longleftarrow R_1$
ثباتی غیرمستقیم	LD (R_1)	$AC \longleftarrow M[R_1]$
پس‌فزاینده	LD (R_1)+	$AC \longleftarrow M[R_1],\ R_1 \longleftarrow R_1+1$
پیش‌کاهنده	LD -(R_1)	$R_1 \longleftarrow R_1-1,\ AC \longleftarrow M[R_1]$

* - این نوع دستورات، معمولاً در حالات کاری خاص مورد استفاده هستند.

دستورات دستکاری داده‌ها

این دستورات معمولاً به سه دسته تقسیم می‌شوند:

۱ـ دستورات حسابی

۲ـ دستورات منطقی و دستکاری بیت

۳ـ دستورات شیفت

دستورات حسابی قبلاً معرفی شده‌اند. جدول (۳ـ۳) دستورات معمول حسابی را نشان می‌دهد.

جدول (۴ـ۳) دستورات حسابی معمول

نام	نماد
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate	NEG

دستورات منطقی، عملیات دودویی را روی بیت‌های ثبات‌ها انجام می‌دهند. دستورات معمول این گروه در جدول (۵ـ۳) مشاهده می‌شود. مثلاً دستور Clear، بیتهای عملوند خود را تماماً صفر می‌کند.

جدول (۵ـ۳) دستورات منطقی و دستکاری بیت

نام	نماد
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

دستورات شیفت نیز که قبلاً معرفی شده‌اند در جدول ۶ـ۳ آمده‌اند.

جدول (۶ـ۳) دستورات شیفت معمول

نام	نماد
Logical shift right	SHR
Logical shift left	SHL
Arithmentic shift left	SHLA
Arithmetic shift right	SHRA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

* در جدول (۳ـ۳) دستور ADDC، دو عملوند را با احتساب رقم نقلی گذشته جمع می‌کند. و دستور SUBB دو عملوند را از هم کم می‌کند و رقم قرضی را نیز که از مرحله قبل است، کم می‌کند.

دستورات کنترل برنامه

دستورات در حافظه، پشت سر هم قرار گرفته‌اند و به ترتیب اجرا می‌شوند مگر اینکه بخواهیم این ترتیب به هم بخورد. دستوراتی که در جدول ۷ـ۳ نشان داده شده‌اند، دستورات کنترل برنامه بوده و به منظور خارج از ترتیب اجراشدن دستورات مورد استفاده قرار می‌گیرند.

جدول (۷ـ۳) دستورات کنترل برنامه

نام	نماد
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

دستورات Branch و Jump معمولاً معادلند. این دستورات یا شرطی هستند یا غیرشرطی. پرش شرطی بر اساس اینکه مثلاً حاصل تفریق صفر شده است یا اینکه A>B یا... پرش می‌کند و اگر شرط برقرار نباشد دستور بعدی در ترتیب برنامه اجرا می‌شود. دستور Skip دستوری بدون فیلد آدرس است و اگر شرط برقرار باشد از دستور بعدی می‌گذرد.

دستورات Call و Return نیز برای زیرروال‌ها مورد استفاده قرار می‌گیرند. دستور Compare، دو عملوند خود را از

هم کم کرده ولی نتیجه را ذخیره نمی‌کند فقط یک سری بیت‌های وضعیت را (Z, S, V, C) دستکاری می‌کند و دستورات پرش بعدی با توجه به این بیتهای وضعیت پرش می‌کنند یا نمی‌کنند. دستور Test نیز دو عملوند خود را با هم AND می‌کند و بیتهای وضعیت طبق آن تغییر می‌کنند، ولی نتیجه در جایی ذخیره نمی‌شود.

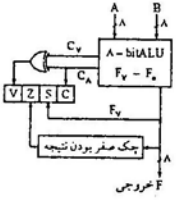
شرایط بیتهای وضعیت Status Bit Condition

در کامپیوترها ثباتی همراه با ALU در نظر گرفته می‌شود که در آن بیتهای وضعیت ذخیره می‌شوند. بیتهای وضعیت را بیتهای شرط[1] یا بیتهای پرچم[2] نیز می‌گویند.

شکل (۸ـ۳) بلوک دیاگرام یک ALU هشت بیتی را همراه با ثبات وضعیت ۴ بیتی نشان می‌دهد. بیتهای وضعیت با V,Z,S,C نشان داده شده‌اند:

- بیت C (Carry رقم نقلی) ۱ می‌شود اگر رقم نقلی خروجی C_A یک شود، در غیر این صورت صفر است.
- بیت S (Sign علامت) که گاهی با N (negative) نیز نمایش می‌دهند زمانی یک می‌شود که بیت علامت (باارزشترین بیت MSB) یعنی F_V یک باشد.
- بیت Z (Zero) یک می‌شود اگر خروجی برابر صفر باشد.
- بیت V (Overflow سرریز) یک می‌شود اگر XOR دو رقم نقلی آخر برابر ۱ باشد. در واقع یک شدن V نشان‌دهنده خطابودن حاصل است.

با بررسی بیتهای وضعیت می‌توان رابطهٔ بین عملوندها را فهمید و بر اساس آن تصمیماتی مثل تصمیم برای پرش، گرفت.



شکل (۸ـ۳)

صدا کردن و بازگشت از زیرروال

یک زیرروال یک دنباله از دستورات است که یک عملی را انجام می‌دهند. در طی اجرای یک برنامه، یک زیرروال ممکن است صدازده شود تا عمل مربوط به خود را انجام دهد. هر زمان که یک زیرروال صدا زده می‌شود، یک پرش به ابتدای زیرروال صورت می‌گیرد. پس از اتمام اجرای زیرروال، یک پرش به برنامه اصلی صورت می‌گیرد.

دستوری که کنترل را در برنامه به یک زیرروال می‌برد، دارای نامهای متفاوتی است. معمولترین این نامها، عبارتند از

1- Condition-Code 2- Flag

«صداکردن زیرروال»، «پرش به زیرروال»، «انشعاب به زیرروال»، یا «انشعاب و ذخیره آدرس». پرش به زیرروال، دو عمل را انجام می‌دهد: ۱ـ آدرس دستور بعدی (محتویات PC) در مکانی ذخیره می‌شود۲ـ کنترل به ابتدای زیر روال منتقل می‌شود. آخرین دستور هر زیرروال، که معمولاً بازگشت از زیرروال (Return) نامیده می‌شود، آدرس بازگشت را که ذخیره شده بود به PC منتقل می‌کند.

کامپیوترها برای ذخیره آدرس بازگشت می‌توانند از مکانی از حافظه استفاده کنند یا می‌توانند آدرس بازگشت را در ابتدای زیرروال ذخیره کنند، یا در یک ثبات ذخیره کنند، و یا می‌توانند از پشته استفاده کنند. مزیت استفاده از پشته این است که اگر از داخل زیرروالها نیز به زیر روال دیگری برویم (دنباله‌ای از زیرروال‌ها)، آدرس‌های بازگشت را می‌توان در یک پشته، ذخیره کرد. و هنگام بازگشت از هر زیرروال پشته را POP کرده و آدرس بازگشت را به PC منتقل کنیم. صدا کردن زیرروال را می‌توان با ریزعملیات زیر انجام داد:

SP ⟵ SP-۱ (کاهش اشاره‌گر پشته)

M[SP] ⟵ PC (ذخیره PC در پشته)

PC ⟵ آدرس مؤثر (انتقال کنترل به زیرروال)

اگر زیرروال دیگری توسط زیرروال فعلی صدا زده شود، آدرس بازگشت جدید در پشته Push خواهد شد. دستوری که از زیرروال باز می‌گردد توسط ریزعملیات زیر پیاده سازی می‌شود:

PC ⟵ M[SP] (پشته را POP کرده و محتویات به PC منتقل می‌شود)

SP ⟵ SP+۱ (اشاره‌گر پشته یک واحد اضافه می‌شود)

وقفه

ایده وقفه، برخورد با مسائلی است که دنباله عادی برنامه را بر هم می‌زند. وقفه به انتقال کنترل برنامه از برنامهٔ در حال اجرای جاری به یک برنامهٔ دیگر گفته می‌شود. پس از اجرای برنامه سرویس وقفه، کنترل به برنامهٔ عادی بازمی‌گردد. پروسه وقفه، در اساس، تقریباً مشابه صدا کردن زیرروال است ولی با آن ۳ تفاوت دارد، ۱ـ یک وقفه معمولاً توسط یک سیگنال داخلی یا خارجی رخ می‌دهد نه با اجرای یک دستور (به جز وقفه‌های نرم‌افزاری)؛ ۲ـ آدرس برنامه سرویس‌دهی به وقفه، سخت‌افزاری مشخص می‌شود نه از روی فیلد آدرس دستور؛ و ۳ـ یک پروسه وقفه، معمولاً تمام اطلاعات لازم برای تعیین حالت CPU (مثل بیتهای وضعیت) را ذخیره می‌کند، ولی صدا کردن زیرروال فقط PC را ذخیره می‌کند.

حالت CPU توسط بندهای زیر تعیین می‌شود:

۱ـ محتویات PC (شمارنده برنامه)

۲ـ محتویات ثبات‌های پردازنده

۳ـ محتویات بعضی شرایط و بیت‌های وضعیت (PSW[1]) گفته می‌شود).

در واقع موارد ۱، ۲ و ۳ در هنگام بروز وقفه ذخیره می‌شوند.

1- Program status word

اگر وقفه‌ای در حین اجرای دستوری رخ دهد، اجرای دستور خاتمه می‌یابد و پس از اجرا، پردازنده چک می‌کند آیا وقفه رخ داده است یا خیر، که در صورت وقوع وقفه، روتین پاسخ‌دهی به وقفه اجرا می‌شود. ممکن است برای وقفه‌های مختلف روتین‌های مختلفی داشته باشیم که در این صورت در صورت بروز هر نوع وقفه، روتین مربوط به همان وقفه اجرا می‌شود.

انواع وقفه

سه نوع اصلی وقفه وجود دارد:

۱ـ وقفه‌های خارجی

۲ـ وقفه‌های داخلی

۳ـ وقفه‌های نرم‌افزاری

وقفه‌های خارجی از وسایل ورودی / خروجی، از یک وسیله زمانبندی، یا از هر منبع خارجی دیگری می‌آیند. مثالهایی از وقوع این نوع وقفه عبارت است از: وسیله I/O درخواست انتقال داده کند، وسیله I/O انتقال داده را خاتمه داده باشد، اشکال در سیستم برق. مثلاً اگر یک برنامه در یک حلقه بی‌نهایت قرار گیرد، ممکن است وقفه timeout توسط timer رخ دهد.

وقفه‌های داخلی به علت استفاده ناصحیح از دستورات یا داده‌ها رخ می‌دهد. به این نوع وقفه trap نیز گفته می‌شود. وقفه‌های این نوع عبارتند از: تقسیم بر صفر، یک کد عملیاتی نامعتبر، سرریز پشته و... تفاوت وقفه داخلی با خارجی یکی منبع بروز آنهاست و دیگر اینکه وقفه‌های داخلی با برنامه سنکرون هستند یعنی هر بار برنامه اجرا شود در یک مکان و زمان ثابت رخ می‌دهند ولی وقفه‌های خارجی آسنکرون هستند یعنی در اجراهای متفاوت برنامه ممکن است در مکان‌های مختلف رخ دهند یا اصلاً رخ ندهند.

به هر حال وقفه‌های داخلی و خارجی توسط سیگنالهایی که در سخت‌افزار CPU رخ می‌دهند راه‌اندازی می‌شوند. وقفه نرم‌افزاری توسط اجرای یک دستور ایجاد می‌شود. یک وقفه نرم‌افزاری یک صدا کردن بخصوص زیرروال است که شبیه یک وقفه عمل می‌کند تا یک صداکردن زیرروال.

کامپیوترهای RISC (Reduced Instruction Set Computers) و CISC (Complex Instruction Set Computers):

کامپیوترهای CISC کامپیوترهای با تعداد دستورات زیاد هستند. مشخصه‌های اصلی این کامپیوترها عبارت است از:

۱ـ دستورات زیاد-معمولاً بین ۱۰۰ تا ۲۵۰ دستور

۲ـ دارای بعضی دستورات هستند که کاربردهای خاص دارند و به ندرت استفاده می‌شوند.

۳ـ مدهای آدرس‌دهی زیاد دارند. بین ۵ تا ۲۰ مد مختلف

۴ـ دستورات با طول متفاوت دارند.

۵ـ دارای دستوراتی هستند که می‌توانند با عملوندهای حافظه‌ای سر و کار داشته باشند.

ایده معماری RISC، تلاشی برای کاهش زمان اجرای دستورات با ساده‌سازی مجموعه دستورات بود. مشخصه‌های اصلی کامپیوترهای RISC:

۱ـ دستورات نسبتاً کم

۲ـ مدهای آدرس‌دهی نسبتاً کم

۳ـ دسترسی به حافظه در این کامپیوترها فقط منحصر به دستورات Load و Store است.

۴ـ تمام عملیات داخل ثباتهای پردازنده انجام می‌شود.

۵ـ دستورات طول ثابت دارند که با راحتی ترجمه (decode) می‌شوند.

۶ـ اجرای دستورات معمولاً در یک سیکل صورت می‌گیرد.

۷ـ واحد کنترل سیم‌بندی شده است.

یکی از مشخصه‌های RISC اجرای دستورات در یک کلاک است. این مسئله با هم‌پوشانی سیکل‌های Fetch و Decode امکان‌پذیر است. برای این منظور از پایپلاین استفاده می‌شود. البته پایپلاین در کامپیوترهای غیر RISC هم وجود دارد. دیگر مشخصه‌هایی که معمولاً به کامپیوترهای RISC نسبت می‌دهند:

۱ـ تعداد نسبتاً زیاد ثبات پردازنده

۲ـ استفاده از پنجره‌های ثبات هم‌پوشان برای تسریع صداکردن و بازگشت از زیرروال

۳ـ استفاده مؤثر از پایپلاین دستورالعمل.

اگر تنها کامپیوتر برای دستور یک آدرس استفاده کند، تعداد کل دستورالعمل‌ها چقدر خواهد بود؟

2^4 دستورالعمل با آدرس دو آدرس

2^4 دستورالعمل با آدرس یک آدرس

$2^6(2^4-1)$ دستورالعمل بدون آدرس

$= 8 + 4 + 6 = 18$ دستورالعمل

۱۶ـ گزینهٔ (د) صحیح است.

۱۷ـ گزینهٔ (ب) صحیح است. گزینه‌های دیگر تعاریف درست کامپیوترهای CISC و RISC است.

۱۸ـ گزینهٔ (الف) صحیح است. سؤال مبهم است. طبق تعریف آدرس مؤثر، آدرس عملوند است. (یعنی گزینهٔ الف) و این هیچ ارتباطی به نوع آدرس‌دهی ندارد. از طرفی در آدرس‌دهی غیر مستقیم، عملوند دستور، آدرس آدرس موثر است یعنی گزینه ب.

۱۹ـ گزینهٔ (ب) صحیح است. ثبات دستور برای نگهداری دستور استفاده می‌شود که کد عملیاتی نیز قسمتی از دستور است.

۲۰ـ گزینهٔ (الف) صحیح است. PC آدرس دستور بعدی را در خود دارد.

۲۱ـ گزینهٔ (الف) صحیح است. R_2 دارای آدرسی است که محتویات آن آدرس به محلی که R_1 به آن اشاره می‌کند یعنی آدرس ۸۰۰۰ منتقل می‌شود.

۲۲ـ گزینهٔ (ب) صحیح است. برای سه بایت باید سه بار حافظه را خواند که یک بار آن opcode fetch است.

۲۳ـ گزینهٔ (ب) صحیح است.

۲۴ـ گزینهٔ (د) صحیح است. مطلق همان مستقیم است.

۲۵ـ گزینهٔ (ج) صحیح است.

۲۶ـ گزینهٔ (د) صحیح است. یعنی محتوای R_1 به آدرس ۲۰۱۰ منتقل می‌شود.

۲۷ـ گزینهٔ (ج) صحیح است. گزینهٔ (الف) صحیح نیست چون وقفه‌های نرم‌افزاری از اجرای دستور ناشی می‌شوند.

۲۸ـ گزینهٔ (ج) صحیح است.

Push AC
t_0 : SP ← SP-۱
t_1 : AR ← SP
t_2 : M[AR] ← AC

پس Push احتیاج به ۲ کلاک دارد.

POP AC
t_0 : AR ← SP
t_1 : AC ← M[AR] , SP ← SP+۱

پس POP احتیاج ۲ کلاک دارد. دقت شود در شکل داده شده، باس می‌تواند به AC مستقیماً Load کند.

فصل چهارم

الگوریتم‌های حسابی

ضرب اعداد مثبت

در شکل ۱ـ۲ الگوریتم ضرب معمولی نشان داده شده است. ضرب دو عدد n بیتی احتیاج به ۲n بیت دارد، بنابراین در این مثال ۸ بیت برای حاصل احتیاج است. ضرب دو عدد مثبت دودویی را می‌توان با آرایه منطقی دو بعدی ترکیبی، مطابق شکل ۱ـ۳b انجام داد. جزء اصلی در در سلول یک مدار جمع‌کننده است (با A نشان داده شده است). گیت AND در هر سلول مشخص می‌کند که آیا بیت m_j از Multiplicand «ضرب‌شونده» با بیت حاصلضرب جزئی[1] جمع شود یا خیر (براساس «مقدار بیت q_i از multiplier «ضرب‌کننده»).

هر ردیف i، ۰ ≤ i ≤ ۳، از این آرایه، مقدار ضرب شونده (که به درستی شیفت داده شده است) را با حاصلضرب جزئی PPi جمع می‌کند تا حاصلضرب جزئی خروجی PP(i+۱) تولید شود (اگر q_i=۱). اگر q_i=۰ باشد، PPi بدون تغییر به سمت پایین حرکت می‌کند. PP۰ تماماً ۰ است، و PP۴ حاصلضرب مورد نظر است. مقدار ضرب شونده در هر سطر توسط مسیر قطری، یک واحد به سمت چپ شیفت می‌یابد. ضرب‌کننده شکل (۱ـ۴) به علت تعداد گیتهای زیاد، در کامپیوترها استفاده نمی‌شود.

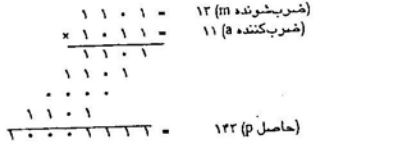
معمولاً ضرب مطابق شکل (۲ـ۴) انجام می‌شود.

ضرب در این مدار توسط n جمع متوالی پیاده‌سازی شده است. ترکیب ثباتهای A و Q مقدار حاصلضرب جزئی PPi را در زمانی که q_i فرمان ADD/NOADD را صادر می‌کند، نگه می‌دارند. این سیگنال (q_i) جمع‌شدن ضرب‌شونده M را به PPi برای تولید PP(i+۱) کنترل می‌کند. n سیکل برای تولید حاصل نیاز است. A در ابتدا صفر است. رقم نقلی خروجی در فلیپ‌فلاپ C ذخیره می‌شود.

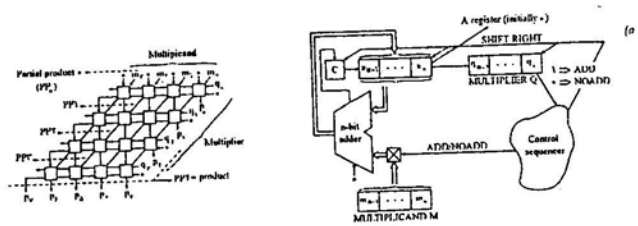
در ابتدا ضرب‌کننده در Q، ضرب‌شونده در M قرار گرفته و A و C صفر می‌شوند. در انتهای هر سیکل C، A و Q به سمت راست یک واحد شیفت می‌یابند. به خاطر این شیفت، بیت ضرب‌کننده q_i در مرتبه پایین (LSB) ثبات Q ظاهر شده تا سیگنال ADD/NOADD در زمان مناسب تولید شود. (در سیکل اول q_0 در سیکل دوم q_1 و...).

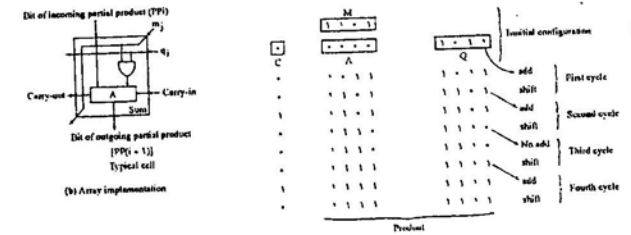
توجه کنید که رقم نقلی خروجی از جمع‌کننده چپ‌ترین بیت PP(i+۱) است و باید به سمت راست شیفت یابد تا به ثبات A وارد شود. شکل (۲ـ۴b) ضرب (۱ـ۴a) را نشان می‌دهد.

1- Partial product



شکل (۲ـ۴) الگوریتم ضرب با قلم و کاغذ



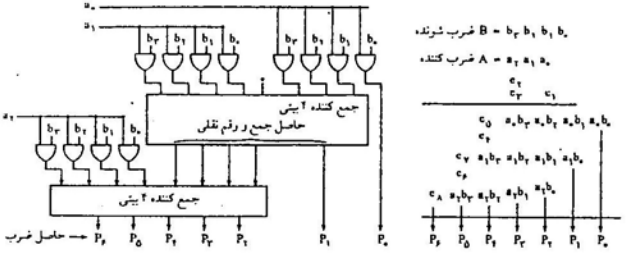


شکل (۳ـ۴)

شکل (۸ـ۴)

ضرب آرایه‌ای

ضرب آرایه‌ای را می‌توان با جمع‌کننده دودویی نیز انجام داد. فرض کنید ضرب‌شونده B و ضرب‌کننده A باشد و B، ۲ بیت و A، ۳ بیت. شکل (۳ـ۴) ساختار ضرب را با جمع‌کننده دودویی نشان می‌دهد. در کل اگر ضرب‌شونده دارای K بیت و ضرب‌کننده دارای J بیت باشد برای ضرب آرایه‌ای به K×J تا گیت AND و J-۱ عدد جمع‌کننده K بیتی نیاز است.



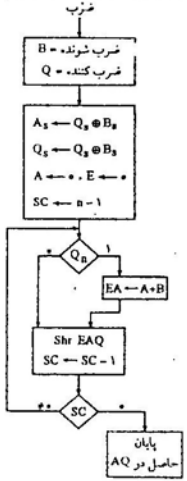
شکل (۳ـ۴)

می‌توان ضرب بالا را با تمام جمع‌کننده و نیم جمع‌کننده نیز انجام داد. با بررسی می‌توان نشان داد که برای این منظور احتیاج به ۵ تمام جمع‌کننده، ۳ نیم جمع‌کننده و ۱۲ گیت AND است. زیرا برای جمع a_0b_1 و a_1b_0 یک H.A نیاز است که رقم نقلی آن به مرتبه بالاتر می‌رسد (c_1). جمع c_1 و a_0b_2 و a_1b_1 یک F.A نیاز دارد رقم نقلی آن (c_2) به مرتبه بالاتر و حاصل جمع آن (s_2) با a_2b_0 باید جمع شود که یک H.A احتیاج دارد، حاصل جمع آن p_2 و رقم نقلی آن نیز (c_3) به مرحله بالاتر و سپس در مرحله بعد باید a_2b_1 و a_1b_2، a_0b_3، c_3، c_2 را با هم جمع کرد که احتیاج به ۲ تا F.A داشته و دو رقم نقلی تولید می‌کند و این دو رقم نقلی با a_1b_3 و a_2b_2 باید جمع شود که احتیاج به یک F.A و یک H.A داشته و دو رقم نقلی تولید می‌شود این دو رقم نقلی با a_2b_3 احتیاج به یک F.A دارد.

ضرب اعداد علامت‌دار

ضرب اعداد در سیستم علامت مقدار، مانند اعداد مثبت است. کافیست ابتدا علامت نتیجه را محاسبه کرده و سپس ضرب را انجام داد. فلوچارت مربوط به این ضرب در شکل ۴ـ۴ مشاهده می‌شود.

الگوریتم شکل (۴ـ۴) ضرب‌کننده ترتیبی است و نسبت به ضرب‌کنندهٔ آرایه‌ای دارای سرعت کمتر می‌باشد. ولی سخت‌افزار ضرب‌کنندهٔ ترتیبی بسیار کمتر از ضرب‌کنندهٔ آرایه‌ای است.



شکل (۴ـ۴)

مطابق فلوچارت شکل (۴ـ۳) علامت حاصل برابر $Q_s \oplus B_s$ می‌باشد (Q_s بیت علامت ضرب‌کننده و B_s بیت علامت ضرب‌شونده) فلیپ‌فلاپ E نگهدارنده رقم نقلی جمع است و در هر بار همراه AQ شیفت داده می‌شود. فرض بر این است که ثباتهای A و Q و B، n بیتی هستند که یک بیت آنها علامت است پس n-۱ بیت را در n-۱ بیت ضرب کرده و حاصل را در دو ثبات AQ ذخیره می‌کنیم. در ابتدا SC برابر n-۱ می‌شود و در هر مرتبه از SC یک واحد کم می‌شود تا صفر شود و الگوریتم خاتمه یابد.

در هر مرتبه بیت مرتبه پایین Q (Q_n) تست می‌شود، اگر یک بود، ضرب شونده B به حاصلضرب جزئی موجود در A اضافه می‌شود و اگر صفر بود هیچ کاری انجام نمی‌شود. سپس EAQ به سمت راست شیفت می‌یابد و مجدداً بیت سمت راست Q تست می‌شود.

برای ضرب اعداد علامت‌دار در سیستم مکمل ۲، ابتدا حالتی را در نظر می‌گیریم که ضرب‌شونده منفی و ضرب‌کننده مثبت باشد. در این حالت موقع افزودن ضرب‌شوندهٔ منفی به حاصل‌ضرب جزئی باید بیت علامت ضرب‌شونده را گسترش دهیم. شکل (۵ـ۴) ضرب دو عدد (۱۳-) و (۱۱+) را نشان می‌دهد. حاصل ۱۰ بیت نیاز دارد. گسترش علامت در شکل با خط زیر مشخص شده است.

حال حالتی را که ضرب‌کننده منفی است در نظر می‌گیریم. یک راه این است که مکمل ۲ی ضرب‌کننده و ضرب‌شونده را محاسبه کنیم، با این عمل علامت حاصل تغییر نخواهد کرد ولی ضرب‌کننده مثبت می‌شود و می‌توان مانند حالت قبل عمل کرد.

```
(-۱۳)  ۱۰۰۱۱  ضرب‌شونده
(+۱۱)  ۰۱۰۱۱  × ضرب‌کننده
       ۱۱۱۱۱۱۰۰۱۱
       ۱۱۱۱۱۰۰۱۱
       ۰۰۰۰۰۰۰۰
       ۱۱۱۰۰۱۱
       ۰۰۰۰۰۰
(-۱۴۳) ۱۱۰۱۱۱۰۰۰۱
```

شکل (۵ـ۴)

الگوریتم ضرب بوث (Booth)

الگوریتم بوث روشی برای ضرب اعداد دودوئی صحیح در سیستم مکمل ۲ است. این الگوریتم بر این اساس استوار است که وجود صفر در ضرب‌کننده احتیاج به جمع ندارد و فقط شیفت نیاز دارد، و یک رشته ۱ در ضرب‌کننده از مرتبه k تا مرتبه m برابر است با $2^{k+1}-2^m$. به عنوان مثال عدد (۱۴)۰۰۱۱۱۰ برابر است با $2^{2+1}-2^1$ بنابراین ضرب M×۱۴ را می‌توان نوشت: $M\times2^4-M\times2^1$. بنابراین ضرب را می‌توان با شیفت ضرب‌شونده M و تفریق انجام داد.

الگوریتم بوث:

۱ـ ضرب‌شونده از حاصلضرب جزئی تفریق می‌شود (کم می‌شود) اگر به اولین یک (از سمت راست) در رشته یک، در ضرب‌کننده برسیم.

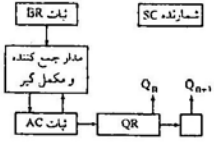
۲ـ ضرب‌شونده به حاصلضرب جزئی افزوده می‌شود اگر به اولین صفر (در یک رشته صفر) در ضرب‌کننده برسیم.

۳ـ حاصلضرب جزئی تغییر نمی‌کند وقتی بیت ضرب‌کننده با بیت قبلی مشابه باشد.

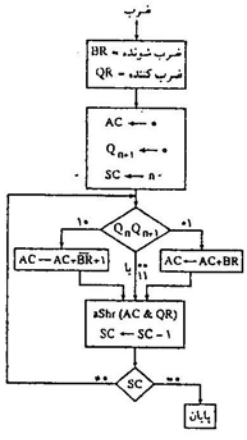
این الگوریتم برای اعداد مثبت و منفی در سیستم مکمل ۲، یکسان است. عدد ۱۲ـ عبارت است از ۱۱۰۰۱۰ که می‌توان نوشت $-۱۴=-2^1-2^2+2^4$.

ثباتهای مورد نیاز برای این الگوریتم در شکل (۴ـ۶) نشان داده شده است. Q_n نشان‌دهنده کم‌ارزش‌ترین بیت ضرب‌کننده QR است. یک فلیپ‌فلاپ اضافه Q_{n+1} به ثبات QR اضافه شده است که مشاهده تغییر بیتها را در ضرب‌کننده ساده کند. فلوچارت ضرب بوث در شکل (۴ـ۷) نشان داده شده است. ثبات AC و فلیپ‌فلاپ Q_{n+1} در ابتدا صفر هستند. SC مقدار n یعنی تعداد بیت‌های ضرب‌کننده را دارد. دو بیت سمت راست ضرب‌کننده Q_n و Q_{n+1} بررسی می‌شوند. اگر دو بیت ۱۰ بود، به این معنی است که به اولین ۱ در یک رشته ۱ رسیده‌ایم و باید تفریق صورت بگیرد (ضرب‌شونده از حاصلضرب جزئی که در AC است کم شود). اگر ۰۱ بود، یعنی به اولین ۰ در یک رشته صفر رسیده‌ایم و باید ضرب‌شونده BR را به AC بیافزاییم اگر دو بیت در Q_n و Q_{n+1} یکی باشند، حاصلضرب جزئی تغییر نمی‌کند.

قدم بعدی این است که حاصلضرب جزئی را همراه با ضرب‌کننده و بیت Q_{n+1} به سمت شیفت راست دهیم. این شیفت، حسابی است و بیت علامت را در AC بدون تغییر باقی می‌گذارد. مثال عددی در شکل (۴ـ۸) آمده است.



شکل (۴ـ۶) سخت‌افزار مورد نیاز بوث



شکل (۴ـ۷) الگوریتم بوث برای ضرب اعداد در سیستم مکمل ۲

○ یک دستور، چند داده[1] (SIMD): آرایه‌های پردازنده در این دسته هستند. ماشینهای SIMD یک واحد کنترل (CU) دارند که مثل ماشین فن‌نیومن عمل می‌کند (یعنی یک دستور اجرا می‌کند). ولی ماشینهای SIMD بیش از یک عنصر پردازشگر[2] (PE) دارند. CU سیگنالهای کنترلی را برای تمام PEها تولید می‌کند که این PEها عمل مشابه روی داده‌های متفاوت انجام می‌دهند.

○ چند دستور، یک داده[3] (MISD): ماشینهایی که چند برنامه متفاوت را روی داده یکسانی انجام می‌دهند. هیچ ماشینی در حال حاضر در این دسته وجود ندارد.

○ چند دستور، چند داده[4] (MIMD): به این دسته چند پردازنده نیز می‌گویند. این ماشینها بیش از یک پردازنده دارند و هر پردازنده می‌تواند برنامه متفاوتی اجرا کند (روی داده‌های خودشان).

زبان انتقال ثبات[5] (RTL)

عملیاتی که بر روی داده‌های یک ثبات صورت می‌گیرد ریزعمل[6] گفته می‌شود. مثالهایی از ریزعمل‌ها عبارتند از Store,Load,Clear,Shift و... به عنوان مثال یک ثبات که دارای امکان load و شیفت به راست و شیفت به چپ است، می‌تواند ریزعملهای load و shift را انجام دهد. ساختار سخت‌افزار داخلی یک کامپیوتر با مشخص شدن موارد زیر تعیین می‌شود:

۱ـ مجموعه ثباتهای کامپیوتر و چگونگی کارکرد و وظایف آنها.

۲ـ توالی انجام ریزعملیات‌ها که روی داده‌های ذخیره شده در ثبات‌ها صورت می‌گیرد.

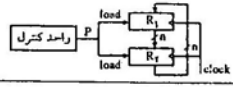
۳ـ کنترلی که توالی ریزعملیات‌ها را مشخص می‌کند.

برای بیان توالی ریزعملیات‌ها و بیان چگونگی انتقال بین ثباتها می‌توان از یک سری نمادهای سمبولیک استفاده کرد. این نمادها که انتقال بین ثبات‌ها را مشخص می‌کند، زبان انتقال ثبات نام دارد. زبان انتقال ثباتی که توصیف می‌شود بسیار ساده و ابتدایی است و برای توصیف کامپیوتر ساده کاربرد دارد.

انتقال ثبات

معمولاً ثبات‌ها را با حروف بزرگ و گاهی به همراه اعداد نشان می‌دهند. مثلاً ثباتی که آدرس حافظه را ذخیره می‌کند[7] MAR نام دارد. ثباتی که شمارنده برنامه است PC و سایر ثباتها نیز نامهایی دارند.

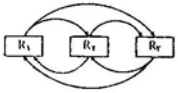
انتقال در ثباتها را با $R_1 \longleftarrow R_2$ نشان می‌دهیم یعنی محتویات R_2 را به R_1 منتقل کن. البته باید مسیر لازم برای این عمل وجود داشته باشد. اگر بنویسیم $P: R_2 \longleftarrow R_1 , R_1 \longleftarrow R_2$ یعنی اگر سیگنال کنترلی P فعال بود محتویات R_1 را به R_2 و R_2 را به R_1 همزمان منتقل کن. علامت ":" بین ریزعملها به معنای همزمانی است. سخت‌افزار لازم برای این عمل در شکل ۱ـ۲ آمده است.



شکل (۲ـ۱)

1- Single Instruction Multiple Data
2- Processing Element
3- Multiple Instruction Single Data
4- Multiple Instruction Multiple Data
5- Register Transfer Language
6- Micro - Operation
7- Memory Address Register

برای انتقال بین ثباتها می‌توان از ارتباط نقطه به نقطه استفاده کرد. در این روش اگر n گره (ثبات) داشته باشیم احتیاج به $n(n-1)$ ارتباط هستیم. (شکل ۱ـ۳)

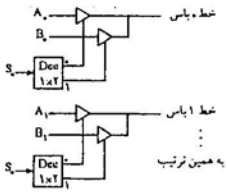


شکل (۳ـ۱)

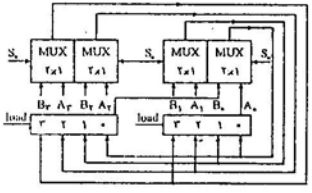
اما این روش، سیم‌بندی زیادی احتیاج دارد.

بنابراین از یک گذرگاه مشترک استفاده می‌کنیم.

اگر در گذرگاه مشترک، خط ارتباطی قطع شود هیچ‌کدام از ثباتها نمی‌توانند با هم ارتباط داشته باشند و همچنین در این روش در آن واحد فقط یک انتقال صورت می‌گیرد. شکل (۱ـ۴) طرح یک گذرگاه مشترک را با استفاده از مالتی‌پلکسر[1] و بافر[2] سه‌حالته نشان می‌دهد.



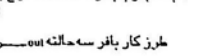
شکل (۴ـ۱b) دقت کنید می‌توان از یک رمزگشا استفاده کرد.



شکل (۴ـ۱a)

برای طراحی باس[3] برای n رجیستر m بیتی با مالتی‌پلکسر احتیاج به m مالتی‌پلکسر n به ۱ است (البته اگر n توانی از ۲ باشد). و با بافر سه‌حالته احتیاج به یک رمزگشا[4] و mn بافر سه‌حالته است. و اندازه رمزگشا باید $\log_2^n \times n$ باشد.[5]

طرز کار بافر سه‌حالته A مطابق جدول زیر است:



c	out
۰	high Impedance
۱	A

تابع خروجی بافر عبارت است از: Out = C.A

یعنی اگر C که ورودی کنترل است صفر باشد خروجی به هیچ ولتاژی وصل نیست و اگر C=۱، خروجی برابر ورودی است. از آنجایی که در شکل (۱ـ۴b) خروجی بافرها فقط یکی در هر لحظه مقدار دارد، می‌توان آنها را به هم وصل کرد. در ضمن اتصال خروجی بافرها مانند OR عمل می‌کند.

1- Multiplexer
2- Buffer
3- Bus
4- Decoder

۵ـ اگر n توانی از ۲ نباشد آنگاه مالتی پلکسرها $\lceil \log_2 n \rceil$ ۲ به یک (۱ × $\lceil \log_2 n \rceil$ ۲) خواهند بود و رمزگشای مورد استفاده، $\lceil \log_2^n \rceil \times 2^{\lceil \log_2 n \rceil}$ می‌باشد. (منظور از ⌈ ⌉ تابع سقف است.)

Q_nQ_{n+1}		AC	QR	Q_{n+1}	SC
۱۰	تفریق	۰۰۰۰۰	۱۰۰۱۱	۰	۵
		+۱۰۰۱۰			
		۱۰۰۱۰			
	ashr	۰۰۱۰۰	۱۱۰۰۱	۱	۴
۱۱	ashr	۰۰۰۱۰	۰۱۱۰۰	۱	۳
۰۱	BR جمع	۰۰۰۱۰			
		+۱۰۱۱۱			
		۱۱۰۰۱			
۰	ashr	۱۱۱۰۰	۱۰۱۱۰	۰	۲
۰۰	ashr	۱۱۱۱۰	۰۱۰۱۱	۰	۱
۱۰	تفریق	۰۰۱۱۱			
	ashr	۰۰۰۱۱	۱۰۱۰۱	۱	۰

BR=۱۰۱۱۱ (-۹)
QR=۱۰۰۱۱ (-۱۳)
n=۵
$\overline{BR}$+۱=۰۱۰۰۱

⇐ حاصل=۰۰۰۱۱۱۰۱۰۱

شکل (۸-۴) مثال عددی ضرب الگوریتم بوث

الگوریتم‌های تقسیم

روش تقسیم دو عدد دودویی در حالت عادی بدین صورت است که ابتدا به تعداد بیتهای تقسیم‌کننده (مقسوم‌علیه) از سمت چپ تقسیم‌شونده (مقسوم) جدا می‌شود و در صورتی که این مقدار از مقدار مقسوم‌علیه کمتر بود، یک بیت دیگر از مقسوم نیز در نظر گرفته می‌شود. سپس تقسیم‌کننده یک واحد به راست شیفت داده می‌شود و از بیتهای جدا شدهٔ مقسوم در مرحله اول یا از باقیماندهٔ میانی (Partial remainder) در مراحل بعد کم می‌شود و یک رقم یک نیز در سمت راست خارج قسمت اضافه می‌گردد. حاصل تفریق به همراه بیت بعدی مقسوم، مجدداً به مقسوم‌علیه تقسیم می‌شود اگر بزرگتر یا مساوی مقسوم‌علیه بود یک در سمت راست خارج قسمت قرار می‌گیرد در غیر این صورت صفر قرار گرفته و بیت بعدی مقسوم سمت راست باقیمانده میانی قرار می‌گیرد. این مراحل تا زمانی که باقیمانده کمتر از مقسوم‌علیه باشد ادامه می‌یابد.

پیاده‌سازی (برای سیستم اندازه-علامت)

به جای شیفت به راست مقسوم‌علیه، مقسوم شیفت به چپ داده می‌شود. عمل تفریق نیز با مکمل ۲ انجام می‌شود. مقایسه نیز با استفاده از رقم نقلی حاصل از تفریق صورت می‌گیرد. سخت‌افزار لازم برای عمل تقسیم همان سخت‌افزار شکل (۶-۴) است. مجموعهٔ AQ شامل عدد مقسوم، ثبات B شامل مقسوم‌علیه، فلیپ‌فلاپهای E، A_s، B_s، Q_s نیز به ترتیب بیتهای علامت ثباتهای A، AQ، B و Q می‌باشند. ثبات SC به عنوان شمارنده، تعداد بیت‌های مقسوم‌علیه یا خارج قسمت را دارد. لازم به ذکر است که پس از عمل تقسیم در ثبات Q خارج قسمت، و در ثبات A باقیمانده وجود خواهد داشت.

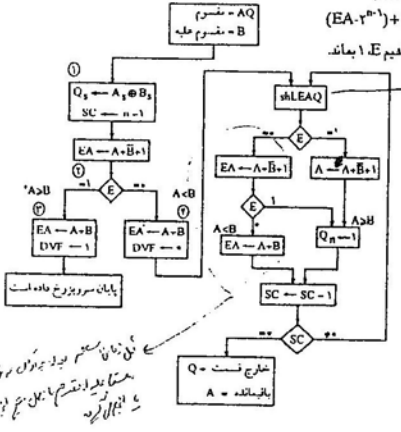
وقوع سرریز (overflow) در عمل تقسیم نیز ممکن است و وقتی است که خارج قسمت به دست آمده از محدودهٔ ثبات اضافه‌تر باشد.

اگر مجموعهٔ AQ، ۲n بیتی (مقسوم)، و B (مقسوم‌علیه)، n بیتی باشد، سرریز وقتی رخ می‌دهد که نیمهٔ باارزش‌تر مقسوم یعنی A از مقسوم‌علیه یعنی B، بزرگتر یا با آن مساوی باشد. در این صورت خارج قسمت بیش از n بیت خواهد بود. برای اعلام وقوع سرریز، فلیپ‌فلاپ دیگری به نام DVF (فلیپ‌فلاپ سرریز تقسیم، Divide overflow Flip-Flop) در سخت‌افزار در نظر گرفته می‌شود. اگر B صفر باشد (تقسیم بر صفر) نیز توسط فلیپ فلاپ DVF مشخص می‌شود. شکل (۹-۴) روش را نشان می‌دهد. همانگونه که در شکل مشاهده می‌شود، AQ شامل مقسوم و B شامل مقسوم‌علیه است، Q_s حاصل از XOR، A_s، B_s است که علامت خارج قسمت را ذخیره می‌کند و ثبات SC نیز مقدار n-۱ (تعداد بیتها بجز بیت علامت) را در خود ذخیره می‌کند. در مرحله بعد وقوع سرریز بررسی می‌گردد، با این روش که A-B در EA قرار می‌گیرد و اگر E=۱ یعنی A≥B و اگر E=۰ یعنی A<B خواهد بود. پس اگر E=۱، سرریز رخ داده است. در هر دو حالت مقدار قبلی A با افزودن B به A حاصل می‌شود. برای شروع عمل تقسیم مجموعهٔ EAQ به چپ شیفت داده می‌شود اگر E صفر شود، B از A کم شده و مقدار E مشخص می‌کند که آیا باید به سمت راست خارج قسمت (Q_n) یک اضافه کرد یا خیر. اگر E=۰ یعنی (A<B) بدین معنی که باید به سمت راست خارج قسمت صفر اضافه گردد (که این صفر با شیفت به چپ بدست می‌آید)، در این حالت عمل تفریق نبایستی انجام شود پس B با A جمع می‌شود تا مقدار قبلی حاصل گردد.

اگر بیت شیفت یافته به E، برابر ۱ باشد، یعنی EA>B چون EA دارای ۱ بیت E و n-۱ بیت A است در حالیکه B دارای n-۱ بیت بوده و EA از B بزرگتر است. در این حالت B باید از EA کم شده و Q_n=۱ شود. از آنجایی که ثبات A بیت باارزش خود را از دست داده است (این بیت در E قرار دارد)، مقدار برابر $EA-2^{n-1}$ است. (2^{n-1} همان بیت باارزش A می‌باشد). اگر به این مقدار مکمل ۲ عدد B را بیافزاییم:

$$(EA-2^{n-1})+(2^{n-1}-B)=EA-B$$

رقم نقلی این جمع به E وارد نمی‌شود اگر ما بخواهیم E، ۱ بماند.



① Q_s علامت نتیجه است.

② برای مقایسه A-B می‌شود.

③ A≥B است پس DVF ← ۱ شده و مقدار قبلی A بدست می‌آید.

④ A<B و مقدار قبلی A بدست می‌آید.

شکل (۹-۴)

دیگر الگوریتم‌ها

روشی که توضیح داده شد، به روش بازیابی (restoring) معروف می‌باشد. علت انتخاب این نام این است که باقیماندهٔ جزئی با جمع کردن مقسوم‌علیه با حاصل منفی، بازیابی می‌شود. دو روش دیگر برای تقسیم اعداد وجود دارد، روش مقایسه (comparison) و روش nonrestoring. در روش مقایسه قبل از عمل تفریق، A و B با هم مقایسه می‌شوند. سپس اگر $A \geq B$، B از A کم می‌شود. اگر $A < B$ هیچ عملی انجام نمی‌شود. باقیماندهٔ جزئی به سمت چپ شیفت می‌یابد و اعداد مجدداً مقایسه می‌شوند. مقایسه را می‌توان قبل از عمل تفریق انجام داد با مشاهده رقم نقلی خروجی حاصل از جمع‌کننده موازی، قبل از اینکه به ثبات E وارد شود.

در روش nonrestoring، اگر حاصل تفریق منفی شود، B به آن اضافه نمی‌شود، حاصل منفی به سمت چپ شیفت می‌یابد و سپس B به آن اضافه می‌شود. برای اینکه ببینیم چرا این روش صحیح است، حالتی که $A<B$ است را در نظر بگیرید. در حالت قبلی (شکل ۹-۴) توجه کنید که ابتدا A-B شده، سپس با B جمع می‌شود یعنی A-B+B که A بازیابی شود. سپس این عدد شیفت به چپ می‌یابد (در ۲ ضرب می‌شود) و B از آن مجدداً کم می‌شود.
یعنی حاصل $2(A-B+B)-B=2A-B$ محاسبه می‌شود.

این نتیجه در روش nonrestoring به این شکل به دست می‌آید که A-B را با B جمع نمی‌کنیم. در دور بعد عدد شیفت به چپ می‌یابد و سپس B به آن اضافه می‌شود $2(A-B)+B=2A-B$، که همان قبلی است. بنابراین در این روش، B کم می‌شود اگر مقدار قبلی Q_n یک باشد، ولی B اضافه می‌شود اگر مقدار قبلی Q_n صفر باشد، و به این ترتیب هیچ بازیابی باقیمانده جزئی نیاز نیست. این روش دیگر، مقسوم‌علیه را اگر $A<B$ باشد اضافه نمی‌کند ولی در عوض احتیاج به منطق کنترلی دارد که حاصل قبلی را به خاطر بسپارد. اولین بار که مقسوم شیفت می‌یابد، B باید کم شود. همچنین، اگر بیت آخر خارج قسمت ۰ باشد، باقیمانده جزئی باید بازیابی شود تا باقیمانده صحیح نهایی بدست آید.

اعداد دودوئی ممیز شناور (floating-point):

یک عدد ممیز شناور از دو قسمت تشکیل شده است: مانتیس m و نمای e. اگر مبنای عدد r باشد، مقدار برابر است با $m \times r^e$.
برای ذخیره در ثبات فقط مقدار m و e ذخیره می‌شود. به عنوان مثال عدد ۵۳۷/۲۵ در مبنای ۱۰ به شکل $0/537725 \times 10^3$ می‌باشد که $m=537725$ و $e=3$ می‌باشد.

یک عدد ممیز شناور را نرمال شده گویند اگر باارزش‌ترین رقم مانتیس مخالف صفر باشد. مثلاً $0/011 \times 2^{101}$ در مبنای ۲ نرمال نیست ولی $0/110 \times 2^{100}$ نرمال است.

عدد صفر را نمی‌توان نرمال کرد و برای نمایش، هم مانتیس و هم نما، تماماً صفر هستند. نمایش ممیز شناور محدودهٔ اعداد قابل نمایش را افزایش می‌دهد. یک کامپیوتر را که دارای کلمات ۳۸ بیتی است در نظر بگیرید. یک بیت برای علامت در نظر گرفته می‌شود. در سیستم ممیز ثابت، محدودهٔ اعداد $\pm(2^{37}-1)$ یعنی تقریباً $\pm 10^{12}$ خواهد بود. در سیستم ممیز شناور، اگر مانتیس ۲۶ بیت و نما ۱۲ بیت در نظر گرفته شود؛ و دو بیت برای علامت (علامت مانتیس و علامت نما) در نظر بگیریم، محدودهٔ اعداد قابل نمایش عبارت خواهد بود از: $\pm(2^{2047} \times (2^{-25}-1))$. (دقت شود $2^{11}-1=2047$) بزرگترین عدد در این نمایش تقریباً 10^{615} خواهد بود!

معمولاً مانتیس در سیستم علامت مقدار است و نما می‌تواند: ۱- علامت مقدار باشد ۲- در سیستم مکمل ۲ باشد ۳- مکمل ۱ باشد ۴- در حالت افزونه (Biased) باشد.

مقدار بایاس یا افزونه عددی است که به هر نما افزوده می‌شود تا تمام نماها مثبت شوند. مثلاً اگر محدودهٔ نماها از ۱۶- تا ۱۵ باشد هر نما را با ۱۶ جمع کرده تا مثبت نمایش داده شوند. و در واقع نما به شکل $e+16$ است که e مقدار واقعی است. مزیت آن، این است که نماها مثبت هستند و برای مقایسه ساده‌تر است. (برای جمع و تفریق، نماها باید مقایسه شوند و یکسان شوند). در ادامه فرض ما این است که نماها به صورت bias ذخیره شده‌اند.

جمع و تفریق ممیز شناور

برای جمع کردن و تفریق کردن دو عدد ممیز شناور، در ابتدا باید دو عدد تنظیم شوند (Align) تا نماها یکسان شوند. برای یکسان کردن نماها می‌توان آنکه توان کمتر دارد را انتخاب کرده و مانتیس آن را به راست شیفت داد و توان را افزایش داد. و یا می‌توان آنکه توان بیشتری دارد را انتخاب و مانتیس آن را به چپ شیفت داد و از نما کم کرد. در حالت دوم بیت‌های مرتبه بالا از دست می‌رود و خطا پدید می‌آید. مثلاً در مبنای ۱۰: $0/2130 \times 10^3 + 0/9101 \times 10^2$ عددی که نمای کمتری دارد را انتخاب کرده و مانتیس آن را به راست شیفت می‌دهیم: $0/2130 \times 10^3 + 0/0091 \times 10^3$ پس حاصل $0/2221 \times 10^3$ است. با شیفت به راست ممکن است بیتهای مرتبه پایین از دست برود که موجب کاهش دقت می‌شود. پس از جمع امکان سرریز وجود دارد، که رقم سرریز را با شیفت به راست و افزایش یک واحد توان به مانتیس وارد کرده و سرریز اصلاح می‌گردد.

پس از عمل تفریق امکان وقوع زیرریز (under flow) است و آن وقتی است که حاصل دارای باارزش‌ترین رقم صفر در مانتیس شود که با شیفت به چپ مانتیس، و کاهش توان اصلاح می‌شود.

در کل برای جمع و تفریق دو عدد ممیز شناور باید اعمال زیر صورت گیرد:

۱- بررسی صفر بودن هر کدام از اعداد
۲- تنظیم نمودن مانتیسها
۳- اضافه یا کم کردن مانتیسها
۴- نرمال کردن نتیجه

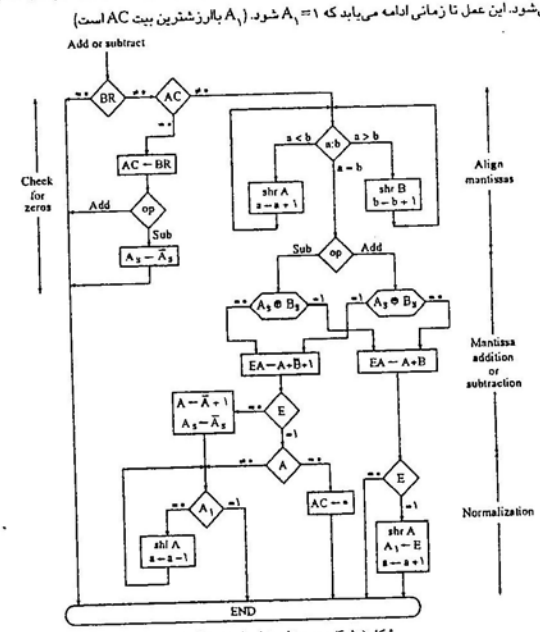
فلوچارت در شکل (۱۰-۴) نشان داده شده است. فرض بر این است که اعداد در AC و BR قرار دارند. هر ثبات شامل سه بخش است: علامت مانتیس، مانتیس و نما. A_s و B_s علامت مانتیس است. A و B مانتیس. و a و b نما. در ضمن نماها به صورت بایاس ذخیره شده است تا مقایسه نماها راحت انجام شود.

ابتدا صفر بودن هر یک از دو عدد بررسی می‌شود، زیرا در حالت صفر بودن هر یک از دو عدد ممیز شناور نرمالیزه بودن آن معنی ندارد و بنابراین جداگانه بررسی می‌گردد. نتیجه نهایی در AC ذخیره می‌گردد. اگر BR صفر باشد نتیجه همان AC است. اگر AC صفر و عمل تفریق باشد، منفی BR در AC قرار می‌گیرد (بیت علامت AC عوض می‌شود).

اگر هیچ کدام صفر نباشند، توانها (b و a) با هم مقایسه می‌شوند. اگر نماها برابر نباشند، مانتیسی که دارای نمای کمتری است به سمت راست شیفت می‌یابد و نمای آن افزایش می‌یابد؛ تا زمانی که دو نما مساوی شوند.

جمع و تفریق مانتیسها مانند جمع و تفریق دو عدد در سیستم علامت مقدار است. دو مانتیس بنا به علامت و عمل مورد نظر، با هم جمع یا از هم کم می‌شوند. اگر در حین عمل جمع بین مانتیسها، سرریز رخ دهد، به فلیپ‌فلاپ E وارد می‌شود. اگر E=۱ باشد به A_1 منتقل می‌شود و تمام بیتهای A به راست شیفت می‌یابند و نما یکی افزوده می‌شود.

اگر دو مانتیس از هم کم شوند، امکان زیرریز یا صفر شدن حاصل است. اگر حاصل صفر شود، تمام AC صفر می‌شود. در غیر این صورت، اگر زیرریز رخ دهد و حاصل صفر نباشد، یعنی $A_1 = ۰$ مانتیس به چپ شیفت می‌یابد و از نما کم می‌شود. این عمل تا زمانی ادامه می‌یابد که $A_1 = ۱$ شود. (A_1 بالارزشترین بیت AC است)



شکل (۱۰ـ۴) جمع و تفریق اعداد ممیز شناور

ضرب ممیز شناور

مانتیسها در هم ضرب و نماها با هم جمع می‌شوند. هیچ نیازی به مقایسه نماها و تنظیم مانتیس نیست. ضرب مانتیسها مانند ضرب دو عدد ممیز ثابت در سیستم علامت مقدار انجام می‌شود.

تنها تفاوت این است که پس از ضرب نیمه با اولویت بالاتر حاصل ذخیره می‌شود و از نیمهٔ پایینی صرف نظر می‌شود. مراحل ضرب:

۱ـ تست صفر بودن دو عدد

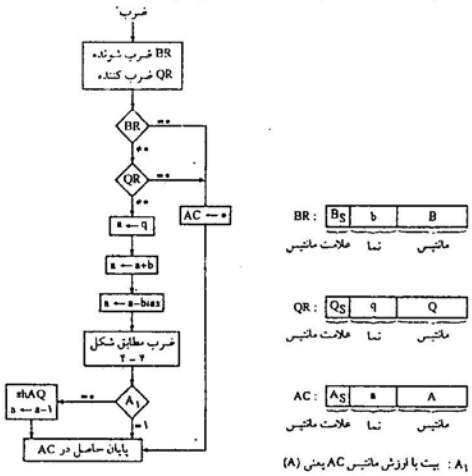
۲ـ افزودن نماها

۳ـ ضرب مانتیسها

۴ـ نرمال کردن نتیجه

فلوچارت ضرب در شکل (۱۱ـ۴) آمده است.

ابتدا نماها با هم جمع می‌شود و حاصل جمع نماها منهای مقدار بایاس می‌شود. چون هر دو نما بایاس شده‌اند. بقیه مراحل مشخص است. در ضرب امکان رویداد زیرریز است ولی سرریز رخ نمی‌دهد.



شکل (۱۱ـ۴) ضرب دو عدد ممیز شناور

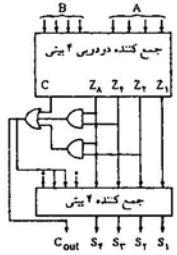
تقسیم ممیز شناور

مانتیسها به هم تقسیم و نماها از هم کم می‌شوند. تقسیم مانتیسها مانند حالت ممیز ثابت صورت می‌گیرد فقط در این حالت مقسوم یک کلمه است که در AC می‌باشد (بر خلاف حالت ممیز ثابت که دو ثبات را مقسوم می‌گرفتیم). پس از انجام عمل تقسیم، باید مقدار بایاس را به نمای حاصل افزود. زیرا مقادیر بایاس هنگام تفریق نماها حذف می‌شود.

واحد محاسباتی دسیمال

۱ـ جمع‌کننده (BCD): جمع BCD در درس مدار منطقی معرفی شده است. جمع دو رقم BCD حداکثر ۱۸ می‌باشد که با احتساب رقم نقلی حداکثر ۱۹ می‌باشد. اگر از جمع‌کننده دودویی مطابق شکل (۱۲ـ۴) استفاده کنیم، اگر حاصل بیش از ۹

باشد باید تصحیح شود. مثلاً جمع‌کننده دودویی حاصل ۱۰ را به شکل ۱۰۱۰ محاسبه می‌کند در صورتی که ۱۰ در کد BCD به شکل ۱۰۰۰۰ می‌باشد. برای تصحیح حاصل، همانطور که در درس منطقی دیده‌اید، باید حاصل را با ۶ جمع کرد.



شکل (۱۳ـ۴) جمع‌کننده BCD

۲ـ تفریق (BCD): برای تفریق دو رقم در کد BCD، بهتر است رقم اول با مکمل ۱۰ رقم دوم جمع شود. برای این منظور باید ابتدا مکمل ۹ عدد دوم یعنی B را محاسبه کنیم و سپس ۱ را به آن بیافزاییم. از آنجائیکه BCD، خود مکمل نیست نمی‌توان بیتهای B را مکمل کرد. در راه برای یافتن مکمل ۹ رقم BCD وجود دارد. اول اینکه ابتدا بیتهای رقم B را مکمل کرده و سپس عدد ۱۰۱۰ را به آن می‌افزاییم و از رقم نقلی صرفنظر می‌کنیم. به عنوان مثال مکمل ۹ رقم ۰۱۱۱ (۷) به این شکل محاسبه می‌شود:

$$\text{۰۱۱۱} \xrightarrow{\text{مکمل ۱}} \text{۱۰۰۰} \xrightarrow{\text{افزودن ۱۰۱۰}} \text{۱۰۰۱۰} \xrightarrow[\text{نقلی}]{\text{صرفنظر از رقم}} \text{۰۰۱۰}$$

در روش دوم، ابتدا ۰۱۱۰ (۶) را به رقم اضافه می‌کنیم و سپس مکمل ۱ رقم حاصل را محاسبه می‌کنیم.

مثال:

$$\text{۰۱۱۱} \xrightarrow{\text{افزودن ۰۱۱۰}} \text{۱۱۰۱} \xrightarrow{\text{مکمل ۱}} \text{۰۰۱۰}$$

به هر حال برای یافتن مکمل ۹ باید یک مدار ترکیبی طراحی کرد.

فرض کنید می‌خواهیم یک مدار جمع‌کننده/تفریق‌کننده طراحی کنیم که بر اساس یک ورودی M، جمع یا تفریق کند یعنی:

$M=0 \Rightarrow$ جمع $A+B$

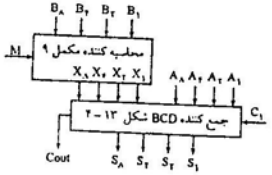
$M=1 \Rightarrow$ تفریق $A-B$

اگر برای تفریق از مکمل ۱۰ استفاده کنیم یعنی $A-B=A+(\text{مکمل ۹ } B)+1$، و اگر $A=A_۴A_۳A_۲A_۱$ و $B=B_۴B_۳B_۲B_۱$ و $X=X_۴X_۳X_۲X_۱$ خروجی مدار محاسبه‌کننده مکمل ۹ باشد می‌توان نشان داد:

$$\begin{cases} x_۱=B_۱\overline{M}+\overline{B}_۱M \\ x_۲=B_۲ \\ x_۳=B_۳\overline{M}+(\overline{B}_۳B_۲+B_۳\overline{B}_۲)M \\ x_۴=B_۴\overline{M}+\overline{B}_۴\overline{B}_۳\overline{B}_۲M \end{cases}$$

با توجه به این معادلات می‌توان مشاهده کرد که اگر $M=0$ آنگاه $X=B$ و اگر $M=1$ باشد X برابر مکمل ۹، B می‌شود. مدار مطابق شکل ۱۴ـ۴ است.

در شکل ۱۴ـ۴ برای انجام عمل تفریق باید $M=1$ و $C_1=1$ باشد که A با مکمل ۱۰ عدد B جمع می‌شود و برای جمع باید $M=0$ باشد که مدار محاسبه‌کننده مکمل ۹، B را تحویل دهد.



شکل (۱۴ـ۴)

جمع و تفریق دو عدد (چند رقمی) دسیمال

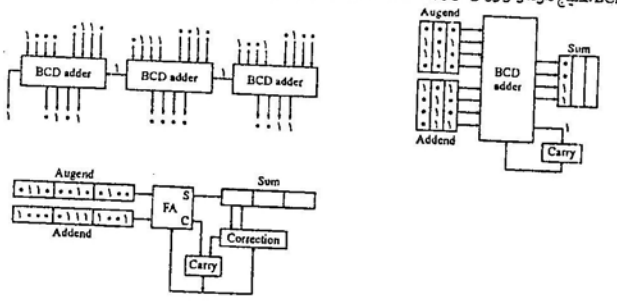
دو عدد دسیمال را می‌توان به سه طریق جمع کرد (مطابق شکل ۱۵ـ۴).

۱ـ روش موازی: این روش به ازای هر رقم، یک جمع‌کننده BCD استفاده می‌کند. جمع به طور موازی محاسبه شده و تنها یک ریزعمل احتیاج دارد.

۲ـ روش رقم سری - بیت موازی (digit-serial bit-parallel): ارقام به یک جمع‌کننده BCD به طور سری اعمال می‌شوند. در حالیکه بیتهای هر رقم موازی وارد می‌شود. ارقام BCD در هر کلاک یک بار شیفت می‌یابند و یکی یکی به جمع‌کننده وارد می‌شوند. برای یک عدد K رقمی، K ریزعمل احتیاج داریم.

۳ـ جمع‌کننده کاملاً سری: بیتها یکی یکی به یک تمام جمع‌کننده (F.A) شیفت می‌یابند. جمع که پس از هر ۴ شیفت بدست می‌آید، باید به رقم درست BCD تصحیح شود. یعنی باید حاصل را چک کرد و اگر از ۱۰۰۱ بزرگتر بود، آن را با ۰۱۱۰ جمع می‌کنیم که به این ترتیب رقم اصلاح شده و یک بیت نقلی ایجاد می‌کند که برای مرحله بالاتر استفاده می‌کنیم.

روش موازی، سریع است ولی تعداد زیادی جمع‌کننده احتیاج دارد. روش رقم سری - بیت موازی فقط یک جمع‌کننده BCD احتیاج دارد، و از روش موازی کندتر است. روش سری بسیار کند است.



شکل (۱۵ـ۴) سه راه جمع اعداد دسیمال

تست‌ها

۱- نتیجه حاصل از انجام محاسبات در ALU ممکن است باعث Set شدن برخی از فلیپ‌فلاپ‌ها در Status Register شود. اگر برای نمایش اطلاعات از ۲'S Compiement استفاده شود. کدامیک از عبارات زیر به ترتیب وجود Overflow در جمع و نتیجه صفر را تشخیص می‌دهد؟ (مهندسی کامپیوتر ۷۱)

الف) رقم نقلی نتیجه، AND منطقی بیت‌های نتیجه

ب) Exclusive - OR ارقام نقلی تولیدشده از موضع sign - bit و ماقبل آن، NOR بیت‌های نتیجه

ج) Exclusive - NOR ارقام نقلی تولیدشده از موضع sign - bit و ماقبل آن، AND بیت‌های نتیجه

د) رقم نقلی نتیجه، NAND منطقی بیت‌های نتیجه

۲- می‌خواهیم دو عدد علامت‌دار را (با قراردادن مکمل ۲ها) در هم ضرب کنیم. (مهندسی کامپیوتر ۷۱)

A × (مضروب) multiplicand

B (مضروب فیه) multiplier

الف) علامت A و B را مثبت می‌کنیم و سپس ضرب می‌کنیم و پس از آن در صورت لزوم علامت را اصلاح می‌کنیم.

ب) اگر A منفی بود آنرا مثبت می‌کنیم، ضرب را با گسترش علامت انجام می‌دهیم، حاصلضرب را در صورت لزوم تغییر علامت می‌دهیم.

ج) اگر B منفی بود آنرا مثبت کرده، ضرب را با گسترش علامت انجام می‌دهیم، حاصلضرب را در صورت لزوم تغییر علامت می‌دهیم.

د) ضرب را با گسترش علامت انجام می‌دهیم.

۳- الگوریتم Booth دارای کدامیک از ویژگیهای زیر است؟ (مهندسی کامپیوتر ۷۴)

الف) سرعت محاسبه با تعداد یکها متناسب است.

ب) تعداد عملیات جمع برابر نصف تعداد یکها و عملیات جابجایی (shift) برابر تعداد بیتهاست.

ج) تعداد زوج بیتهای صفر و یک یا یک و صفر تعداد جمعها و تعداد کل بیتها تعداد عملیات جابجایی را تعیین می‌کند.

د) سرعت محاسبه با تعداد بیتها متناسب است.

۴- یک ضرب سه‌بیتی در سه بیتی را با کدام عناصر زیر می‌توان انجام داد؟ (مهندسی کامپیوتر ۷۴)

الف) ۹ گیت And+۳ جمع‌کننده نیم‌افزا (Halfadder) + ۳ جمع‌کننده تمام افزا (Full adder)

ب) ۶ گیت And+۶ جمع‌کننده نیم افزا

ج) ۹ گیت And+۶ جمع‌کننده تمام افزا + ۳ جمع‌کننده نیم افزا

د) ۶ گیت And+۳ جمع‌کننده نیم افزا + ۶ جمع‌کننده تمام افزا

۵- با فرض داشتن یک پردازش هنجار شده (normalized) و ثابتهای ۵ رقمی برای ماشین و نمای اضافه ۱۶ (۱۶ Excess) ۵ بیتی محاسبه زیر چه جوابی خواهد داشت؟ (مهندسی کامپیوتر ۷۴)

$(0/4F36)_{16} \times 16^{(1010)_2} + (0/A8)_{16} \times 16^{(1000)_2}$ (اعداد را مثبت فرض نمایید.)

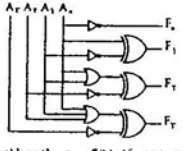
الف) $(0/4F36) \times 16^{(1010)_2}$

ب) $(0/4F36A) \times 16^{(1010)_2}$

ج) $(0/4F40A) \times 16^{(1010)_2}$

د) $(0/4F40A) \times 16^{(1000)_2}$

۶- با توجه به مدار زیر و به فرض اینکه اعداد a و f در سیستم مکمل دو باشند رابطه بین این دو عدد چهار بیتی $(f=F_3F_2F_1F_0, a=A_3A_2A_1A_0)$ چیست؟ (مهندسی کامپیوتر ۷۵)



الف) f مکمل a است.

ب) f برابر با منفی a است.

ج) f برابر a+1 است.

د) f برابر a-1 است.

۷- اگر بخواهیم ضرب علامت‌دار دو عدد ۰۰۰۰۰۱۱ و ۰۱۱۱۰۱ را با الگوریتم booth انجام دهیم. با فرض اینکه هر عمل جمع ۱۰ns و هر عمل شیفت ۲ns و هر مکمل‌گیری 0ns طول بکشد، زمان ضرب با این الگوریتم و حاصل ضرب به ترتیب برابر کدام مقدار خواهد بود؟ (زمان آماده‌سازی اولیه انباشتگر صفر در نظرگرفته می‌شود). (مهندسی کامپیوتر ۷۵)

الف) ۴۲ ns و ۰۰۰۰۰۱۰۱۱۱۱

ب) ۶۲ ns و ۰۰۰۰۱۰۱۰۱۱۱

ج) ۶۱ns و ۰۰۰۰۰۰۱۱۰۱۱۱

د) ۴۱ ns و ۰۰۰۰۰۱۰۱۰۱۱۱

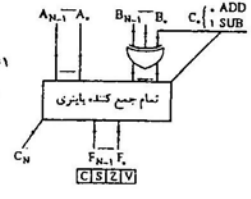
۸- مشخص کنید که با توجه به بخشی از راحد یک CPU که در شکل نشان داده شده است، در اجرای دستورالعمل [IF(A-B) adr۱,adr۲,adr۳] که در آن A و B اعداد با علامت می‌باشند، (و برای اعداد منفی از مکمل دو استفاده می‌شود) چه نوایعی از پرچم‌ها (flag) به عنوان شرط رفتن به adr۲,adr۱,adr۳ استفاده می‌شود؟ (مهندسی کامپیوتر ۷۵)

توضیح: اثر دستور فوق‌الذکر:

{if(A<B) then PC ← adr۱

else {if(A=B) then PC ← adr۲}

else PC ← adr۳



	adr۱	adr۲	adr۳
الف)	$(v \oplus s)=1$	$z=1$	$(s \oplus v)'.z'=1$
ب)	$(v \oplus s)=1$	$z=0$	$(s \oplus v)=0$
ج)	$(v \oplus s)=1$	$z=1$	$(s \oplus v)=1$
د)	$(v \oplus s)=1$	$z=0$	$(s \oplus v)=1$

فصل پنجم

موازات و پایپلاین

پردازش موازی

به معنای توزیع وظایف یا taskها بین چندین پردازنده یا چند عنصر پردازشی به منظور نیل به سرعت بیشتر است. در واقع به جای پردازش هر دستور به طور ترتیبی، یک سیستم با پردازش موازی، قادر به انجام همزمان دستورات به منظور دستیابی به سرعت بیشتر است.

به عنوان مثال، یکی از جنبه‌های توازی می‌تواند این باشد: زمانی که در حال خواندن داده‌ای از حافظه هستیم همزمان دستور دیگری در ALU در حال اجرا باشد.

هدف از پردازش موازی، افزایش قدرت پردازش کامپیوتر و افزایش توان عملیاتی سیستم (توان عملیاتی یعنی تعداد پردازشی که در یک مدت زمان معین قابل انجام است) می‌باشد. میزان سخت‌افزار و همچنین قیمت کامپیوتر با پردازش موازی افزایش می‌یابد.

همانطور که در فصل یک توضیح داده شد کامپیوترها بر اساس نظر آقای Flynn به چهار دسته تقسیم می‌شوند که در هر یک درجه‌ای از توازی موجود بود.

تسریع[1]

تسریع (Speed up) به بیان ساده عبارت است از نسبت سرعت‌های دو محاسبه از یک مسئله. به عنوان مثال اگر شما یک پردازنده در اختیار داشته باشید و زمان اجرای برنامه خاصی روی این پردازنده برابر t_1 باشد. و اگر P پردازنده در اختیار داشته باشد و زمان اجرای همان برنامه روی این P پردازنده (پردازش موازی) برابر t_p باشد آنگاه:

$$\text{تسریع} = \frac{\text{سرعت P پردازنده}}{\text{سرعت یک پردازنده}} = \frac{\text{زمان اجرا روی یک پردازنده}}{\text{زمان اجرا روی P پردازنده}} = \frac{t_1}{t_p}$$

توجه کنید که ($t_p < t_1$)

شاید انتظار بر آن است که زمان اجرای برنامه روی P پردازنده $\frac{1}{P}$ زمان اجرا با یک پردازنده باشد (یعنی سرعت P پردازنده، P برابر سرعت یک پردازنده باشد). ولی در واقعیت اینطور نیست و تسریع به برنامه اجرا شده و میزان تبادل اطلاعات بین P پردازنده بستگی دارد. اگر فرض کنیم f درصد از دستورات برنامه قابل موازی شدن نیستند یعنی نتوان آنها را با P پردازنده اجرا کرد و مجبور باشیم به ترتیب و توسط یک پردازنده آن دستورات را اجرا کنیم (مثلاً دستوراتی که به یکدیگر وابستگی داده‌ای دارند)، در این صورت $1-f$ درصد برنامه قابل موازی‌شدن می‌باشد (به f مؤلفه ترتیبی گویند).

حال اگر زمان اجرا روی یک پردازنده را t بگیریم، زمان اجرا با P پردازنده برابر است با:

$$t_p = \left(f + \frac{1-f}{P}\right)t$$

چون f درصد برنامه توسط یک پردازنده و $1-f$ درصد باقیمانده در P پردازنده تقسیم می‌شود. پس تسریع عبارت است:

$$s = \frac{t}{\left(f + \frac{1-f}{P}\right)t} = \frac{1}{f + \frac{1-f}{P}}$$

حال اگر تعداد پردازنده‌ها را به سمت بی‌نهایت میل دهیم، حداکثر تسریع حاصل می‌شود:

$$s_{max} = \lim_{P \to \infty} \frac{1}{f + \frac{1-f}{P}} = \frac{1}{f}$$

یعنی تسریع از $\frac{1}{f}$ کمتر است.

به عنوان مثال اگر ۲۰٪ برنامه قابل موازی‌شدن نباشد یعنی $f = 0/2$ آنگاه حداکثر تسریع $\frac{1}{0/2} = 5$ می‌باشد. یعنی حتی اگر ۱۰۰ پردازنده نیز بخریم، برنامه ما حداکثر ۵ برابر سریع می‌شود! این مسئله به قانون آمدال معروف است.

تعریف بازدهی

پارامتر تسریع به تنهایی محک معتبری برای سنجش کامپیوتر پیشرفته نیست؛ چه بسا تسریع به ظاهر بالا باشد ولی به قیمت به کارگیری تعداد بسیار زیادی از منابع پردازشی. برای اینکه ببینیم پردازنده‌های سیستم به طور متوسط چقدر در طول برنامه فعال هستند از پارامتر بازدهی (Efficiency) استفاده می‌کنیم:

$$\text{بازدهی (E)} = \frac{s}{P} = \frac{\text{تسریع}}{\text{تعداد پردازنده}}$$

مثال: اگر برنامه‌ای روی یک پردازنده در ۱۰ ثانیه و روی ۵ پردازنده فقط در ۲/۵ ثانیه اجرا شود، آنگاه:

$$S(\text{تسریع}) = \frac{10}{2/5} = 4$$

$$E(\text{بازدهی}) = \frac{4}{5} = 0/8$$

هر چه بازدهی به یک نزدیک‌تر باشد، پردازنده‌ها بیشتر فعال بوده‌اند و تقسیم کار بهتر بوده است.

پایپلاین[1]

پایپلاین، تکنیکی است برای مبارزه با قانون آمدال! یعنی می‌توان با این تکنیک مؤلفه ترتیبی یک برنامه را تسریع کرد. در واقع پایپلاین تقسیم‌کردن یک عمل ترتیبی به چند واحد عملیاتی است.

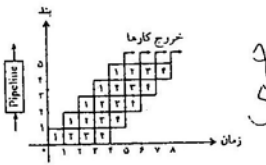
می‌توان عملیاتی که دارای نظم و ترتیب خاص، یا قابل پیش‌بینی هستند را توسط چندین واحد اجرایی به طور همزمان انجام و سرعت کل را افزایش داد.

پایپلاین را می‌توان به صورت چند طبقه[2] متوالی در نظر گرفت که داده‌ها وارد شده و هر بند یا طبقه عمل خاصی را روی داده‌ها انجام می‌دهد. این طبقات بطور همزمان فعال هستند. شکل (۱-۵) نمایانگر یک اجرای روی هم‌افتاده از ۵ واحد عملیاتی (۵ بند یا ۵ طبقه) پایپلاین است. شماره‌های هاشور زده نشانگر شماره دستورالعمل یا کار (task) واردشده در پایپلاین است.

1- Pipeline

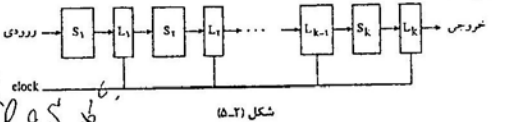
2- Stage , Segment

شکل (۱): در زمان ۱ کار شماره ۱ وارد بند ۱ شده
است و در زمان ۲ کار شماره ۲ وارد بند شماره ۱ شده
و کار شماره ۱ در بند ۲ می‌باشد و به همین ترتیب.



شکل (۱ـ۵)

پایپلاین را می‌توان به صورت بندهای پشت سر هم تصور کرد که این بندها مدارات ترکیبی هستند و هر کدام یک ثبات (یا لچ) دارند. در واقع داده‌ها در هر کلاک (کلاک‌ها به ثبات‌ها اعمال می‌شود) به ثبات‌ها وارد شده و عملیات مورد نظر روی آنها انجام می‌شود و در کلاک بعد نتیجه هر بند وارد ثبات بند بعد می‌شود. شکل (۲ـ۵) ساختار عمومی یک پایپلاین دارای k بند را نشان می‌دهد.



شکل (۲ـ۵)

فرض کنید ماکزیمم تاخیر انتشار در طبقات t_m باشد و زمان هر لچ برابر t_L.

در این صورت حداقل پریود کلاک برای کارکرد صحیح پایپلاین برابر است با: $T=t_m+t_L$ و فرض کنید که n تعداد کل کارهای اعمال شده باشد. در این صورت زمان اجرای این n کار در پایپلاین عبارت است از: $(k+(n-1))T$. (چون کار اول تمام k بند را طی می‌کند ولی n-۱ کار باقیمانده هر کدام تاخیر یک بند را متحمل می‌شوند زیرا در زمانی که مثلاً کار شماره ۱ به بند آخر رسیده است تمام k-۱ بند قبلی پر هستند).

حال اگر فرض کنیم بدون پایپلاین زمان اجرا برابر KTn است، (این زمان اجرا منطقی است زیرا بدون پایپلاین هر کار به طور جداگانه باید تمام مراحل را طی کند که زمان اجرای آن KT می‌باشد)، آنگاه تسریع برابر است:

$$S=\frac{\text{زمان اجرا بدون پایپلاین}}{\text{زمان اجرای پایپلاین}}=\frac{nKT}{[K+(n-1)]T}=\frac{nK}{K+(n-1)}$$

اگر تعداد کارها را به بی‌نهایت میل دهیم $(n\to\infty)$ حداکثر تسریع حاصل می‌شود: $S_{max}=K$

یعنی حداکثر تسریع برابر طول پایپلاین است. یعنی یک پایپلاین ۴ طبقه حداکثر ۴ برابر کار را تسریع می‌کند.

مثال: فرض کنید پایپلاین ۴ بند دارد که تاخیر بندها عبارتند از: ۶۰ns و ۵۰ns و ۹۰ns و ۸۰ns و همچنین هر لچ دارای تاخیر $t_L=10ns$ می‌باشد: آنگاه

پریود کلاک $T=\max(\text{تاخیر بندها})+t_L=90+10=100ns$

پایپلاین $t=[4+(n-1)]\times 100$

برای محاسبه زمان اجرا بدون پایپلاین، بهتر آن است که برای هر کار زمان بندها را جمع کرده و از لچ‌ها صرفنظر کنیم. زیرا در حالت بدون پایپلاین لچ‌ها نیاز نیستند پس

$t=60+50+90+80=280ns$ هر کار

بنابراین زمان اجرای n کار برابر است با ۲۸۰n

تسریع برابر است با:

$$S=\frac{280n}{[4+(n-1)]100}$$

اگر $n\to\infty$ حداکثر تسریع حاصل می‌شود:

$$S_{max}=\frac{280}{100}=2/8$$

یعنی حداکثر تسریع کمتر از ۴ (تعداد طبقات پایپلاین) می‌باشد.

بازدهی (Efficiency): برابر است با $E=\frac{S}{K}$ که S تسریع به دست آمده و K تعداد طبقات پایپلاین است. در مثال قبل حداکثر بازدهی برابر است با:

$$E_{max}=\frac{2/8}{4}=0/7$$

توجه ۱: اگر تمام طبقات دارای تاخیر انتشار مساوی باشند بهترین کارایی به دست می‌آید.

توجه ۲: هر چه تعداد طبقات بیشتر باشند به این معنی است که taskهای بیشتری در یک زمان پردازش می‌شوند.

توجه ۳: وقتی اولین task به آخرین بند رسید نتایج در هر کلاک حاصل می‌شوند.

طبقه‌بندی پایپلاین‌ها

۱ـ پایپلاین محاسباتی[1]: (پردازنده‌های Pentium-pro ، Powerpc ، Cray-۱ و ...)

۲ـ پایپلاین‌های دستور: (Pentium ، Intel ۴۸۶ و ...)

۳ـ پایپلاین‌های پردازنده: سری کردن پردازنده‌ها به طوری که هر یک وظیفه مشخصی را بر عهده دارد. این مورد با چند پردازنده متفاوت است؛ زیرا در چند پردازنده، هر پردازنده می‌تواند اعمال مختلفی انجام دهد، ولی در پایپلاین‌های پردازنده، هر پردازنده وظیفه مشخصی دارد.

پایپلاین محاسباتی

این نوع برای پیاده‌سازی عملیات ممیز شناور، ضرب اعداد ممیز ثابت، و سایر اعمال محاسباتی استفاده می‌شود.

فرض کنید بخواهیم دو عدد نرمال‌شده ممیز شناور $X=A\times 2^a$ و $Y=B\times 2^b$ را با هم جمع و تفریق کنیم.

A و B مانتیس و b و a نما هستند. می‌توان جمع و تفریق را در ۴ بند مطابق شکل (۳ـ۵) انجام داد.

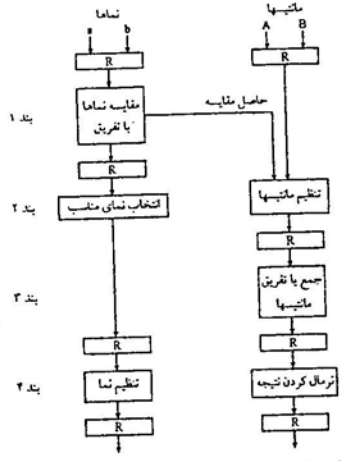
ثبات‌هایی که با R مشخص شده‌اند بین طبقات به منظور ذخیره نتایج میانی استفاده می‌شوند. عملیاتی که در این ۴ بند انجام می‌شود عبارت است از:

۱ـ مقایسهٔ نماها ۲ـ تنظیم مانتیس ۳ـ جمع و تفریق مانتیس‌ها ۴ـ نرمال کردن نتیجه.

نماها با کم‌شدن از یکدیگر مقایسه می‌شوند. تفاضل نماها نشان می‌دهد که مانتیس مربوط به نمای کوچکتر، چند واحد به سمت راست باید شیفت یابد. این عمل موجب مساوی‌شدن نماها و تنظیم مانتیس می‌شود. در بند ۳ مانتیس‌ها جمع یا

1- *Arithmetic pipelines*

تفریق می‌شوند. و در بند ۴ نتیجه نرمال می‌شود. اگر سرریز رخ دهد، مانتیس نتیجه، به راست شیفت می‌یابد و نمای آن یک واحد افزوده می‌شود. و اگر زیرریز رخ دهد، تعداد صفرهای سمت چپ مانتیس نشان می‌دهد که نتیجه چند واحد به چپ باید شیفت یابد.



شکل (۵ـ۳) پایپلاین برای جمع و تفریق ممیز شناور

تمام بندهای مذکور در شکل (۵ـ۳) را می‌توان با مدار ترکیبی طراحی نمود.

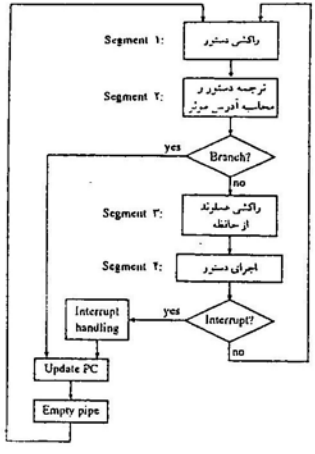
فرض کنید تاخیر هر بند به ترتیب برابر است با: $t_1=۶۰ns$ و $t_2=۷۰ns$ و $t_3=۱۰۰ns$ و $t_4=۸۰ns$ و تاخیر ثبات‌های میانی برابر است با $t_r=۱۰ns$. پریوکلاک باید برابر $t_p=t_3+t_r=۱۰۰+۱۰=۱۱۰ns$ باشد. یک جمع‌کننده / تفریق‌کننده بدون پایپلاین دارای تاخیری برابر $t_n=t_1+t_2+t_3+t_4=۲۲۰ns$ می‌باشد. (دقت کنید در این محاسبه یک ثبات در نظر گرفته شده است). که حداکثر تسریع برابر است با $\frac{۲۲۰}{۱۱۰}=۲/۹$. در واقع تسریع عبارت است از:

$$S=\frac{۳۲۰n}{[۴+(n-۱)]۱۱۰} \Rightarrow S_{max}=\frac{۳۲۰}{۱۱۰}$$

پایپلاین دستورالعمل

اجرای دستورات ترتیبی است (الگوریتم فن‌نیومن). ولی می‌توان زمانی که دستوری در حال واکشی است دستور دیگری را همزمان اجرا کرد، یعنی می‌توان مراحل الگوریتم فن‌نیومن را روی هم انداخت.

شکل (۵ـ۴) مثالی از یک پایپلاین دستور ۴ طبقه را نشان می‌دهد. طبقه اول دستور را از حافظه واکشی می‌کند، بند دوم دستور را رمزگشایی می‌کند و آدرس موثر را محاسبه می‌کند اگر دستور پرش نباشد، عملوندها از حافظه واکشی می‌شوند و دستور در بند ۴ اجرا می‌شود.



شکل (۵ـ۴)

ممکن است دستور پرش باشد که در این صورت اجرای دستورات موجود در خط لوله کامل می‌شود و سپس خط لوله خالی می‌شود و دستور جدید وارد خط لوله می‌شود. همچنین در صورت بروز و پذیرش وقفه، خط لوله خالی می‌شود. شکل (۵ـ۵)، عملکرد پایپلاین شکل (۵ـ۴) را نشان می‌دهد. زمان روی محور افقی مشخص شده است. چهار بند نمایش داده شده در شکل با علامت مختصر مشخص شده‌اند:

۱ـ FI بندی است که یک دستور را واکشی می‌کند.

۲ـ DA بند ترجمه دستور و محاسبه آدرس موثر است.

۳ـ FO مشخص‌کننده واکشی عملوند است.

۴ـ EX نیز بند اجرای دستور است.

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3	*		FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

شکل (۵ـ۵)

فرض بر آن است که پردازنده دارای دو حافظه جداگانه، یکی برای دستور و یکی برای داده است و بنابراین عملیات موجود در بندهای FO و FI می‌توانند همزمان اجرا شوند. اگر هیچ دستور پرشی وارد خط لوله نشود، تمام بندها همزمان مشغول هستند و هر یک روی یک دستور عملی را انجام می‌دهد. بنابراین در زمان ۲ (کلاک ۲)، دستور یک در حال اجراست، دستور ۲ در بند FO می‌باشد و دستور ۳ و دستور ۴ به ترتیب در بندهای DA و FI هستند.

فرض کنید که دستور ۲ پرش است. زمانی که این دستور در کلاک ۴ در بند DA رمزگشایی می‌شود، دستورات دیگر متوقف می‌شوند تا پرش انجام شود. اگر نتیجه پرش مثبت باشد، یک دستور جدید در کلاک ۷ وارد بند FI می‌شود ولی اگر نتیجه پرش منفی باشد (عدم پرش) همان دستور واکشی شده در کلاک ۴ می‌تواند استفاده شود.

عموماً، سه مشکل اساسی وجود دارد که موجب خروج پایپلاین از حالت نرمال می‌شود:

۱ـ تنازع بر سر منابع مشترک: اگر دو بند به طور همزمان احتیاج به حافظه داشته باشند باید یکی متوقف شود. برای حل این مشکل باید از چند واحد حافظه (حافظه‌های جداگانه برای دستور و داده‌ها استفاده کرد.) تنازع بر سر منابع دیگر نیز ممکن است رخ دهد.

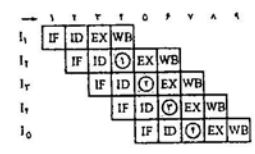
۲ـ وابستگی داده‌ای: زمانی رخ می‌دهد که دستوری به نتیجه دستور قبلی نیاز دارد بنابراین تا آماده شدن نتیجه باید صبر کند.

۳ـ مشکلات مربوط به پرش که در مثال بالا مشاهده کردیم.

مثال: با توجه به برنامه مقابل و با داشتن یک پایپلاین ۲ طبقه که شامل بندهای واکشی (IF)، ترجمه (ID)، اجرا (EX) و پس‌نویسی نتیجه (WB) می‌باشد زمان اجرا را محاسبه کنید؟

جواب) منظور از [DI] محلی از حافظه است که DI به آن اشاره می‌کند.

I_1	Mov	ax , bx
I_2	Mov	[DI] , ax
I_3	Add	SI , ۲
I_4	Add	DI , ۲
I_5	Dec	CX



تاخیر (۱) به خاطر وابستگی دستورات I_1 و I_2 است. ثبات ax در کلاک ۴ آماده می‌شود و دستور I_2 در کلاک ۵ می‌تواند آن را در اختیار داشته باشد و اجرا شود. تاخیرهای ۲ و ۳ و ۴ به خاطر آن است که بند اجرا مشغول است و یک واحد اجرایی بیشتر نداریم (تنازع بر سر منابع). بنابراین اجرای این دستورات ۹ کلاک طول می‌کشد. برای رفع مخاطره وابستگی، می‌توان چندین روش مطرح کرد.

یکی از این روش‌ها پیش‌فرستادن عملوند (operand forwarding) نام دارد. در این روش بین بندهای پایپلاین مسیرهای ویژه‌ای در نظر گرفته می‌شود. به عنوان مثال، در مثال قبلی اگر از این تکنیک استفاده شود تاخیر (۱) به وجود نخواهد آمد. زیرا نتیجه دستور I_1 که در بند WB مشخص می‌شود را می‌توان پیشاپیش به بند EX ارسال کرد تا دستور I_2 از آن استفاده کند.

تست‌ها

۱ـ در صورتی که Momory Word در برابر Instruction Word باشد چرخه دستورالعمل شامل Fetch ,Instruction Decode and Operand Fetch و Execution باشد و هر کدام در یک cycleانجام گیرد، و پردازنده از نوع segment pipelined - ۲ باشد، حداکثر توان عملیاتی (Max Throughput, d_0) چند برابر افزایش می یابد؟ (مهندسی کامپیوتر ۷۳)

الف) ۳ برابر　　ب) ۱/۵ برابر　　ج) ۲/۵ برابر　　د) ۲ برابر

۲ـ یک واحد محاسباتی لوله‌ای (Pipeline) دارای پنج قسمت (stage) با زمان اجرای ۲۶ns، ۳۹ns، ۲۲ns، ۲۸ns و ۴۲ns است. اگر از ثباتهایی با تأخیر یک ns در بین قسمت‌های مختلف لوله استفاده شده باشد. حداکثر تسریع این واحد محاسباتی نسبت به تأخیر غیر خط لوله‌ای چقدر است؟ (مهندسی کامپیوتر ۷۹)

الف) ۲/۹۲　　ب) ۳/۵　　ج) ۵　　د) ۷/۹۲

۳ـ در پردازنده‌ای با ساختار خط لوله (pipeline) دستورات در هشت مرحله (stage) اجراء می‌شوند. چنانچه دستوری از نوع پرش (Branch) باشد به دستورات بعد اجازهٔ ورود به خط لوله داده نمی‌شود تا اینکه اجراء دستور پرش به پایان برسد برنامه‌ای در حال اجراء است که ۱۰۰ دستور دارد و بعد از هر ۱۹ دستور معمولی یک دستور پرش در آن ظاهر می‌شود. اگر تأخیر هر مرحله و ثباتهای مربوط به آن جمعاً ۱۰ns باشد، اجراء این برنامه چقدر طول می‌کشد؟(بر حسب ns) (مهندسی کامپیوتر ۸۰)

الف) ۱۷۰۰　　ب) ۱۲۲۰　　ج) ۱۳۵۰　　د) ۱۰۷۰

۴ـ اگر یک خط لوله (Pipeline) سه ایستگاهی را به چهار ایستگاه تبدیل کنیم پریود ساعت از T به ۰/۹T کاهش می‌یابد فرض کنید ۳۰٪ دستورات پرش هستند. دستور بعد از دستور پرش وارد لوله نمی‌شود تا اینکه دستور پرش به اتمام برسد. نسبت زمان اجراء n دستور در ساختار سه ایستگاهی به ساختار چهار ایستگاهی چقدر است؟ (مهندسی کامپیوتر ۸۱)

الف) $\frac{۳}{۲/۶}$　　ب) $\frac{۱/۶}{۱/۷۱}$　　ج) $\frac{۲T+۱/۶nT}{۲/۶T+۱/۷۱nT}$　　د) $\frac{۲T+۱/۶nT}{۲/۷T+۱/۷۱nT}$

۵ـ در یک کامپیوتر که از روش pipeline برای اجرای دستورات استفاده می‌شود. هر دستور در سه مرحله واکشی (Fetch)، رمزگشایی (decode) و اجرا (execute) که هر کدام یک کلاک طول می‌کشند، انجام می‌شود. اگر در این کامپیوتر برای رفع مشکلات ناشی از pipeline در روش خالی کردن صف دستورات از پیش واکشی شده و روش delayed load وجود داشته باشد، اجرای برنامه زیر در هر کدام از این روشها چند کلاک طول می‌کشد؟ (علوم کامپیوتر ۷۹)

```
mov   ax, ۲
add   ax, ۲
jmp   next
```

فصل ششم

سازمان ورودی - خروجی

واسط ورودی - خروجی

واسط ورودی - خروجی، واسطه‌ای است که روشی برای انتقال داده‌ها بین حافظه داخلی سیستم و دستگاه‌های ورودی - خروجی تعبیه می‌نماید. منظور از قراردادن این واسط حل مشکل تفاوت‌های اساسی بین دستگاه‌های ورودی - خروجی و خود کامپیوتر می‌باشد. عمده این تفاوت‌ها عبارتند از:

۱ـ دستگاه‌های ورودی - خروجی (جانبی) عموماً دستگاه‌های الکترومکانیکی و یا الکترومغناطیسی هستند و یا حافظه و CPU که الکترونیکی هستند تفاوت دارند و بنابراین تبدیل مقادیر سیگنال‌ها در این ارتباط نیاز است.

۲ـ نرخ انتقال داده در دستگاه‌های جانبی کندتر از CPU و حافظه است و مکانیزمی برای همگامی آن‌ها نیاز است.

۳ـ کدها و قالب‌های داده‌ها در دستگاه‌های جانبی با قالب کلمات در CPU و حافظه متفاوتند.

۴ـ مدهای عملیاتی دستگاه‌های مختلف جانبی با یکدیگر متفاوت است و باید هر کدام توسط کنترل‌کنندهٔ خاص خود هدایت شوند. به سخت‌افزار مخصوصی که بین CPU و دستگاه‌های جانبی قرار می‌گیرد تا مشکلات فوق را حل کند «واسطه» یا Interface گفته می‌شود.

گذرگاه I/O

برای انتقال اطلاعات بین CPU و واسطها و به دنبال آن دستگاه‌های جانبی، خطوط داده و آدرس و خطوط کنترل وجود دارد. این مجموعهٔ خطوط در کنار یکدیگر «گذرگاه I/O»[1] را تشکیل می‌دهند. گذرگاه I/O، پردازنده را به همهٔ واسطها وصل می‌کند. برای ایجاد ارتباط با یک دستگاه جانبی، ابتدا پردازنده آدرس خاص آن را بر روی گذرگاه I/O (بخش آدرس) قرار می‌دهد. هر کدام از واسطها رمزگشای آدرسی دارند که تشخیص می‌دهند که آیا آدرس موجود بر روی خطوط آدرس مربوط به دستگاه خودشان است یا خیر و هر واسط که آدرس را از آن خود دانست، مسیر بین پردازنده و دستگاه‌های مربوط به خود را فعال می‌کند و بقیه واسطها غیرفعال می‌شوند.

فرمان I/O

در طول مدتی که آدرس خاص روی خطوط آدرس فعال است، CPU می‌تواند کدهایی بر روی خطوط کنترلی ارسال نماید که واسط، این کدها را گرفته و مسئول پاسخ‌دادن به آن‌ها می‌باشد. این کدها که عموماً توسط واسط باید اجرا شوند «فرمان I/O»[1] نامیده می‌شوند.

1- I/O command

چهار نوع فرمان I/O وجود دارد که واسط دریافت می‌کند:

الف) فرمان کنترلی: یک «فرمان کنترلی»[1] یک دستگاه جانبی را فعال می‌کند و به آن اطلاع می‌دهد که چه باید انجام دهد که این فرمان‌ها و ترتیب ارسال آن‌ها برای دستگاه‌های جانبی مختلف فرق می‌کند.

ب) فرمان وضعیت: یک «فرمان وضعیت»[2] برای بررسی حالت‌های مختلف یک واسط یا دستگاه جانبی به کار می‌رود. وضعیت واسط یا دستگاه جانبی معمولاً در ثباتی قرار داده می‌شود که این ثبات توسط پردازنده خوانده می‌شود.

ج) فرمان خروج داده‌ها: یک «فرمان خروج داده»[3] باعث می‌شود که واسط آنچه بر روی خطوط داده در گذرگاه I/O قرار دارد را به ثبات‌های خود منتقل نماید.

د) فرمان ورود داده‌ها: یک «فرمان ورود داده»[4] باعث می‌شود که واسط داده‌ای را که قبلاً از دستگاه جانبی دریافت نموده و در ثبات آن موجود می‌باشد بر روی خطوط داده از گذرگاه I/O قرار می‌دهد.

گذرگاه I/O و گذرگاه حافظه

پردازنده‌ها علاوه بر ارتباط با I/O نیاز به ارتباط با حافظه نیز دارند، گذرگاه حافظه نیز مانند I/O از خطوط داده، آدرس و کنترل تشکیل می‌شود. منظور از خطوط کنترلی، سیگنال‌های I/O Read و I/O Write برای خواندن از ورودی و نوشتن در خروجی و سیگنال‌های Memory Read و Memory Write برای خواندن از حافظه و نوشتن در حافظه می‌باشند. سیگنالهای دیگری نیز به عنوان سیگنال کنترلی مطرح می‌باشند مثل سیگنال‌های بررسی وجود خطا). گذرگاه I/O و گذرگاه حافظه به سه صورت می‌توانند در کنار یکدیگر واقع شوند:

۱ـ دو گذرگاه مجزا، یکی برای I/O و دیگری برای حافظه

۲ـ یک گذرگاه نیمه‌مشترک با خطوط داده و آدرس مشترک و خطوط کنترلی مجزا برای I/O و حافظه

۳ـ یک گذرگاه مشترک برای I/O و حافظه

گذرگاه مجزا

در این حالت عموماً برای کار با I/O، پردازندهٔ جداگانه‌ای تعبیه می‌شود، این پردازنده با I/O و حافظه ارتباط دارد و گذرگاه حافظه آن با گذرگاه حافظه CPU مشترک است. پردازندهٔ I/O گاهی «کانال داده»[5] نیز نامیده می‌شود.

گذرگاه نیمه‌مشترک (روش Isolated I/O)

هنگامی که CPU، آدرسی روی گذرگاه مشترک قرار می‌دهد با فعال نمودن I/O Read و یا I/O Write اعلام می‌نماید که آدرس، مربوط به واسطهای I/O می‌باشد و یا با فعال نمودن Memory Read و یا Memory Write مشخص می‌نماید که آدرس، مربوط به مکانی از حافظه است.

* **نکته:** در Isolated I/O، دستورات جداگانه‌ای از سوی CPU برای I/O و حافظه وجود دارند.

1- Control command
2- Status command
3- Output Data Command
4- Input Data command
5- Data channel

انواع ریزعملیات

ریز عملیات‌ها شامل ریز عملیات‌های انتقال، حسابی، منطقی و شیفت هستند. بعضی از این ریز عملیات‌ها در جدول (۱ـ۱) لیست شده‌اند.

جدول (۱ـ۱)

نوع	توصیف	ریزعملیات
ریزعملیات انتقال	انتقال R_1 به R_2	$R_2 \longleftarrow R_1$
ریزعملیات‌های حسابی	جمع R_1 و R_2 حاصل در R_2	$R_2 \longleftarrow R_1+R_2$
		$R_2 \longleftarrow R_1-R_2$
	مکمل ۱،[1] R_2	$R_2 \longleftarrow \overline{R}_2$
	مکمل ۲،[2] R_2	$R_2 \longleftarrow \overline{R}_2+1$
	R_1-R_2 حاصل در R_2	$R_2 \longleftarrow R_1+\overline{R}_2+1$
		$R_2 \longleftarrow R_2+1$
		$R_2 \longleftarrow R_2-1$
ریزعملیات‌های منطقی	and منطقی	$R_1 \longleftarrow R_1 \wedge R_2$
	or منطقی	$R_1 \longleftarrow R_1 \vee R_2$
	xor منطقی	$R_3 \longleftarrow R_1 \oplus R_2$
	مکمل ۱	$R_1 \longleftarrow \overline{R}_1$
	NAND منطقی	$R_3 \longleftarrow \overline{R_1 \wedge R_2}$
	NOR منطقی	$R_3 \longleftarrow \overline{R_1 \vee R_2}$
ریزعملیات‌های شیفت	شیفت چپ R	R ⟵ shlR
	شیفت راست R	R ⟵ shrR
	چرخش چپ R	R ⟵ cilR
	چرخش راست R	R ⟵ cirR
	شیفت حسابی R به چپ	R ⟵ ashlR
	شیفت حسابی R به راست	R ⟵ ashrR

1- 1's Complement 2- 2's Complement

با چند مثال عملیات شیفت را توضیح می‌دهیم.

shl (۱۰۱۱)=۰۱۱۰

shr (۱۰۱۱)= ۰۱۰۱

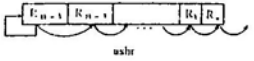
cil (۱۰۱۱)=۰۱۱۱

cir (۱۰۱۱)=۱۱۰۱

ashl (۱۰۱۱)=۰۱۱۰

ashr (۱۰۱۱)=۱۱۰۱

در ashr (شیفت حسابی) بیت علامت در جای خود مجدداً قرار می‌گیرد یعنی:



ashl و shl مانند هم عمل می‌کنند. ashl و ashr مخصوص اعداد علامت‌دار هستند.

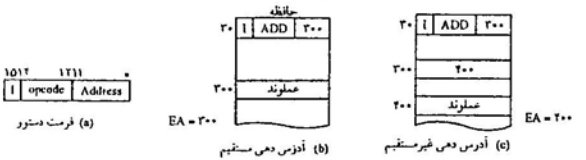
در ashl اگر بیت علامت عوض شود، خطای سرریز[1] رخ داده است این خطا را با V نشان می‌دهیم یعنی $V=R_{n-1} \oplus R_{n-2}$ (سرریز وقتی رخ می‌دهد که دو بیت سمت چپ ثبات یکسان نباشند. مثال بالا).

تمام ریزعملیات‌های گفته شده را می‌توان با گیتها[2] و مدارات منطقی، پیاده‌سازی کرد.

طراحی یک کامپیوتر پایه

می‌خواهیم یک کامپیوتر هر چند بسیار ساده ولی مفید برای درک بهتر معماری کامپیوتر طراحی کنیم.

کد دستورالعمل به همراه داده در حافظه ذخیره شده است. یک کد دستورالعمل تشکیل شده از تعدادی بیت که به کامپیوتر دستور می‌دهد تا یک عمل مشخص را انجام دهد. کد دستور معمولاً به چند فیلد تقسیم می‌شود، یکی از این فیلدها opcode است که نوع دستور را مشخص می‌کند. تعداد بیتهای opcode بستگی به تعداد دستورات ماشین دارد، اگر ماشین دارای 2^n یا کمتر دستور باشد، کد عملیاتی (opcode) باید n بیتی باشد. دقت کنید منظور ما از عمل همان کد عملیاتی دستور است که موجب اجرای چند ریزعمل می‌شود. فرمت دستور در ماشین پایه در شکل (a ۱ـ۵) نمایش داده شده است. همانطور که ملاحظه می‌شود، کد دستور دارای سه فیلد است، فیلد آدرس مشخص‌کننده محل عملوند است. این ماشین دارای یک فیلد آدرس است. پس یک عملوند برای آن می‌توان مشخص کرد، مثلاً می‌توان نوشت ADD A عملوند دوم عمل مثلاً جمع در یک ثبات همه منظوره به نام آکومولاتور (AC) ذخیره شده است. پس، دستور فوق یعنی از محل A در حافظه، داده را بردار و با محتویات AC جمع کن و نتیجه را در AC قرار بده. نتیجه تمام عملیات در AC ذخیره می‌شود.



(a) فرمت دستور (b) آدرس دهی مستقیم (c) آدرس دهی غیرمستقیم

شکل (۵ـ۱)

1- Overflow 2- Gate

گذرگاه مشترک (روش Memory-Mapped)

در این روش آنچه از تداخل آدرس‌ها جلوگیری می‌نماید و در مواقعی حافظه و در مواقع دیگر I/O را فعال می‌نماید جدانمودن آدرس‌ها است. در این حالت بخشی از کل فضای آدرس به حافظه و بخشی دیگر به ثبات‌های واسط‌های I/O منتسب می‌شوند. از این دیدگاه ثبات‌های واسط نیز به عنوان بخشی از سیستم حافظه در نظر گرفته می‌شوند.

* نکته: در روش Memory-Mapped I/O دستورات جداگانه‌ای برای I/O و حافظه وجود ندارد و فقط آدرس مشخص می‌نماید که هدف برقراری ارتباط با واسط یا حافظه است.

انتقال دادهٔ ناهمگام

عملیات‌های داخلی یک سیستم دیجیتال توسط پالس ساعتی که همهٔ اجزاء را تغذیه می‌نماید همگام می‌شوند. در انتقال داده اگر دو واحد نظیر CPU و I/O که مستقلاً طراحی می‌شوند، از یک پالس ساعت مشترک استفاده نمایند، «انتقال دادهٔ همگام» صورت خواهد پذیرفت، ولی در عموم حالات چنین امکانی وجود ندارد و هر واحد، پالس ساعت مخصوص به خود را دارا می‌باشد و انتقال داده بین دو واحد ناهمگام خواهد بود. در این حالت سیگنال‌هایی لازم است که بین دو واحد مستقل مبادله شوند و انتقال داده و یا آماده بودن آن را اعلام نمایند. مبادلهٔ این سیگنال‌ها به دو روش Strobe و Handshaking قابل انجام است.

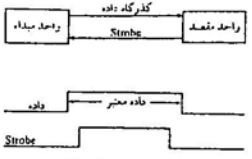
کنترل Strobe

در این روش تنها یک سیگنال کنترلی با نام Strobe برای هر انتقال لازم است. این سیگنال می‌تواند «فعال‌شونده توسط مبدأ» و یا «فعال‌شونده توسط مقصد» باشد.

الف) فعال‌شونده توسط مبدأ: همانگونه که در شکل (۱-۶) دیده می‌شود، مبدأ ابتدا داده را روی گذرگاه داده قرار می‌دهد بعد از مدتی ناچیز که از پایداری داده اطمینان حاصل شد، سیگنال Strobe را فعال می‌نماید، این سیگنال به همراه داده مدتی فعال می‌ماند تا مقصد فرصت کافی برای انتقال داشته باشد، البته عموماً مقصد از لبه پایین رونده سیگنال Strobe برای انتقال داده به ثبات‌های خود استفاده می‌نماید. مبدأ، سیگنال Strobe را غیر فعال نموده و بعد از مدتی داده را نیز از روی گذرگاه داده برمی‌دارد.

شکل (۱-۶) انتقال داده ناهمگام - سیگنال Strobe فعال‌شونده توسط مبدأ.

ب) فعال‌شونده توسط مقصد: این حالت نیز در شکل (۲-۶) مشاهده می‌شود. مقصد ابتدا توسط سیگنال Strobe از مبدأ درخواست داده می‌نماید، مبدأ نیز در پاسخ، داده را روی گذرگاه قرار می‌دهد. بعد از مدتی مقصد سیگنال Strobe را غیرفعال می‌نماید و در همان زمان می‌تواند داده را نیز به ثبات‌های داخلی خود انتقال دهد و سپس به دنبال آن مبدأ نیز داده را از روی گذرگاه برمی‌دارد.



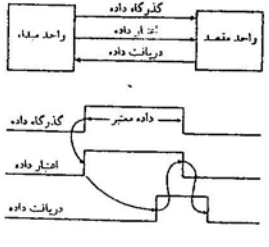
شکل (۲-۶) انتقال داده ناهمگام - سیگنال Strobe فعال‌شونده توسط مقصد.

مثال‌هایی از در حالت فوق می‌تواند عملیات خواندن و نوشتن بین CPU و حافظه باشد، در حالت اول CPU مبدأ و حافظه مقصد و سیگنال Strobe، سیگنال نوشتن (Write) می‌باشد و در حالت دوم CPU مقصد و حافظه مبدأ و سیگنال Strobe سیگنال خواندن (Read) می‌باشد.

Handshaking

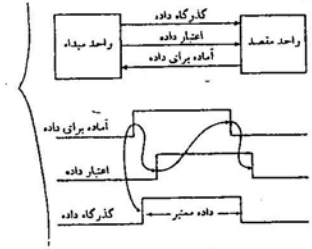
روش Strobe معایبی دارد که توسط Handshaking رفع می‌شود. از جمله این معایب این است که در حالت فعال‌شونده توسط مبدأ، مبدأ نمی‌تواند مطمئن باشد که مقصد داده‌ها را دریافت نموده و یا خیر و تنها با حدس این امر، داده و سیگنال Strobe را غیر فعال می‌نماید و یا در حالت فعال‌شونده توسط مقصد، مقصد نمی‌تواند اطمینان حاصل نماید که مبدأ، داده را به موقع روی خطوط گذرگاه قرار خواهد داد یا خیر. Handshaking با استفاده از سیگنال اضافی دیگری این مشکل را حل می‌نماید قابلیت اطمینان انتقال را بالاتر می‌برد. در Handshaking یک سیگنال کنترلی به همراه داده از مبدأ به مقصد ارسال می‌شود تا اعتبار داده را اعلام نماید و سیگنال دیگر در جهت معکوس از سوی مقصد به مبدأ اعلام می‌نماید که داده دریافت شده است. در این روش نیز دو حالت وجود دارد:

الف) شروع عملیات توسط مبدأ: در این حالت که در شکل (۳-۶) دیده می‌شود، مبدأ با قراردادن دادهٔ معتبر روی گذرگاه و فعال نمودن سیگنال «اعتبار داده» (Data Valid) عملیات را آغاز می‌نماید و منتظر می‌ماند تا مقصد سیگنالی بفرستد و توسط آن اعلام نماید که داده دریافت شده است. این سیگنال، «دریافت داده» (Data Accepted) نام دارد، سپس مبدأ می‌تواند دادهٔ معتبر و نیز سیگنال اعتبار داده را غیرفعال نماید و به دنبال آن مقصد نیز سیگنال دریافت داده را غیر فعال می‌کند و بعد از این مرحله است که دوباره دادهٔ جدید می‌تواند برای عملیات انتقال بعدی روی گذرگاه قرار داده شود.



شکل (۳-۶) Handshaking شروع عملیات توسط مبدأ

ب) شروع عملیات توسط مقصد: با توجه به شکل (۴-۲) مشاهده می‌شود که در این حالت مقصد توسط سیگنالی با نام «آماده برای داده» (Ready for Data) برای دریافت داده تقاضا و اعلام آمادگی می‌نماید. مبدأ نیز با دریافت این سیگنال دادهٔ معتبر را بر روی گذرگاه قرار داده و سیگنال اعتبار داده را فعال می‌نماید، در این هنگام داده برای مقصد آماده است. مقصد با دریافت داده سیگنال آماده برای داده را به نشانهٔ دریافت موفقیت‌آمیز داده، غیر فعال می‌کند و مبدأ نیز به دنبال آن سیگنال اعتبار داده و نیز دادهٔ روی گذرگاه را غیرفعال می‌نماید.



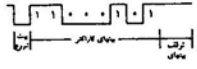
شکل (۴-۴) Handshaking شروع عملیات توسط مقصد.

هر کدام از سیگنال‌ها اگر به دلیل خرابی در مقصد یا مبدأ به تعویق بیفتد، بعد از مدت زمان از پیش تعیین‌شده‌ای توسط گیرندهٔ سیگنال، عدم دریافت آن اعلام شده و وقفه‌ای صورت می‌پذیرد که خطا را پوشش دهد. این خطا Time Out نامیده می‌شود.

انتقال سریال ناهمگام

در انتقال اطلاعات سریال همگام بین دو واحد فرستنده وگیرنده، پالس ساعت مشترکی وجود دارد که همگام بودن دو واحد را تضمین می‌نماید ولی در انتقال سریال ناهمگام این پالس مشترک وجود ندارد.

در این حالت داده‌ها کاراکتر به کاراکتر ارسال می‌شوند و در کنار هر کاراکتر بیت‌های کمکی ارسال می‌گردند تا شروع و پایان کاراکتر را مشخص نمایند. معمولاً قبل از ۸ بیت کاراکتر یک بیت شروع (start bit) و بعد از ۸ بیت کاراکتر یک یا بیش از یک بیت توقف (stop bit) ارسال می‌گردد. در حالت عادی خط توسط فرستنده در حالت یک نگهداشته می‌شود، به عنوان بیت شروع یک بیت صفر ارسال می‌شود که شروع کاراکتر را اعلام کند. آخرین بیت نیز که بعد از کاراکتر ارسال می‌شود و بیت توقف است یک می‌باشد و خط ارتباطی در حالت یک باقی می‌ماند. گیرنده نیز با دانستن این قوانین در بازه‌های زمانی مشخصی که به نرخ ارسال بیت‌ها بستگی دارد خط را بررسی می‌کند و بیت‌ها را می‌خواند. شکل (۴-۵) نمونه‌ای از انتقال سریال ناهمگام را نشان می‌دهد.



شکل (۴-۵) نمونه‌ای از انتقال سریال ناهمگام

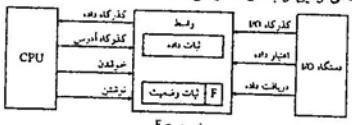
baudrate: نرخ انتقال اطلاعات سریال baudrate گفته می‌شود که معادل مقدار داده ارسالی بر حسب بیت در واحد زمان می‌باشد. و واحد بیت بر ثانیه ($\frac{bit}{s}$) می‌باشد.

حالت‌های انتقال

۱-ورودی - خروجی تحت کنترل برنامه (Programmed I/O)

در این حالت دستگاه ورودی - خروجی ارتباط مستقیمی با حافظه ندارد و هر گونه انتقال داده از دستگاه ورودی خروجی به حافظه اجرای یک یا چند دستور را توسط CPU نیاز دارد.

CPU از طریق واسط ورودی - خروجی به دستگاه ورودی - خروجی وصل می‌شود و در حقیقت CPU داده‌ها را از واسط می‌خواند و به واسط می‌نویسد و واسط خود با ایجاد ارتباط با دستگاه ورودی - خروجی داده را گرفته و یا می‌فرستد. شکل (۴-۶) نمونه‌ای از این ارتباط را نشان می‌دهد.



بیت پرچم = F

شکل (۴-۶) انتقال داده در حالت ورودی - خروجی تحت کنترل برنامه

در این ارتباط هر گاه دستگاه ورودی - خروجی داده‌ای آماده دارد بر روی گذرگاه قرار می‌دهد و سیگنال اعتبار داده (Data Valid) را فعال می‌نماید. واسط نیز داده را از روی گذرگاه به ثبات داده درون خود منتقل می‌نماید و سیگنال دریافت داده (Data Accepted) را فعال می‌نماید و نیز پرچمی در ثبات وضعیت که ثبات دیگری درون واسط است را نیز فعال می‌کند. CPU با خواندن ثبات وضعیت و بررسی پرچم نشان‌دهندهٔ آماده بودن داده، پی به وجود داده می‌برد و حتی داده را می‌خواند و پرچم مربوطه را غیر فعال می‌کند، با غیر فعال شدن این پرچم، واسط نیز سیگنال دریافت داده را غیرفعال می‌نماید تا به دستگاه اعلام نماید که دادهٔ جدید را می‌تواند ارسال نماید. در این روش، CPU به طور دائم در حال بررسی بیت پرچم می‌باشد تا حضور داده اعلام شود و این امر باعث هدر رفتن وقت CPU خواهد شد که از معایب این روش انتقال است.

۲-ورودی - خروجی وقفه‌دهنده (Interrupt Inititated I/O)

برای جلوگیری از اتلاف وقت CPU، به نظر می‌رسد که بهتر است به جای بررسی مداوم وجود دادهٔ آماده در واسط توسط CPU، واسط عهده‌دار این امر گردد که هنگام آماده‌شدن داده، به CPU وجود آن را اعلام نماید تا از بررسی‌های مکرر توسط CPU جلوگیری به عمل آید. در حالی که CPU در حال اجرای یک برنامه است، وقفه‌ای مبنی بر آماده بودن داده از سوی واسط دریافت می‌نماید. CPU اجرای عادی را موقتاً ترک کرده، آدرس دستور بعدی را ذخیره می‌نماید و اجرا را از آدرس برنامه سرویس‌دهندهٔ وقفه که همان برنامه سرویسی I/O می‌باشد ادامه می‌دهد و بعد از اتمام عملیات دریافت داده، اجرای عادی برنامهٔ قبلی را از آدرسی که هنگام دریافت وقفه ذخیره نمود بود، ادامه می‌دهد.

به دست آوردن آدرس برنامه سرویس ورودی - خروجی به دو صورت ممکن است: Vectored Interrupt و Nonvectored Interrupt.

الف) Nonvectored Interrupt: در این حالت آدرس پرش هنگام وقوع وقفه مکان ثابتی در حافظه است. و همه وقفه‌ها یک روتین وقفه مشترک دارند.

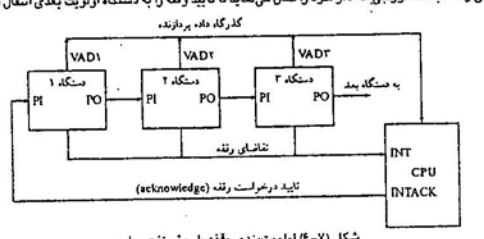
ب) Vectored Interrupt: منبع وقفه‌دهنده، خود اطلاعات مربوط به محل پرش را معین می‌نماید که این اطلاعات «بردار وقفه» را تشکیل می‌دهند.

در بعضی از کامپیوترها بردار وقفه آدرس شروع برنامه سرویسی I/O می‌باشد و در برخی دیگر بردار وقفه آدرس مکانی از حافظه است که آدرس شروع برنامه سرویسی در آن مکان واقع می‌باشد.

اولویت‌بندی وقفه: یک سیستم «اولویت‌بندی وقفه» سیستمی است که روی منابع متقاضی وقفه که همزمان درخواست وقفه می‌نمایند، ترتیبی برقرار می‌نماید و معین می‌نماید که کدام یک در ابتدا سرویس‌دهی شوند و نیز مشخص می‌نماید تحت چه شرایطی هنگام سرویس‌دهی به یک وقفه، وقفهٔ دیگری می‌تواند CPU را در اختیار بگیرد. اولویت‌بندی وقفه‌ها می‌تواند به روش‌های مختلف توسط نرم‌افزار و یا سخت‌افزار صورت گیرد. سه روش اولویت‌بندی وقفه عبارتند از:

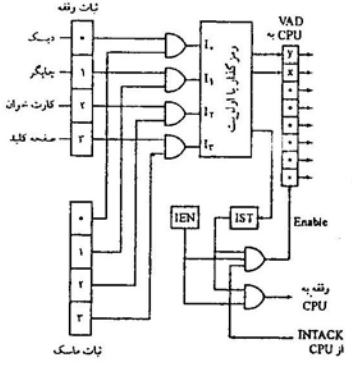
الف) سرکشی (Polling): در این روش که همان روش نرم‌افزاری است، برنامه‌ای که وقفه‌ها را در نظر دارد با هر وقفه به محلی ثابت از حافظه پرش می‌کند و به ترتیب اولویت، منابع وقفه را بررسی می‌کند و هر کدام وقفه را سبب شده باشد، برنامه سرویسی مربوط به آن اجرا می‌گردد. عیب این روش در زمان بررسی منابع است که ممکن است حتی از زمان سرویس به وقفه نیز بیشتر شود.

ب) زنجیره‌ای (Daisy-chain): این روش، یکی از روش‌های سخت‌افزاری است که عیب اتلاف زمان روش قبل را ندارد. در این روش دستگاه‌هایی که امکان تقاضای وقفه دارند بطور سریال به یکدیگر متصل می‌گردند. بطوری که اولویت‌های بالاتر در ابتدا و اولویت‌های پایین‌تر در انتها قرار می‌گیرند. شکل (۷-۶) چگونگی این ارتباط را نشان می‌دهد. هر دستگاه یک ورودی و سه خروجی دارد. خروجی اول، خط درخواست وقفه می‌باشد که به ورودی وقفه CPU متصل می‌باشد. CPU خروجی‌ای با نام «تأیید وقفه» (Interrupt acknowledge) دارد که به ورودی دستگاه بالاترین اولویت (ورودی PI) متصل می‌شود و خروجی PO از این دستگاه به ورودی PI دستگاه با اولویت بعدی وصل می‌گردد و به همین ترتیب همه دستگاه‌ها متصل می‌گردند. هر دستگاهی که درخواست وقفه نماید، CPU آن را با فعال نمودن خط تأیید وقفه تأیید می‌نماید. تا ورودی اولین PI فعال گردد، اگر دستگاه اول درخواست‌کنندهٔ وقفه باشد، در خروجی VAD خود که به گذرگاه داده CPU متصل است بردار وقفه را قرار می‌دهد و خروجی PO را غیر فعال نگه می‌دارد ولی در صورتی که این دستگاه متقاضی وقفه نباشد خروجی PO از خود را فعال می‌نماید تا تأیید وقفه را به دستگاه اولویت بعدی انتقال داده باشد.



شکل (۷-۶) اولویت‌بندی وقفه با روش زنجیره‌ای.

ج) موازی: این روش نیز مانند روش قبلی یک روش سخت‌افزاری است. در این روش ثباتی با نام «ثبات وقفه» وجود دارد که هر یک از بیت‌های آن یک درخواست وقفه را از جانب یک دستگاه جانبی اعلام می‌نمایند. ثبات دیگری با همین تعداد بیت در کنار این ثبات برای ماسک نمودن اثر هر یک از درخواست‌ها وجود دارد و برای این امر، خروجی در ثبات با یکدیگر AND می‌شوند و ورودی سیستم اولویت بند وقفه را تشکیل می‌دهند. شکل (۸-۶) این سخت‌افزار را برای ثبات‌های ۴ بیتی نمایش می‌دهد. خروجی حاصل از AND بیت‌های دو ثبات به جزئی با نام «رمزگذار اولویت» وارد می‌شوند. روش کار این رمزگذار بدین صورت است که از ورودی‌های I_0 تا I_3، به ترتیب اولین ورودی فعال (یک)، به عنوان تنها ورودی فعال این رمزگذار در نظر گرفته می‌شود تا خروجی دو بیتی آن را بسازد و این خروجی، به همراه ۴ بیت دیگر، بردار وقفه مربوط به آدرس برنامه سرویسی وقفه را ایجاد می‌نماید. رمزگذار اولویت خروجی دیگری نیز دارد که تنها در هنگام صفر بودن همهٔ ورودی‌ها صفر است و در غیر این صورت یک می‌باشد. این خروجی یک فلیپ‌فلاپ با نام IST را فعال و غیرفعال می‌نماید. فلیپ‌فلاپ دیگری با نام IEN نیز وجود دارد که در صورت داشتن مقدار یک، اجازه درخواست وقفه از CPU را می‌دهد و در صورت داشتن مقدار غیر فعال صفر، تغییر بیت‌های ثبات وقفه بی‌فایده است. با فرستادن درخواست وقفه به CPU و برگرداندن سیگنال تأیید (INTACK) از CPU بردار وقفه‌ای که توسط «رمزگذار وقفه» ساخته شده است، روی گذرگاه CPU قرار می‌گیرد و قابل استفاده برای CPU می‌باشد.



شکل (۸-۶) اولویت‌بندی وقفه به روش موازی.

۳ـ دسترسی مستقیم به حافظه (Direct Memory Access-DMA)

هنگام ارتباط CPU با I/O، همیشه لازم است که CPU برای انتقال داده‌های ورودی به حافظه و یا از حافظه به دستگاه‌های خروجی مدت زمانی را صرف کند که عموماً به دلیل کند بودن دستگاه‌های ورودی - خروجی، انتظار CPU به صرفه نیست و CPU می‌تواند در خلال آماده شدن داده‌ها به اجرای دستورات دیگر که نیاز فوری به این داده‌ها ندارند بپردازد و یا از دید دیگر در برخی موارد، داده‌هایی توسط دستگاه‌های ورودی آماده می‌شوند ولی CPU فرصت لازم برای انتقال بدون وقفه این اطلاعات را ندارد و در صورتی که این اطلاعات مستقیماً به حافظه منتقل شوند، داده‌های

ورودی دیگر می‌توانند توسط دستگاه ورودی آمادهٔ ورود گردند و این امر هنگام خروج داده‌ها به دستگاه‌های خروجی نیز اتفاق می‌افتد، در همهٔ موارد فوق نیاز به قسمتی در سیستم احساس می‌شود که فارغ از CPU ارتباط حافظه و I/O را فراهم آورد. این تکنیک DMA نامیده می‌شود و قسمتی که این انتقال را کنترل می‌کند «کنترل‌کنندهٔ DMA» نام دارد.

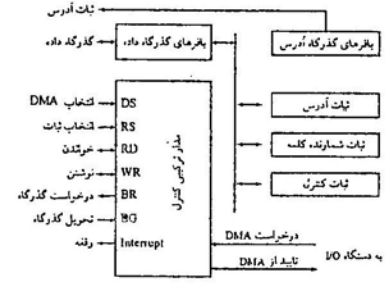
مهمترین تداخل CPU با کنترل‌کنندهٔ DMA، تداخل در کنترل گذرگاه داده و آدرس متصل به حافظه می‌باشد که در در لحظه فقط توسط یکی از آن دو قابل استفاده می‌باشد. CPU یک ورودی و یک خروجی برای حل این مسئله دارا می‌باشد، ورودی[1] BR به CPU کنترل گذرگاه‌ها را از CPU درخواست می‌نماید و CPU با بردن خروجی‌های آدرس و داده و نیز خطوط فعال‌کنندهٔ خواندن و نوشتن به حالت امپدانس بالا (شبیه مدار باز)، عملاً گذرگاه‌ها را در اختیار کنترل‌کنندهٔ DMA قرار می‌دهد و خروجی[2] BG خود را فعال می‌نماید تا این امر را به کنترل‌کنندهٔ DMA اعلام نماید. پس از اتمام کار کنترل‌کنندهٔ DMA، این کنترل‌کننده، خط درخواست گذرگاه (BR) را غیر فعال می‌نماید تا CPU به کار عادی خود ادامه دهد، با توجه به اینکه CPU در خلال کار کنترل‌کننده می‌تواند دستوراتی که به گذرگاه نیاز ندارند را اجرا نماید.

وقتی که کنترل‌کنندهٔ DMA، گذرگاه را در اختیار می‌گیرد به دو طریق می‌تواند انتقال داده‌ها بین حافظه و I/O را انجام دهد:

الف) انتقال توده‌ای: در این روش یک بلاک پیوسته از حافظه بدون توقف به I/O ارسال و یا از I/O دریافت می‌گردد و در مدت انتقال، گذرگاه‌ها در اختیار کنترل‌کنندهٔ DMA می‌ماند. این روش عموماً برای I/O سریع (مثل دیسک سخت) کاربرد دارد.

ب) سیکل Stealing: در این روش در هر مرتبه یک کلمه انتقال داده می‌شود و دوباره گذرگاه‌ها در اختیار CPU قرار می‌گیرند و CPU اجرای هر عملیات خود را به اندازهٔ یک سیکل حافظه به تأخیر می‌اندازد تا انتقال داده‌های I/O کامل گردد.

کنترل‌کنندهٔ DMA: کنترل‌کنندهٔ DMA علاوه بر اینکه نقش یک واسط را برای انتقال ورودی - خروجی دارد، اجزاء دیگری نظیر ثبات آدرس، ثبات شمارنده، ثبات کنترل و مجموعه‌ای از خطوط آدرس را نیز نیاز دارد. شکل (۹-۶) ساختار یک کنترل‌کنندهٔ DMA را نشان می‌دهد.



شکل (۹-۶) ساختار کنترل‌کنندهٔ DMA.

1- Bus Request 2- Bus Grant

کنترل‌کنندهٔ DMA علاوه بر ارتباط با حافظه و I/O با CPU نیز باید در ارتباط باشد تا مقداردهی اولیه در آن توسط CPU انجام گیرد، مثلاً ثبات آدرس در کنترل‌کنندهٔ DMA که آدرسی از حافظه می‌باشد که شروع انتقال از آنجا و یا به آنجا صورت می‌گیرد توسط CPU تعیین می‌شود و علاوه بر آن تعداد کلمات حافظه و نیز حالت و عملکرد کنترل‌کننده نیز به ترتیب توسط ثبات‌های شمارنده و کنترل معین می‌گردند که CPU همه را مقداردهی می‌نماید و به علاوه لازم است که CPU همه را مقداردهی نماید و به علاوه لازم است که CPU این اطلاعات را در مواقع لازم از DMA بخواند تا وضعیت کنترل‌کنندهٔ DMA را تحت کنترل داشته باشد.

کنترل‌کنندهٔ DMA همچنین دارای ورودی‌های DS (انتخاب DMA) و RS (انتخاب ثبات) می‌باشد که هر دو توسط خطوط آدرس CPU فعال و یا غیرفعال می‌شوند تا خود کنترل‌کننده و یکی از ثبات‌های آن را برای عملیات خواندن و نوشتن آماده نمایند.

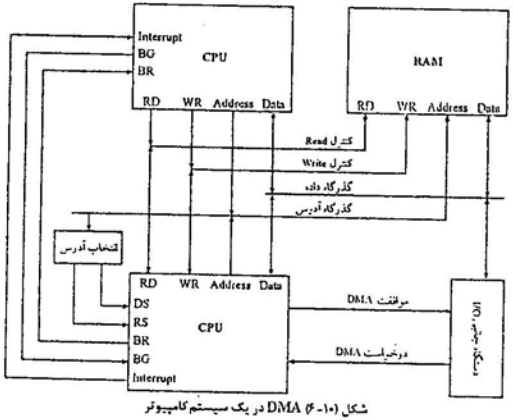
دو خط RD و WR مربوط به عملیات خواندن و نوشتن، خطوط در جهته در کنترل‌کنندهٔ DMA می‌باشند. هنگام کار CPU این خطوط ورودی‌های کنترل‌کننده می‌باشند ولی هنگام کار کنترل‌کننده این خطوط خروجی کنترل‌کننده و ورودی حافظه هستند.

خطوط BR و BG نیز همانگونه که قبلاً شرح داده شد، برای تحت اختیار گرفتن گذرگاه‌های آدرس و داده می‌باشند و خط وقفه نیز برای درخواست وقفه از CPU هنگام آماده‌بودن داده‌ها کاربرد دارد.

کنترل‌کنندهٔ DMA در خط دیگر نیز برای ارتباط Handshaking با دستگاه I/O دارد.

انتقال به روش DMA، CPU همانگونه که با دیگر واسط‌ها ارتباط برقرار می‌نماید. با کنترل‌کنندهٔ DMA که به اختصار DMA نامیده می‌شود نیز توسط خطوط داده و آدرس ارتباط برقرار می‌کند. DMA آدرسی دارد که خطوط DS و RS آن را فعال می‌کند. CPU از طریق گذرگاه داده مقادیر اولیه ثبات‌های DMA را معین می‌نماید و سپس با یک دستور کنترلی شروع عمل انتقال را از DMA درخواست می‌نماید. (این دستور کنترلی در ثبات کنترل DMA وارد می‌شود).

شکل (۱۰-۶) جایگاه DMA در سیستم کامپیوتر را نشان می‌دهد.



شکل (۱۰-۶) DMA در یک سیستم کامپیوتر

هنگامی که دستگاه جانبی تقاضایی را به DMA می‌فرستد (توسط خط درخواست DMA)، DMA نیز با فعال نمودن خروجی BR، از CPU تقاضای گذرگاه می‌نماید. CPU پاسخ مثبت خود را توسط خط BG اعلام می‌نماید. با این پاسخ DMA می‌فهمد که گذرگاه آزاد است و مقدار کنونی ثبات آدرس خود را بر روی گذرگاه آدرس قرار می‌دهد، خط WR و یا RD را فعال می‌نماید و به دستگاه جانبی اجابت درخواست را اعلام می‌نماید (توسط خط موافقت DMA).

توجه شود که خطوط RD و WR دو جهته هستند و این امر بستگی به حالت خط BG دارد. هنگامی که BG=۰ است، گذرگاه‌ها در اختیار CPU می‌باشند و این دو خط در DMA ورودی هستند و هنگامی که BG=۱ است و گذرگاه‌ها را DMA در اختیار دارد، این دو خط خروجی می‌باشند.

وقتی که دستگاه جانبی تأیید درخواست خود را دریافت نمود، کلمه‌ای را روی گذرگاه داده قرار می‌دهد و یا از گذرگاه دریافت می‌کند (بسته به عملیات ورودی یا خروجی) و داده مستقیماً ولی تحت کنترل DMA بین دستگاه جانبی و حافظه جابجا می‌گردد و DMA آدرس حافظه و حالت خطوط WR و DR را تحت کنترل خود دارد.

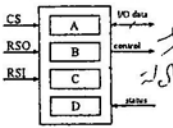
در کلمه که منتقل می‌گردد، ثبات آدرس DMA یک واحد کاهش و ثبات تعداد کلمات یک واحد کاهش می‌یابد. اگر در حالی که ثبات شمارنده به صفر نرسیده، داده‌های دستگاه I/O به اتمام رسیده و خط درخواست DMA غیر فعال گردید، DMA نیز خط BR را غیر فعال نموده و گذرگاه‌ها را در اختیار CPU قرار می‌دهد. ولی در صورت بالا بودن سرعت دستگاه جانبی به طوری که به محض اتمام اطلاعات انتقالی I/O، خط درخواست DMA از سوی دستگاه جانبی برای دسته دیگری از اطلاعات فعال باقی ماند، انتقال ادامه می‌یابد و گذرگاه همچنان در اختیار DMA می‌ماند.

در صورت صفر شدن ثبات شمارنده، DMA انتقال را متوقف می‌نماید و خط BR را غیر فعال می‌کند و توسط خط خروجی وقفه، به CPU اتمام انتقال را اعلام می‌کند. وقتی CPU تقاضای وقفه را دریافت نمود محتوای شمارنده را می‌خواند که مقدار صفر آن انتقال موفقیت‌آمیز کلمه‌ها را نشان می‌دهد این در حالی است که CPU در هر زمانی می‌تواند با خواندن این ثبات تعداد کلمات ارسال شده را بداند.

یک DMA می‌تواند چند کاناله باشد، که هر کانال به طور جداگانه یک دستگاه I/O را تحت پوشش قرار می‌دهد و دارای خطوط کنترلی جداگانه می‌باشند. هر کانال نیز ثبات‌های آدرس و شمارنده مخصوص به خود را دارد. آنچه مهم است اولویت کار کانال‌ها است که باید برنامه ریزی گردد و کانال‌های با اولویت بالاتر زودتر سرویس بگیرند.

از جمله کاربردهای DMA علاوه بر انتقال سریع اطلاعات از دیسک‌ها به حافظه، تازه کردن تصویر در ترمینال با استفاده از حافظه تصویر است. محتوای حافظه به طور متناوب به صفحهٔ نمایش ارسال می‌گردد و برای تغییر تصویر، حافظه تصویر تغییر داده می‌شود.

مثال: در شکل زیر، واسطی وجود دارد که یک خط انتخاب و در خط برای انتخاب یکی از ۲ ثبات موجود در خود دارد: برای اینکه آدرس این واسط، اعداد دودویی معادل ۱۲، ۱۳، ۱۴ و ۱۵ باشد:



۱) کم ارزش‌ترین خطوط آدرس CPU باید به RS_0 و RS_1 وصل گردند.
۲) تنها صفر بودن ۴ بیت پرارزش‌تر خطوط آدرس باید CS را فعال کند.
۳) کم ارزش‌ترین بیت خطوط آدرس به CS وصل می‌شود.
۴) ۱ و ۲ صحیح است.

جواب) گزینهٔ ۱ صحیح است. زیرا اعداد ۸ بیتی ۰۰۰۰۱۱۰۰ و ۰۰۰۰۱۱۰۱ و ۰۰۰۰۱۱۱۰ و ۰۰۰۰۱۱۱۱ آدرس‌های این واسط هستند که هر کدام نیز مربوط به یک ثبات می‌باشند. بنابراین دو بیت کم ارزشتر خط آدرس انتخاب کنندهٔ ثبات‌ها هستند و به RS_0 و RS_1 وصل می‌شوند و ۶ بیت با ارزش‌تر نیز انتخاب کنندهٔ واسط می‌باشند که با یک مدار ترکیبی نظیر رمزگشا CS فعال می‌نماید. جواب ۲ به این دلیل ناصحیح است که نه تنها صفر بودن ۴ بیت پرارزش خطوط آدرس بلکه یک بودن دو بیت بعدی نیز در انتخاب واسط و فعال شدن CS اثر دارد.

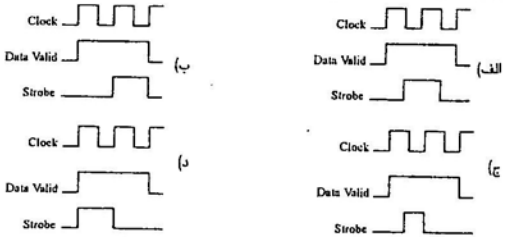
مثال: شش قطعه از واسط شکل مثال قبل در یک سیستم وصل شده‌اند، به گونه‌ای که ۶ خط پرارزش آدرس به ترتیب به ورودی CS شش قطعهٔ واسط وصل شده است و کم ارزش‌ترین بیت‌ها نیز به ورودی‌های RS_0 و RS_1 هر قطعه متصل است.

۱) H (۰۲) H،(۱۲) H،(۲۲) H،(۳۲) H،(۴۲) H،(۵۲) آدرس ثبات‌های شمارهٔ دو از ۶ واسط می‌باشند.
۲) H (۰۲) H،(۸۲) H،(۴۲) H،(۲۲) H،(۱۲) H،(۰A) آدرس ثبات‌های شمارهٔ دو از ۶ واسط می‌باشند.
۳) H (۸۲) H،(۴۲) H،(۲۲) H،(۱۲) H،(۰A) H،(۰۶) آدرس ثبات‌های شمارهٔ دو از ۶ واسط می‌باشند.
۴) اتصال بدین صورت باعث تداخل در آدرس‌ها خواهد شد.

جواب) گزینهٔ ۳ صحیح است. زیرا آدرس‌های متفاوت از حرکت عدد ۱ از سمت چپ‌ترین بیت تا سه بیت مانده به منتهی‌الیه سمت راست به دست می‌آیند و ثبات شمارهٔ دو نیز در حالتی که دو بیت راست ۱۰ باشد انتخاب می‌شود.

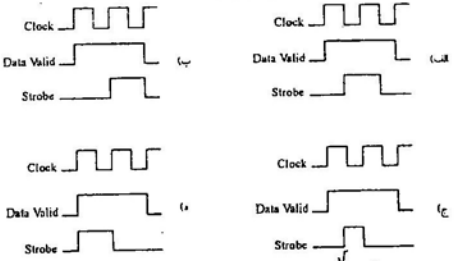
۱۰۰۰ ۰۰۱۰ → (۸۲) H
۰۱۰۰ ۰۰۱۰ → (۴۲) H
۰۰۱۰ ۰۰۱۰ → (۲۲) H
۰۰۰۱ ۰۰۱۰ → (۱۲) H
۰۰۰۰ ۱۰۱۰ → (۰A) H
۰۰۰۰ ۰۱۱۰ → (۰۶) H

مثال: یک CPU دارای پالس ساعت ۲۰ MHz می‌باشد که به حافظه با زمان دسترسی ۳۰ ns وصل شده است. کدام یک از گزینه‌های زیر عمل Write به حافظه در حالت Strobe را نشان می‌دهد؟



جواب) گزینهٔ ۱ صحیح است. پالس ساعت ۲۰ MHz دارای دورهٔ تناوب ۵۰ns می‌باشد و زمان یک پالس آن ۲۵ ns است. طول پالس Strobe باید ۴۰ns باشد تا انتقال به خوبی صورت پذیرد. در گزینهٔ ۳ طول پالس Strobe، ۲۵ ns است، در دو گزینهٔ نادرست دیگر طول پالس Strobe برابر ۵۰ns می‌باشد که حداقل ۴۰ ns را پوشش داده است ولی زمان‌های لازم در فاصله فعال شدن خطوط داده و فعال شدن Strobe و یا غیر فعال شدن آن دو را ندارند.

مثال: یک CPU دارای پالس ساعت ۲۰ MHz می‌باشد که به یک حافظه با زمان دسترسی ۴۰ns متصل است کدام یک از گزینه‌های زیر عمل Read از حافظه توسط CPU را در روش Strobe نشان می‌دهد؟



جواب) گزینهٔ ۲ صحیح است. در گزینهٔ ۱ پس از فعال شده دادهٔ معتبر تا غیر فعال شدن سیگنال Strobe تنها ۲۵ ns زمان وجود دارد که از زمان ۴۰ns بسیار کمتر است، در گزینهٔ ۳ با وجود طولانی‌تر بودن زمان اعتبار داده باز هم مشکل گزینهٔ ۴ نیز که با روش Strobe هماهنگی ندارد و سیگنال Strobe در ابتدا باید فعال شود و بعد از آن دادهٔ معتبر روی گذرگاه قرار داده شود. گزینهٔ ۲ مشکلی ندارد و زمان ۵۰ns برای دسترسی در نظر گرفته شده است.

مثال: در انتقال سریال بر روی یک خط با baud ۱۲۰۰ و کدهای کاراکتر ۸ بیتی:

۱) در حالت همگام تعداد کاراکتر ارسالی در واحد زمان ۳۰۰، در ناهمگام با دو بیت توقف $\frac{1200}{11}$ و با یک بیت توقف ۱۲۰ می‌باشد.

۲) در حالت همگام تعداد کاراکتر ارسالی در واحد زمان ۳۰۰، در ناهمگام با دو بیت توقف ۱۲۰ و با یک بیت توقف $\frac{120}{9}$ می‌باشد.

۳) در حالت همگام تعداد کاراکتر ارسالی در واحد زمان ۱۲۰، در ناهمگام با دو بیت توقف $\frac{1200}{9}$ و با یک بیت توقف ۱۲۰ می‌باشد.

۴) در حالت همگام تعداد کاراکتر ارسالی در واحد زمان ۱۲۰۰، در ناهمگام با دو بیت توقف ۳۰۰ و با یک بیت توقف ۱۲۰ می‌باشد.

جواب) گزینهٔ ۱ صحیح است. زیرا در حالت همگام بیت اضافی ارسال نمی‌گردد و نرخ ۱۲۰۰ بیت بر هر ثانیه همان ۳۰۰ کاراکتر بر ثانیه می‌باشد ولی در حالت ناهمگام با یک بیت توقف برای هر کاراکتر ۱۰ بیت (یک بیت شروع و یک بیت توقف) ارسال می‌گردد که نرخ ارسال را به ۱۲۰ کاراکتر بر ثانیه تبدیل خواهد نمود و در حالت دو بیت توقف برای هر

کاراکتر ۱۱ بیت ارسال می‌گردد که نرخ $\frac{1200}{11}$ کاراکتر در هر ثانیه را سبب می‌شود.

مثال: کدام یک از جملات زیر نادرست است؟

۱) مهمترین تفاوت دو حالت انتقال ورودی-خروجی تحت کنترل برنامه (Programmed-I/O) و ورودی-خروجی وقفه‌دهنده (Interrupt-Initiated I/O) در اتلاف زمان CPU است.

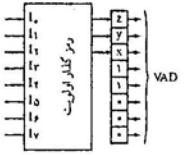
۲) تشخیص یک وقفه همیشه بعد از اجرای دستور جاری صورت می‌گیرد.

۳) روش‌های اولویت‌بندی تنها در حالتی که یک وقفه در حال سرویس باشد و وقفه‌ای دیگر از راه برسد عمل می‌کنند و نه در حالت همزمان تقاضای دو یا چند وقفه.

۴) Vectord Interrupt و Non Vectored Interrupt دو روش به دست آوردن آدرس برنامه سرویسی وقفه می‌باشند.

جواب) گزینهٔ ۲ صحیح است، زیرا روش‌های یاد شدهٔ اولویت‌بندی اعم از سخت‌افزاری و نرم‌افزاری هر دو حالت را پوشش می‌دهند.

مثال: بخشی از ساختار یک اولویت‌بند موازی با ۸ منبع وقفه به صورت شکل زیر می‌باشد.



با این «رمزگذار اولویت» بیت‌های x و y و z از بردار وقفه از کدام رابطه به دست می‌آیند، به شرط آنکه آدرس‌های حاصل از آن متوالی و شروع آدرس‌ها از 18H باشد؟ (A و B عبارات منطقی هستند)

۱) $x = I_0 I_1 I_2 I_3$, $y = I_0' B$, $z = I_0' A$

۲) $x = I_0' I_1' I_2' I_3'$, $y = I_0' B$, $z = I_0' A$

۳) $x = I_0 + I_1 + I_2 + I_3$, $y = I_0' + B$, $z = I_0' + A$

۴) $x = I_0' + I_1' + I_2' + I_3'$, $y = I_0' + B$, $z = I_0' + A$

جواب) گزینهٔ ۲ صحیح است. در رمز گذار اولویت اگر I_0 وقفه‌ای ارسال نماید ($I_0 = 1$)، صفر بودن و یا یک بودن بقیه ورودی‌ها همیتی ندارد و از آنجا که آدرس‌های VAD متوالی می‌باشند آدرس 18H مربوط به وقفه I_0 می‌باشد و در این آدرس x، y و z هر سه صفر هستند. از این توضیحات نتیجه می‌شود که گزینه‌ای صحیح است که با یک شدن I_0 در آن، صرف نظر از بقیه ورودی‌ها، x و y و z را صفر گرداند که گزینهٔ ۲ این خاصیت را دارد.

مثال: یک کنترل کنندهٔ DMA در هر سیکل Stealing کلمه‌ای ۱۶ بیتی را انتقال می‌دهد. دستگاه جانبی کاراکتر را با سرعت ۲۴۰۰ کاراکتر بر ثانیه آماده می‌نماید. CPU عملیات واکشی و اجرای دستورات را با سرعت ۱ میلیون دستور در

ثانیه انجام می‌دهد. به دلیل استفاده از این DMA:

۱) سرعت کار CPU ۲ برابر می‌شود.
۲) سرعت کار CPU نصف می‌شود.
۳) سرعت کار CPU برابر $\frac{1200}{10^6 + 1200}$ می‌شود.
۴) سرعت کار CPU $\frac{10^6 + 1200}{1200}$ برابر می‌شود.

جواب) گزینهٔ ۳ صحیح است. زیرا در این روش بعد از اجرای هر دستور، ۱۶ بیت توسط DMA منتقل می‌شود و دوباره نوبت به CPU می‌رسد تا دستور دیگری اجرا نماید. به نوعی می‌توان گفت هر دستور CPU به جای $\frac{1}{10^6}$ ثانیه، $\frac{1}{1200} + \frac{1}{10^6}$ ثانیه طول می‌کشد زیرا هر کاراکتر ۸ بیت است و سرعت در حقیقت ۱۲۰۰ کلمهٔ ۱۶ بیتی بر ثانیه می‌باشد.

تست‌ها

۱ـ عمل (Direct Memory Access (DMA در میکروپروسسورها به چه منظور است؟ (مهندسی کامپیوتر ۷۱)

الف) انتقال اطلاعات بین دستگاه‌های جانبی

ب) انتقال اطلاعات بین دستگاه جانبی و حافظه از طریق میکروپروسسور

ج) انتقال مستقیم اطلاعات از دستگاه جانبی به حافظه

د) هیچکدام

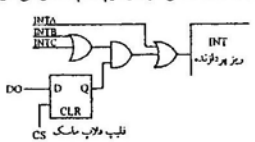
۲ـ شکل زیر نحوهٔ اتصال سیگنال‌های اینتراپت سه منبع وقفه به پایه اینتراپت ریزپردازنده را نشان می‌دهد. با توجه به شکل مدار، برای اینکه اولویت منبع A بیشتر از بقیه و منابع B و C الویت مساوی داشته باشند به چه صورت باید عمل شود: (مهندسی کامپیوتر ۷۱)

الف) در ابتدای برنامه سرویس A وقفه enable نمی‌شود و در برنامه سرویس B و C وقفه enable می‌شود و فلیپ‌فلاپ نیز صفر می‌شود.

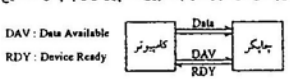
ب) در ابتدای برنامه سرویس A وقفه enable می‌شود، و فلیپ‌فلاپ صفر می‌شود. در برنامه سرویس B و C وقفه enable نمی‌شود.

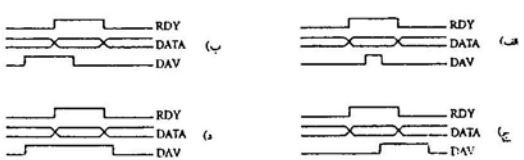
ج) در ابتدای برنامه سرویس A وقفه enable می‌شود، و در برنامه سرویس B و C وقفه enable می‌شود. و فلیپ‌فلاپ صفر می‌شود.

د) در ابتدای برنامه سرویس A وقفه enable می‌شود، و فلیپ‌فلاپ صفر می‌شود. در برنامه سرویس B و C وقفه enable می‌شود.



۳ـ کدام یک از سیگنالهای زمان‌بندی در مورد ارتباط بین کامپیوتر و چاپگر صحیح است؟ (مهندسی کامپیوتر ۷۶)





پاسخنامه تست‌ها

۱ـ گزینه (ج) صحیح است. از آنجا که انتقال اطلاعات بین یک وسیله ذخیره‌سازی سریع از قبیل دیسک مغناطیسی و حافظه اغلب سرعت cpu را محدود می‌کند، حذف cpu از مسیر انتقال و ایجاد این امکان که دستگاه جانبی مستقیماً باسهای حافظه را کنترل کند، سرعت انتقال را بالا می‌برد. به این روش DMA یعنی مراجعه مستقیم به حافظه گویند.

۲ـ گزینه (د) صحیح است. در ابتدای روتین سرویس‌دهنده وقفه A ، وقفه را enable می‌کنیم تا وقفه‌های B و C بتوانند تولید شوند ولی فلیپ‌فلاپ را صفر می‌کنیم تا این وقفه‌ها پذیرفته نشوند. زیرا وقفه A بیشترین اولویت را دارد. در برنامه سرویس‌دهنده وقفه‌های B و C ، وقفه را enable می‌کنیم تا اگر وقفه A صادر شد، بتوان ابتدا به وقفه A سرویس داد. زیرا اولویت آن بیشتر است.

۳ـ گزینهٔ (ج) درست است. این ارتباط یک ارتباط Handshaking شروع توسط مقصد می‌باشد. می‌توان این سناریو را در نظر گرفت که ابتدا چاپگر سیگنال RDY را بعنوان آمادگی برای دریافت فعال می‌نماید تا کامپیوتر با دیدن آن داده را روی خط گذاشته و سیگنال DAV را به نشان اعتبار داده فعال نماید. چاپگر با دریافت این سیگنال، داده را از روی خط برداشته و سپس سیگنال RDY را غیرفعال می‌نماید. این شرایط فقط توسط گزینهٔ ج تأمین می‌شود.

۴ـ گزینهٔ (د) صحیح است. پردازنده، دارای پایه‌های مربوط به درخواست و تایید DMA و همچنین پایه‌های وقفه نیست. پس وقفه‌دهی و DMA امکان‌پذیر نیست. از آنجا که باس آدرس مشترک است و ۱۶ بیت می‌باشد می‌توان بیش از ۲۵۶ درگاه ورودی / خروجی داشت.

۵ـ گزینهٔ (د) صحیح است.

۶ـ گزینهٔ (الف) صحیح است.

$$1Kbyte = 2^{10}byte = 2^{10} \times 11bit = 1024 \times 11bit = 11264bit \Rightarrow \frac{11264}{1200} \simeq 9/38 \text{ ثانیه}$$

۷ـ گزینهٔ (ج) صحیح است.

۸ـ گزینهٔ (ج) صحیح است.

۹ـ گزینهٔ (ج) صحیح است. این پردازنده امکان DMA ندارد چون دارای پایه‌های BR و BG نیست.

۱۰ـ گزینهٔ (ج) صحیح است.

۱۱ـ گزینهٔ (ج) صحیح است. بدون DMA زمان اجرای هر دستور $\frac{1}{10^7}$ ثانیه است و در هر اجرای دستور یک کاراکتر انتقال می‌یابد، هر کاراکتر در زمان $\frac{1}{2400}$ ثانیه انتقال می‌یابد. پس به طور متوسط هر دستور $\frac{1}{2400} + \frac{1}{10^7}$ ثانیه طول می‌کشد.

۱۲ـ گزینهٔ (د) صحیح است. در روش موازی از یک Priority Encoder استفاده می‌شود.

فصل هفتم

حافظه

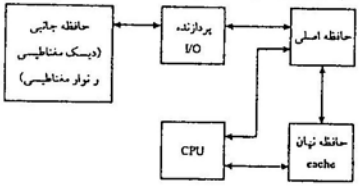
سلسله مراتب حافظه

در یک کامپیوتر انواع مختلفی حافظه وجود دارد که از لحاظ ظرفیت، سرعت، قیمت و نزدیکی به پردازنده با یکدیگر متفاوتند. هر چه حافظه سریعتر باشد، ظرفیت آن کمتر است و به پردازنده نزدیکتر است. حافظه با سرعت کمتر در مواقع خاص مورد استفاده قرار می‌گیرد.

سلسله مراتب حافظه به ترتیب سرعت عبارتند از: حافظه سریع نهان (Cache Memory)، حافظه اصلی (Main Memory) و حافظه جانبی (Auxilary Memory).

حافظه اصلی با CPU در ارتباط مستقیم است (البته از طریق حافظه نهان) و با حافظه جانبی از طریق یک پردازنده I/O مرتبط است. برنامه‌های در حال اجرا در حافظه اصلی قرار می‌گیرند ولی برنامه‌هایی که فعلاً به آنها نیاز نیست در حافظه کمکی مستقر می‌شوند.

به دلیل تفاوت سرعت CPU و حافظه اصلی (CPU نسبت به حافظه اصلی چندین برابر سریعتر است)، هنگام دسترسی به داده‌ها، گاهی CPU باید در حالت انتظار بماند تا حافظه، داده‌ها را آماده کند. برای رفع این مشکل، از حافظه‌های سریع‌تر (حافظه نهان) به عنوان حافظه‌ای که به CPU نزدیکتر است و رابط بین حافظه اصلی و CPU است استفاده می‌شود. شکل (۱ـ۷) چگونگی ارتباط را نشان می‌دهد.



شکل (۱ـ۷) CPU می‌تواند با حافظه اصلی بدون حافظه نهان، ارتباط برقرار کند.

حافظه اصلی

حافظه اصلی حافظه‌های سریع نسبتاً می‌باشند که دو نوع هستند: RAM و ROM

RAM (Random Access Memory) - حافظه با دسترسی تصادفی)

این نوع حافظه هم امکان خواندن و هم امکان نوشتن دارد و خود در نوع است:

الف) static (ایستا): اطلاعات در فلیپ‌فلاپ‌ها ذخیره می‌شوند و تا زمانیکه برق آن تامین باشد، اطلاعات آن معتبر می‌ماند.

ب) dynamic (پویا): اطلاعات به عنوان شارژ خازنی ذخیره می‌شوند. از آنجایی که شارژ خازن به مرور زمان از بین می‌رود، باید این نوع حافظه متناوباً تازه‌نویسی (Refresh) شود، یعنی خازن‌ها باید متناوباً شارژ شوند تا اطلاعات از بین نرود.

حافظه دینامیک مصرف توان کمتری دارد و ظرفیت بیشتر. حافظه استاتیک دارای سیکل خواندن و نوشتن کوتاهتری است زیرا احتیاج به تازه‌نویسی ندارد. حافظه استاتیک گرانتر است.

ROM (Read Only Memory - حافظه فقط خواندنی)

معمولاً اصطلاح حافظه اصلی را برای RAM به کار می‌برند ولی ROM نیز می‌تواند حافظه اصلی باشد. این حافظه نیز حافظه‌ای است با دستیابی تصادفی (مثل RAM) ولی فقط می‌توان اطلاعات را از آن خواند. این حافظه برای ذخیرهٔ برنامه‌هایی که به طور دائم باید در حافظه مقیم باشند، استفاده می‌شود. این حافظه به تغذیه سیستم وابسته نیست و با قطع تغذیه، اطلاعات آن از بین نمی‌رود.

در یک کامپیوتر، برنامه راه‌انداز سیستم را در این حافظه قرار می‌دهند تا هنگام روشن‌شدن سیستم این برنامه شروع به اجرا شده و کنترل سیستم را در اختیار سیستم‌عامل موجود روی دیسک قرار دهد.

ساختار بیرونی ROM و RAM

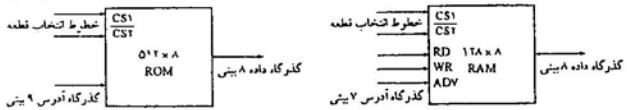
ورودی‌های این دو قطعه عموماً خطوط کنترلی، آدرس و داده می‌باشند و خروجی‌ها، خطوط داده هستند. از آنجا که RAM خاصیت خواندن و نوشتن را دارد ولی ROM تنها قابل خواندن است، خطوط داده در RAM دو جهته و خطوط دادهٔ ROM یک جهته می‌باشند. خطوط کنترلی ورودی، دو خط WR (نوشتن) و RD (خواندن) برای مشخص نمودن عمل حافظه و یک یا چند خط CS[1] (انتخاب قطعه) برای فعال نمودن قطعه می‌باشند. بافرهای خروجی حافظه‌ها معمولاً بصورت سه حالته ساخته می‌شوند، یعنی علاوه بر منطق صفر و یک، حالت امپدانس بالا را نیز دارا می‌باشند. هنگامی که دو خط RD و WR هر دو غیر فعال باشند و یا یکی از خطوط فعال‌کنندهٔ قطعه (CS)ها فعال نباشند، گذرگاه داده در حالت امپدانس بالا قرار می‌گیرد. استفاده از چند خط CS ترجمه خطوط آدرس را زمانی که چند تراشه وجود دارد، ساده می‌کند.

دو خط WR و RD گاهی به صورت یک خط با نام R/W ارائه می‌شوند که در دو منطق صفر و یک خواندن و یا نوشتن را مشخص می‌نمایند. حافظه‌های ROM چون فقط خواندنی هستند، خط کنترلی نوشتن و خواندن ندارند و عملیات حافظه از قبل مشخص است.

* نکته: هر خط کنترلی که با علامت بار مشخص گردد، نشان‌دهنده فعال صفر بودن آن است، یعنی حالت فعال آن در منطق صفر است.

1- Chip select

شکل (۲ـ۷) نمونه‌ای از دو حافظهٔ ROM و RAM را نشان می‌دهد. و جدول (۱ـ۷) چگونگی عملکرد تراشه RAM را بیان می‌کند.



(شکل ۲ـ۷) نمونه‌ای از حافظه‌های ROM و RAM.

جدول (۱ـ۷) عملکرد RAM

CS_1	$\overline{CS_2}$	RD	WR	Memory function
۰	۰	X	X	Inhibit (دسترسی غیر ممکن)
۰	۱	X	X	Inhibit (دسترسی غیر ممکن)
۱	۰	۰	۰	Inhibit (دسترسی غیر ممکن)
۱	۰	۰	۱	Write (عمل نوشتن)
۱	۰	۱	X	Read (عمل خواندن)
۱	۱	X	X	Inhibit (دسترسی غیر ممکن)

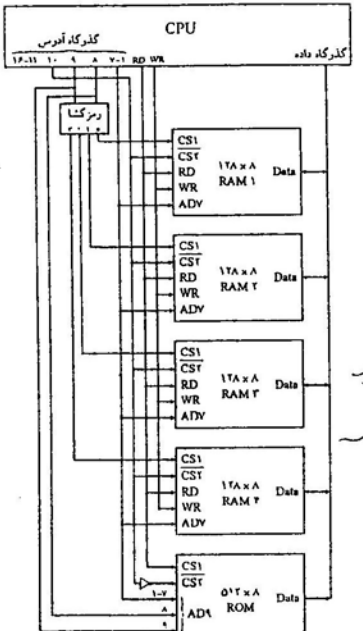
نقشهٔ آدرس حافظه

هر قطعه حافظهٔ RAM و یا ROM دارای ظرفیت محدودی است و طراح سیستم کامپیوتری باید تعدادی از قطعه‌های RAM و ROM را در کنار یکدیگر قرار دهد تا مجموع این قطعات، حافظهٔ لازم را در اختیار سیستم قرار دهد.

از آنجا که هر قطعه از حافظه RAM یا ROM از آدرس صفر (وقتی همهٔ خطوط گذرگاه آدرس صفر هستند) تا ۱ ـ 2^{Ad} که Ad تعداد خطوط آدرس است را می‌پذیرد و نظیر هر آدرس کلمه‌ای در خود ذخیره دارد، هنگام ترکیب این قطعات حافظه باید روشی پیاده شود تا آدرس‌های آن‌ها تداخل پیدا نکند زیرا در مجموع تنها یک سری خط آدرس به سیستم ترکیبی حافظه وارد می‌شود و یک سری خط داده نیز خارج می‌گردد، به گونه‌ای که آدرس صفر فقط کلمهٔ اول اولین قطعه حافظه را به بیرون ارسال نماید و آخرین آدرس نیز کلمهٔ آخر آخرین قطعه را روی گذرگاه داده منتقل نماید. جدولی که آدرس‌های پوشش داده شده توسط هر قطعه را مشخص می‌نماید، «نقشهٔ آدرس حافظه» (Memory Address Map) گفته می‌شود.

اتصال حافظه‌ها به CPU

خطوط گذرگاه داده در همه حافظه‌ها و CPU به هم متصل می‌باشند. خطوط کنترلی خروجی CPU نظیر RD و WR نیز می‌توانند مستقیماً خطوط نظیر در حافظه‌ها را به طور مشترک تغذیه نمایند. خطوط آدرس ارزش پائین‌تر CPU، عموماً برای انتخاب کلمه در هر حافظه و خطوط ارزش بالاتر برای انتخاب یک قطعه از حافظه به کار می‌روند. شکل (۳ـ۷) این اتصال را به همراه جزئیات نشان می‌دهد.



(شکل ۳-۷) اتصال چندین قطعه حافظه به یک CPU.

همانگونه که مشاهده می‌شود حافظه‌های RAM دارای ۷ خط ورودی آدرس می‌باشند و حافظه ROM دارای ۹ خط ورودی است. ۷ خط کم ارزش آدرس CPU همهٔ خطوط آدرس حافظه‌ها را تغذیه می‌نمایند و در مورد ROM دو خط دیگر نیز استفاده می‌شود. خطوط کنترلی RD و WR نیز به صورت متناظر متصل شده‌اند. خطوط بالارزش‌تر آدرس CPU برای انتخاب قطعه‌های حافظه استفاده می‌شوند. این خطوط در هر حالت خود، یکی از قطعه حافظه‌ها را فعال می‌نمایند تا خطوط کم‌ارزش‌تر آدرس نیز از قطعهٔ انتخاب‌شده کلمهٔ خاصی را انتخاب نماید. انتخاب هر قطعه توسط یک رمزگشا صورت می‌گیرد که با هر مقدار خطوط آدرس CPU یکی از خروجی‌های آن فعال می‌باشد. ورودی‌های انتخاب قطعه در هر حافظه می‌توانند از خروجی‌های این رمزگشا استفاده نمایند و در صورتی که قطعه‌ای بیش از یک خط انتخاب داشته باشد، می‌توان بقیه خطوط انتخاب را همیشه فعال نگهداشت یا مانند آنچه در شکل بالا مشاهده می‌شود به طور دائم به یکی از خطوط آدرس متصل نمود.

می‌توان با استفاده از مدار اتصال نقشه آدرس حافظه را به دست آورد و یا به عکس با استفاده از نقشهٔ آدرس حافظه اتصال‌ها را ترتیب داد. در مدار شکل بالا، RAM۱ در حالتی که خطوط ۸ و ۹ و ۱۰ آدرس CPU صفر باشند انتخاب می‌شود، پس این RAM آدرس‌های ۰۰۰۰ تا ۰۰۷F (با توجه به ۱۶ بیتی بودن خطوط CPU) را پشتیبانی می‌نماید. RAM۲ نیز در حالت صفر بودن خط ۹ و ۱۰ و یک بودن خط ۸ فعال می‌شود. این RAM نیز آدرس‌های ۰۰۸۰ تا ۰۰FF را پوشش می‌دهد و به همین ترتیب تا حافظهٔ ROM که از ۰۲۰۰ تا ۰۳FF را می‌پذیرد. جدول (۲-۷) نقشهٔ آدرس حافظه در اتصال شکل (۳-۷) را نشان می‌دهد.

(جدول ۲) نقشهٔ آدرس حافظه.

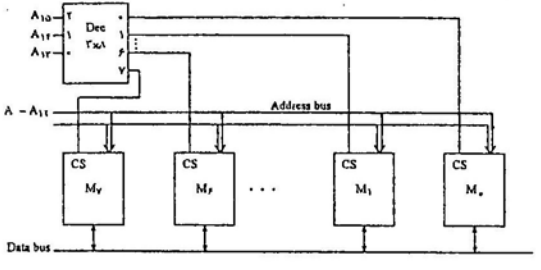
	محدودهٔ آدرس در مبنای ۱۶	خطوط آدرس CPU ۱۰ ۹ ۸ ۷ ۶ ۵ ۴ ۳ ۲ ۱ قطعه
RAM۱	۰۰۰-۰۰۷F	۰ ۰ ۰ × × × × × × ×
RAM۲	۰۰۸۰-۰۰FF	۰ ۰ ۱ × × × × × × ×
RAM۳	۰۱۰۰-۰۱۷F	۰ ۱ ۰ × × × × × × ×
RAM۴	۰۱۸۰-۰۱FF	۰ ۱ ۱ × × × × × × ×
ROM	۰۲۰۰-۰۳FF	۱ × × × × × × × × ×

همانطور که در شکل (۳-۷) ملاحظه کردید بیت‌های مرتبه بالای آدرس به منظور انتخاب هر تراشه دیکود شدند. این روش سنتی است و به آن High-order Interleaving می‌گویند.

اما روش بهتر آن است که بیت‌های مرتبه پایین آدرس را به منظور انتخاب هر تراشه دیکود کنیم، چرا که به این ترتیب آدرس‌های متوالی دیگر مانند روش قبل در یک تراشه قرار نمی‌گیرند بلکه در تراشه‌های متفاوت ظاهر می‌شوند؛ و این یک مزیت است زیرا می‌توان به عناصر پشت سر هم، مثلاً عناصر یک آرایه همزمان دسترسی داشت. به این روش Low-order Interleaving می‌گویند.

مثال: فرض کنید می‌خواهیم یک حافظه ۶۴KB را با تراشه‌های ۸KB بسازیم در این صورت می‌توان به دو روش عمل کرد.

روش اول: برای ۶۴KB به ۱۶ بیت خط آدرس نیاز داریم ($۶۴KB=۲^{۱۶}B$) با توجه شکل زیر می‌توان بیت‌های مرتبه بالا را برای انتخاب تراشه‌ها در نظر گرفت.



با توجه به شکل (۱.۵a) فیلد سوم (بیت شماره ۱۵) به نام I است. که مشخص‌کننده مستقیم[1] یا غیرمستقیم[2] بودن دستور است. در آدرس‌دهی مستقیم، فیلد آدرس مشخص‌کننده آدرس عملوند است. در این حالت به فیلد آدرس، آدرس موثر[3] می‌گویند. در حالت غیرمستقیم (I=1) فیلد آدرس: آدرس عملوند است پس با توجه به شکل (۱.۵c) در این حالت آدرس موثر (EA) برابر ۲۰۰ است نه ۳۰۰. در هر دو حالت محتویات محل آدرس موثر با AC جمع شده و نتیجه در AC ذخیره می‌شود. یعنی:

AC ← AC + عملوند

ثباتهای کامپیوتر

حافظه اصلی برای این کامپیوتر یک حافظه دارای ۲۰۹۶ کلمه ۱۶ بیتی است. ثباتهای مورد نیاز عبارتند از:

PC[4]: شمارنده برنامه که به دستور بعدی در حافظه برای اجرا اشاره می‌کند و چون آدرس را در خود ذخیره می‌کند باید ۱۲ بیتی باشد (کلمات حافظه $2^{12}=4096$ است که احتیاج به ۱۲ بیت آدرس دارد.)

DR[7]: ثبات داده است و عملوندی که از حافظه خوانده می‌شود را در خود نگه می‌دارد. به DR گاهی MDR[5] یا MBR[6] نیز می‌گویند این ثبات باید ۱۶ بیتی باشد.

IR[7]: ثبات دستور، دستوری که از حافظه خوانده می‌شود در این ثبات قرار می‌گیرد که ۱۶ بیتی است.

AC: ثبات همه منظوره است.

TR[8]: ثبات موقت است و برای داده‌های موقت استفاده می‌شود.

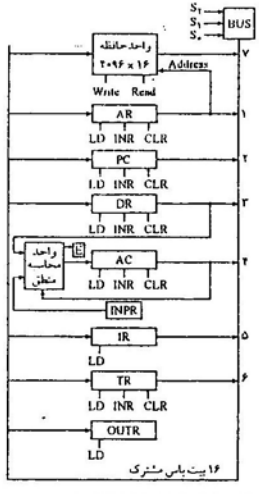
AR[9]: ثبات آدرس، نگهدارنده آدرس حافظه است (آدرس عملوند) و ۱۲ بیتی است.

ثباتهای INPR و OUTR نیز برای ورودی/خروجی استفاده می‌شوند و ۸ بیتی هستند.

کامپیوتر پایه دارای ۸ ثبات، یک واحد حافظه، و واحد کنترل است. برای ارتباط بین این واحدها باید مسیری وجود داشته باشد. شکل ۱.۶، باس مشترک را در این کامپیوتر نشان می‌دهد.

برای آنکه یک ثبات محتویات خود را روی باس قرار دهد باید انتخاب شود و برای آنکه محتویات باس به یک ثبات وارد شود باید ورودی بار (LD) آن ثبات فعال شود. برای آنکه حافظه مقداری روی باس قرار دهد باید انتخاب شده ($S_2S_1S_0=111$) و فرمان Read داده شده و آدرس توسط AR مشخص شود. و برای آنکه محتویات باس به حافظه منتقل شود باید write فعال شده و آدرس توسط AR مشخص شود.

1- Direct
2- Indirect
3- Effective Address
4- Data Register
5- Memory Data Register
6- Memory Buffer Register
7- Instruction Register
8- Temprory Register
9- Address Register



محتویات باس	$S_2S_1S_0$
AR	۰۰۱
PC	۰۱۰
DR	۰۱۱
AC	۱۰۰
IR	۱۰۱
TR	۱۱۰
Momory	۱۱۱

شکل (۱.۶) تمام ثباتها دارای کلاک نیز هستند

ثباتهای AC، DR، IR و TR هر کدام ۱۶ بیتی هستند ولی ثباتهای PC و AR، ۱۲ بیتی هستند. بنابراین هنگامی که PC و AR محتویات خود را روی باس می‌گذارند، چون باس ۱۶ بیتی است، ۴ بیت بالای باس صفر می‌شود. همچنین هنگامی که محتویات باس به PC و AR منتقل می‌شود فقط ۱۲ بیت مرتبه پایین منتقل می‌شود.

ورودی AC از یک مدار محاسباتی و منطقی می‌آید. ورودیهای این مدار عبارتند از DR و INPR و AC. مثلاً حاصلجمع DR و AC قابل محاسبه است و حاصل در AC و رقم نقلی در فلیپ‌فلاپ E قرار می‌گیرد.

در این ساختار، می‌توان همزمان بعضی عملیات را انجام داد. به عنوان مثال، اگر $S_2S_1S_0=100$ محتوای AC روی باس قرار می‌گیرد و با فعال کردن LD ثبات DR و همزمان انتقال محتوای DR از طریق مدار عملیات حسابی و منطقی به AC و فعال کردن LD ثبات AC، می‌توان در یک پالس ساعت DR ← AC، AC ← DR را انجام داد.

دستورات کامپیوتر پایه

کد عملیاتی دستور در این کامپیوتر ۳ بیتی است و ۱ بیت نیز برای نوع آدرس‌دهی (مستقیم یا غیرمستقیم) در نظر گرفته شده است. دستورات این کامپیوتر به سه دسته تقسیم می‌شوند: ۱ـ دستورات حافظه‌ای ۲ـ دستورات ثباتی ۳ـ دستورات ورودی/خروجی. جدول ۱.۲ دستورات ماشین را نشان می‌دهد.

که در این صورت عناصر یک آرایه که عناصر متوالی هستند همگی مثلاً در تراشه M_0 قرار می‌گیرند و امکان دسترسی همزمان به آنها نیست.

روش دوم: به جای $A_{۱۳}$ و $A_{۱۴}$ و $A_{۱۵}$ برای انتخاب تراشه‌ها از بیت‌های مرتبه پایین یعنی A_0 و A_1 و A_2 استفاده کنیم که با این روش می‌توان همزمان به عناصر متوالی دسترسی داشت. زیرا در این صورت آدرس‌های متوالی در یک تراشه قرار ندارند و در تراشه‌های متوالی هستند. مثلاً آدرس صفر در m_0، آدرس یک در m_1 و به همین ترتیب آدرسهای متوالی در تراشه‌های متوالی قرار دارند.

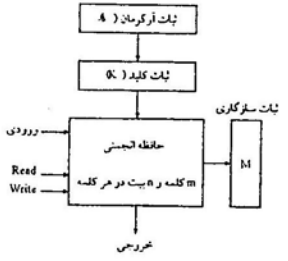
پهنای باند حافظه

برابر است با تعداد کلمه‌ای که می‌توان در ثانیه (از / به حافظه) منتقل کرد. مثلاً اگر کلاک باس ۵۰۰MHZ باشد و عرض باس ۳۲bit و در هر کلاک یک دسترسی به حافظه داشته باشیم و در هر دسترسی بتوانیم ۴ بایت از حافظه بخوانیم یا بنویسیم در این صورت پهنای باند (BW) برابر است با:

$$BW = ۱ \times \frac{۴byte}{۲ns} = \frac{۲GByte}{s}$$ (توجه کنید که ۵۰۰MHZ معادل ۲^{ns} است.)

حافظه انجمنی (Associative Memory)

این حافظه ساختاری دارد که برای ذخیرهٔ جداولی که در آینده عناصر آن‌ها قابل جستجو می‌باشند به کار می‌رود. در این نوع حافظه به جای آدرس ورودی برای دسترسی به عناصر حافظه، خود محتوای مکان حافظه برای دسترسی استفاده می‌گردد. این نوع حافظه علاوه بر حافظه انجمنی، «حافظه قابل دسترسی توسط محتوا» (Content Access Memory-CAM) نیز نامیده می‌شود. نوشتن در این حافظه نیاز به آدرس ندارد و خود حافظه ساختاری دارد که دادهٔ ورودی را در اولین مکان خالی حافظه وارد می‌نماید و خواندن از این حافظه نیز با تصدیق یک کلمهٔ ورودی (یا بخشی از آن) با کلیهٔ کلمات مکان‌های حافظه (یا بخشی از آن‌ها) صورت می‌پذیرد و مکان‌هایی از حافظه که مطابق کلمهٔ ورودی باشند برای خواندن مشخص می‌شوند. یک حافظه CAM علاوه بر داشتن سلول‌های ذخیرهٔ بیت، مانند حافظه‌های RAM، دارای مدارات منطقی تطبیق نیز می‌باشد. شمای کلی یک حافظه CAM در شکل (۴-۷) آمده است. در این حافظه m کلمهٔ n بیتی وجود دارد. در این حافظه علاوه بر خطوط کنترلی خواندن و نوشتن و خطوط دادهٔ ورودی و خروجی، ورودی دیگری نیز وجود دارد. این ورودی کلمه‌ای است که می‌باید با تک‌تک کلمات حافظه مقایسه شود. این کلمه در ثباتی با نام «ثبات آرگومان» قرار داد. ثبات دیگری نیز وجود دارد که عمل ماسک نمودن مقایسه بعضی بیت‌ها را انجام می‌دهد که به «ثبات کلید» معروف است. بدین معنی که مقایسه تنها در مورد بیت‌هایی صورت می‌پذیرد که متناظر با آن‌ها در ثبات کلید، بیت یک وجود داشته باشد. بعد از مقایسه، در ثبات سازگاری بیت متناظر در کلمه با ردیف که با کلمهٔ ورودی مطابق باشد یک می‌شود و در صورت عدم تطابق کامل، بیت مربوطه صفر می‌گردد.

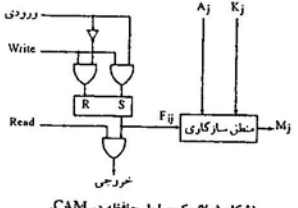


(شکل ۴-۷) شمای کلی حافظهٔ CAM.

شکل (۴-۷) نیز سخت‌افزار مربوط به یک سلول حافظهٔ CAM را نشان می‌دهد این تک سلول دارای مدار منطقی برای تطابق بیت‌ها می‌باشد که سه ورودی A، K و F دارد. F بیت ذخیره شده (از یک کلمهٔ حافظه)، A یک بیت از کلمه ورودی و K نیز بیت مربوط به عمل ماسک می‌باشد. این بیت در صورت صفر بودن، عمل مقایسه را غیر فعال می‌کند به صورتی که مقایسهٔ بیت‌های متناظر از کلمهٔ ورودی و کلمهٔ موجود در حافظه در نتیجهٔ مقایسه اثری ندارد و در صورت یک بودن بیت یادشده عمل مقایسه F و A انجام می‌گیرد و اگر یکسان باشند خروجی مدار (M) یک خواهد بود.

$$M=AF+A'F'+K'$$

برای کلمهٔ حافظه در CAM، خروجی‌های M از هر سلول تشکیل‌دهنده کلمه حافظه با یکدیگر AND می‌شوند تا خروجی نهایی حاصل از تطبیق یک ردیف را تولید نمایند.



(شکل ۵-۷) یک سلول حافظه در CAM.

عمل خواندن: با اعمال کلمهٔ مورد جستجو به حافظه، خروجی مربوط به مکانی از حافظه که با کلمهٔ ورودی یا بخشی از آن یکسان است یک می‌گردد و البته اگر چند مکان حافظه این خاصیت را داشته باشند، چندین خروجی ممکن است یک شوند و کلماتی که خروجی تطابق آن‌ها یک باشد، خوانده می‌شوند، البته در صورتی که خروجی M به ورودی کنترلی خواندن حافظه وصل شود، با پیدا شدن عنصری خاص همزمان خواندن آن نیز انجام می‌گیرد با شرط اینکه این تطابق تنها در مورد یک ردیف یا کلمهٔ حافظه وجود داشته باشد.

عمل نوشتن: عمل نوشتن در این حافظه نیز می‌تواند بعد از یک عملیات جستجو صورت گیرد. خالی بودن و یا پر بودن یک کلمهٔ حافظه می‌تواند توسط ثباتی معین شود که بیت خاصی از این ثبات در صورت صفر بودن نشانگر خالی بودن و در صورت یک بودن نشان‌دهندهٔ پربودن مکان حافظه است. از این ثبات که برای هر کلمهٔ حافظه به طور جداگانه وجود دارد،

تحت عنوان tag یاد می‌شود.

در عمل نوشتن ابتدا ثبات‌های tag بررسی می‌شوند تا با رسیدن به اولین مکان خالی، عمل نوشتن صورت گیرد. برای حذف یک کلمه نیز تنها لازم است بیت خاص ثبات tag مربوط به آن کلمه صفر گردد.

حافظه نهان (Cache memory)

سریع‌ترین نوع حافظه در سلسله مراتب حافظه است که سرعت دسترسی به داده‌ها را زیاد می‌کند.

محلی بودن مراجعات (Locality of Reference): مهمترین علتی که استفاده از حافظه‌های سریع را توجیه می‌نماید محلی بودن مراجعات به حافظه توسط CPU می‌باشد. محلی بودن مراجعات به این معنی است که عموماً داده‌هایی که به صورت پی‌درپی توسط CPU از حافظه خوانده می‌شوند در محدوده‌ای کوچک از حافظه قرار دارند. مثلاً داده‌های یک آرایه و یا دستورات مربوط به یک حلقه دو برنامه و یا دستورات یک برنامه فرعی و ... همه پشت سر هم در حافظه واقع می‌شوند و در صورتی که مجموعه‌ای از داده‌های نزدیک به هم و یا دستورات نزدیک به هم به حافظه سریع منتقل شوند می‌توان احتمال قوی داد که چند مراجعه بعدی CPU به جای حافظه اصلی، به حافظه سریع خواهد بود.

در واقع محلی بودن مراجعات از یک جنبه دیگر نیز مطرح است و آن مراجعات زمانی است، یعنی اگر اکنون (در زمان t_1) به داده I دسترسی داریم به احتمال زیاد در آینده نزدیک به این داده مجدداً نیاز خواهیم داشت، پس بهتر است آن را درون حافظه نهان نگاهداریم، مثل حلقه‌ها که داده‌ها و دستورات آن متناوباً مورد نیاز هستند.

عملیات حافظه نهان: وقتی CPU نیاز به دسترسی به حافظه دارد، در ابتدا حافظه نهان بررسی می‌شود، اگر کلمه مورد نظر در حافظه نهان یافت شود، از آن خوانده می‌شود ولی اگر کلمه در حافظه نهان یافت نشد، دسترسی به حافظه اصلی صورت می‌گیرد و در این دسترسی علاوه بر کلمهٔ موردنظر، چندین کلمهٔ اطراف آن نیز به حافظهٔ نهان منتقل می‌گردند. این بلوک قابل انتقال می‌تواند از یک کلمه تا چندین کلمه باشد. کلماتی که علاوه بر کلمهٔ اصلی به حافظهٔ نهان منتقل می‌گردند، احتمالاً در مراجعات آتی CPU استفاده می‌گردند. تعداد کلماتی که به حافظه نهان منتقل می‌شوند و چگونگی قرار گرفتن آنها در حافظه نهان به ساختار حافظه‌ها و پیاده‌سازی الگوریتم‌های مختلف بستگی دارد.

کارآیی حافظه نهان: کارآیی حافظه نهان توسط پارامتری با نام «نرخ hit» اندازه‌گیری می‌شود. هنگام مراجعه به حافظه نهان در صورتی که کلمهٔ مورد نظر در حافظه نهان موجود باشد یک hit صورت پذیرفته است و در صورت عدم وجود کلمه در حافظه نهان و لزوم مراجعه به حافظه اصلی یک miss واقع شده است. تعداد hit‌ها در یک محدودهٔ زمانی به تعداد کل مراجعات (مجموع hit‌ها و miss‌ها) نرخ برخورد[1] گفته می‌شود.

یعنی می‌توان نوشت:

$$h\ (\text{نرخ برخورد}) = \frac{\text{تعداد مراجعات موفق به حافظه نهان}}{\text{تعداد کل مراجعات}} = \frac{\text{تعداد hit}}{\text{تعداد (hit+miss)}}$$

به عنوان مثال اگر پردازنده ۱۰۰ بار به حافظه نهان رجوع کند و ۹۵ بار داده مورد نظر در حافظه نهان موجود باشد و ۵ بار داده در حافظه نهان نباشد در این صورت نرخ برخورد h=٪۹۵ می‌باشد. در واقع نرخ برخورد حافظه نهان (h)

1- hit ratio

احتمال حضور داده مورد نظر در حافظه نهان است.

نرخ عدم برخورد یا نقصان (miss ratio) برابر ۱-h می‌باشد. زمان متوسط دسترسی به حافظه (t_a) برابر است با:

$$t_a = t_c + (1-h)t_m \qquad (1)$$

که t_c زمان دسترسی به حافظه نهان و t_m زمان دسترسی به حافظه اصلی است.

فرمول فوق با این فرض به دست آمده است که اولین مراجعه به حافظه نهان (کَش)[1] است و اگر داده در کش نبود که احتمال آن ۱-h می‌باشد به سراغ حافظه اصلی می‌رویم و با زمان t_m داده را از حافظه اصلی می‌خوانیم. مثلاً اگر نرخ برخورد ۰/۸۵ باشد و زمان دسترسی به حافظه اصلی ۲۰۰ns و زمان دسترسی به حافظه کش ۲۵ns باشد در این صورت زمان متوسط دسترسی برابر است با: ۲۵+۰/۱۵×۲۰۰=۵۵ns

اگر حافظه جانبی را نیز در نظر بگیریم و تعاریف زیر را بپذیریم:

t_c : زمان دستیابی به حافظه کش

t_{m_1} : زمان دستیابی به حافظه اصلی

t_{m_2} : زمان دستیابی به حافظه جانبی

h_1 : نرخ برخورد کش (احتمال بودن داده در کش)

h_2 : نرخ برخورد حافظه اصلی (احتمال بودن داده در حافظه اصلی)

t_a : زمان متوسط دسترسی

آنگاه:

$$t_a = t_c + (1-h_1)t_{m_1} + (1-h_1)(1-h_2)t_{m_2} \qquad (2)$$

نگاشت (Mapping)

انتقال داده‌ها از حافظه اصلی به حافظه کش نگاشت نام دارد. سه روش عمده نگاشت مطرح است که عبارتند از:

۱ـ نگاشت انجمنی یا شرکت‌پذیر (Associative)

۲ـ نگاشت مستقیم (Direct)

۳ـ نگاشت شرکت‌پذیر مجموعه‌ای (Set-associative)

۱) نگاشت شرکت‌پذیر

در این نوع، داده‌ها از حافظه اصلی به هر خانه از کش که خالی باشد نگاشته می‌شوند و آدرس به صورت شرکت‌پذیر جستجو می‌شود. در واقع در این روش حافظه کش به صورت حافظه انجمنی، آدرس و محتوای حافظه اصلی را در خود ذخیره می‌نماید. آدرس ارائه شده توسط CPU با بخش آدرس در حافظه کش مقایسه می‌شود و در صورت وجود دادهٔ مورد نظر در کش، آن داده به CPU ارسال می‌شود (البته در صورتی که عملیات خواندن باشد). در صورتی که عملیات

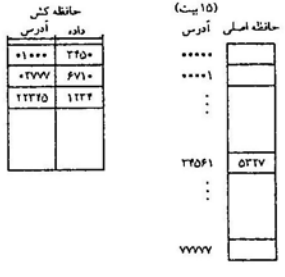
1- cache

نوشتن باشد CPU به محل مورد نظر می‌نویسد. در صورت عدم وجود کلمه مورد نظر در کش، آدرس CPU به حافظه اصلی اعمال می‌شود و کلمه مورد نظر (یا گروهی از کلمات همراه با کلمه مورد نظر) به همراه آدرس به حافظه کش نگاشته می‌شود.

اگر حافظه کش گنجایش نداشته باشد، باید کلمه‌ای (یا گروهی) از حافظه کش حذف شود و کلمه مورد نظر جای آن را بگیرد. این که کدام کلمه از کش حذف شود به الگوریتم به کار رفته بستگی دارد.

مثال (۱): فرض کنید حافظه اصلی ۱۲k×۱۲ باشد و حافظه کش دارای ۵۱۲ کلمه باشد.

با توجه به شکل (۷ـ۶)، سه کلمه در کش وجود دارد. مقادیر آدرس که باید ۱۵ بیتی باشند (به خاطر اینکه حافظه اصلی ۳۲k یعنی 2^{15} می‌باشد و احتیاج به ۱۵ خط آدرس دارد.) به صورت اُکتال (مبنای ۸) نمایش داده شده‌اند (در ۵ رقم) و همچنین داده متناظر هر کدام در ۲ رقم مبنای ۸ نشان داده شده است.



(شکل ۶ـ۷) اعداد در مبنای ۸ هستند.

وقتی CPU یک آدرس ۱۵ بیتی تولید می‌کند در حافظه کش آن آدرس جستجو می‌شود، اگر برد، داده مورد نظر خوانده می‌شود. فرض کنید CPU آدرس ۲۲۳۴۵ را تولید کرده است در این صورت داده ۱۲۳۴ به CPU منتقل می‌شود ولی اگر مثلاً CPU آدرس ۳۴۵۶۱ را تولید کند در کش نیست و از حافظه اصلی داده ۵۳۳۷ به کش و CPU منتقل می‌شود. (می‌توان گفت هر کلمه حافظه کش ۲۷ بیتی است.) در این روش می‌توان حافظه اصلی و کش را بلوک‌بندی (blocking) کرد. یعنی چند کلمه را با هم یک بلوک در نظر گرفت در این صورت نبودن داده در کش، به جای فقط آن کلمه، کل بلوک به کش منتقل می‌شود. این عمل موجب بهبود نرخ برخورد خواهد شد زیرا طبق اصل مجاورت مکانی اگر اکنون به داده i رجوع کرده‌ایم به احتمال زیاد بعداً به داده i+۱ نیاز خواهیم داشت.

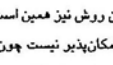
در مثال قبل اگر فرض کنیم هر بلوک شامل ۸ کلمه باشد در این صورت حافظه کش شامل $\frac{512}{8}=64$ بلوک و حافظه اصلی شامل $\frac{32k}{8}=\frac{2^{15}}{2^3}=2^{12}$ یعنی ۲۰۹۶ بلوک خواهد بود و در صورت عدم وجود داده در کش یک بلوک به کش منتقل می‌شود.

۲) نگاشت مستقیم:

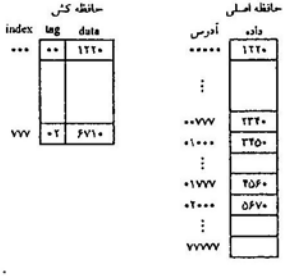
در این نگاشت هر کلمه از حافظه اصلی فقط یک مکان در حافظه کش دارد مثلاً داده موجود در آدرس صفر فقط می‌تواند

به آدرس صفر حافظه کش نگاشته شود. در کل داده موجود در آدرس i می‌تواند به آدرس i mod k که k تعداد کلمات کش است نگاشته شود در مثال قبل داده موجود در آدرس ۵۱۲ (دهدهی) از حافظه اصلی فقط می‌تواند به آدرس ۵۱۲ mod ۵۱۲ یعنی آدرس یک از حافظه کش نگاشته شود. اشکال این روش نیز همین است. به عنوان مثال اگر بخواهیم به طور همزمان به آدرس‌های یک و ۵۱۳ رجوع کنیم، در این روش امکان‌پذیر نیست چون هر دو از یک آدرس مشترک در کش استفاده می‌کنند پس نرخ برخورد در این روش پایین است، ولی عمل نگاشت سریعتر انجام می‌شود.

در این روش آدرس تولیدشده توسط CPU به دو قسمت به نام‌های Index و tag تقسیم می‌شود. تعداد بیت‌های Index برابر تعداد بیت‌هایی است که برای آدرس‌دهی کش نیاز است مثلاً اگر ظرفیت حافظه اصلی 2^m و ظرفیت کش 2^k کلمه باشد، آدرس تولید شده توسط CPU که m بیتی است، دارای دو بخش یکی k بیتی به نام Index و یکی m-k بیتی به نام tag می‌باشد:

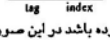


مثال (۲):



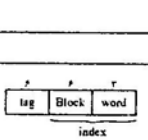
شکل (۷ـ۷) اعداد مبنای ۸ هستند.

اگر فرض کنیم حافظه اصلی دارای ۳۲k و حافظه کش دارای ۵۱۲ کلمه باشند در این صورت آدرس ۱۵ بیتی است که ۹ بیت مربوط به Index و ۶ بیت tag می‌باشد.



فرض کنید CPU آدرس مثلاً ۰۰۷۷۷ را تولید کرده باشد در این صورت به آدرس ۷۷۷ کش رجوع می‌شود چون این داده فقط در این آدرس امکان دارد وجود داشته باشد. tag آدرس مورد نظر ۰۰ است ولی در حافظه کش در قسمت tag عدد ۰۲ وجود دارد پس داده مورد نظر در کش نیست، به سراغ حافظه اصلی رفته و داده مورد نظر یعنی عدد ۲۳۲۰ را خوانده در کش قرار می‌دهیم و tag آن یعنی ۰۰ را نیز در قسمت tag ذخیره می‌کنیم.

اگر در این روش حافظه اصلی و کش را بلوک‌بندی کنیم در این صورت نرخ برخورد افزایش می‌یابد فرض کنید هر بلوک شامل ۸ کلمه باشد در این صورت حافظه کش شامل ۶۴ بلوک خواهد بود. آدرس در این حالت شامل سه فیلد خواهد بود:



در واقع بخش Index در قسمت شده است یکی Block که ۶ بیت است و این به خاطر تعداد بلاک‌های کش یعنی $2^6=64$ می‌باشد و دیگری word که ۳ بیت است و این به خاطر تعداد کلمات در بلوک یعنی $2^3=8$ می‌باشد. شکل (۸ـ۷) را ملاحظه کنید.

فرض کنید CPU آدرس ۰۵۰۰۰ (مبنای ۸) را تولید کند در این صورت:

Tag = ۰۵

Block = ۰۰

Word = ۰

می‌باشد یعنی داده مورد نظر در بلوک صفر (Block ۰) جستجو می‌شود و اولین کلمه بلوک است، ولی tagها برابر نیستند پس miss رخ می‌دهد و بلوک مورد نظر از حافظه اصلی به بلوک صفر کش منتقل می‌شود.

	Index	tag	Data
$Block_0$	000	01	3450
	007	01	6575
$Block_1$	010		
	017		
	⋮	⋮	⋮
$Block_{63}$	770	02	
	777	02	6710

شکل (۸ـ۷) حافظه کش اعداد مبنای ۸ هستند.

۳) نگاشت شرکت‌پذیر مجموعه‌ای

این نگاشت حُسن دو روش قبلی را دارد و از معایب آنها مصون است. در این روش هر کلمه کش می‌تواند دو یا بیشتر کلمه را در خود جای دهد. مثلاً اگر هر کلمه کش بتواند دو کلمه از حافظه اصلی را در خود جای دهد (به هر کلمه کش یک set گفته می‌شود) به این روش 2-way set associative گویند. در این حالت می‌توان دو کلمه با Index یکسان و tagهای متفاوت را در کش قرار داد. شکل ۹ـ۷ چگونگی این امر را نشان می‌دهد. در این روش (2-way) هر کلمه کش $2(6+12)=36$ بیت می‌باشد (2Data + 2tag) با توجه به شکل ۹ـ۷ اگر پردازنده آدرس ۰۳۷۷۷ را تولید کند با توجه به اینکه ایندکس این آدرس ۷۷۷ است به آدرس ۷۷۷ حافظه کش رجوع می‌شود و tag این آدرس یعنی ۰۳ با tagهای موجود (۰۲ و ۰۰) مقایسه می‌شود و مشخص است که کلمه مورد نظر در کش نیست، کلمه مورد نظر از حافظه اصلی به کش منتقل می‌شود و به جای یکی از کلمات موجود در آدرس ۷۷۷ کش قرار می‌گیرد. اینکه کدام کلمه کش دور ریخته شود به الگوریتم به کار رفته بستگی دارد. مثلاً می‌توان از الگوریتم FIFO استفاده کرد یعنی کلمه‌ای که زودتر در کش قرار گرفته زودتر بیرون رود. و یا از الگوریتم‌های LRU، LFU و ... می‌توان استفاده کرد.

index + tag تعیین آدرس
index: indicates block set

Index	Tag	Data	Tag	Data
000	01	3450	02	5670
⋮				
777	02	6710	00	2349

شکل (۹ـ۷) اعداد مبنای ۸

در این روش نیز می‌توان حافظه اصلی و کش را بلوک‌بندی کرد که نرخ برخورد افزایش یابد. اگر حافظه‌ها را بلوک‌بندی کنیم در این صورت مثلاً اگر روش k-way set asseciative را استفاده کنیم در این صورت اگر در هر مجموعه k بلوک موجود باشد و فرض کنیم:

ظرفیت حافظه اصلی = 2^m بایت

ظرفیت کش = 2^c بایت

تعداد کلمات هر بلوک = 2^w کلمه

آنگاه:

تعداد بلوک‌های موجود در کش = $\frac{2^c}{2^w} = 2^{c-w}$

تعداد بلوک‌های موجود در حافظه اصلی = $\frac{2^m}{2^w} = 2^{m-w}$

تعداد مجموعه‌ها در کش = $\frac{2^{c-w}}{k}$

مثال: فرض کنید ظرفیت حافظه اصلی = ۱۶MB

ظرفیت کش = ۵۱۲KB و هر مجموعه دارای ۴ بلوک باشد و هر بلوک ۳۲ بایت باشد، در این صورت:

تعداد مجموعه‌های موجود در کش = $\frac{512KB}{32\times4} = \frac{2^9\times2^{10}}{2^5\times2^2} = \frac{2^{19}}{2^7} = 2^{12}$ مجموعه

نوشتن در حافظه نهان (کش)

اگر درخواست CPU خواندن باشد، کلمه مورد نظر در کش جستجو می‌شود و اگر موجود باشد، خوانده می‌شود. در این حالت هیچ عملی روی حافظه اصلی صورت نمی‌گیرد. ولی اگر CPU آدرس را تولید کند و درخواست نوشتن (write) به آن آدرس را صادر کند در این صورت اگر کلمه با آدرس مورد نظر در حافظه کش باشد CPU به آن آدرس می‌نویسد و حال در سیاست مطرح است: یکی اینکه آیا کلمه تغییریافته در کش، در حافظه اصلی نیز نوشته شود (یعنی محتویات حافظه اصلی به روز شود) که به این سیاست write-through گویند. این روش دارای این مزیت است که محتویات حافظه اصلی و کش همواره یکسان است ولی این عیب را دارد که سرعت را پایین می‌آورد.

سیاست دوم به write-back معروف است. در این روش فقط محتویات کش تغییر می‌کند و این تغییرات تا زمانیکه کلمه داخل کش باشد به حافظه اصلی اعمال نمی‌شود و زمانی که کلمه از کش حذف می‌شود، حافظه اصلی به روز می‌شود. این روش نسبت به قبلی دارای سرعت بیشتری است، زیرا ممکن است یک کلمه داخل کش چندین بار تغییر کند که با این روش فقط یک بار تغییرات به حافظه اصلی اعمال می‌شود. این روش در مواقعی که واحدهای دیگری با حافظه اصلی مستقیماً در ارتباط هستند مشکل‌زا است. مثلاً DMA امکان ارتباط مستقیم حافظه و I/O را فراهم می‌کند که در این روش باید قبل از DMA تمام کلمات حافظه اصلی به روز شوند.

فرمول‌های زمان‌های متوسط دسترسی

۱ـ اگر فرض کنیم هر دسترسی اول به کش می‌باشد و از زمان‌های نوشتن صرفنظر کنیم فرمول قبلی را داریم:

$t_a = t_c+(1-h)t_m$

زمان رجوع به حافظه اصلی : t_m نرخ برخورد : h زمان دسترسی به کش : t_c زمان متوسط : ta

۲ـ گاهی از فرمول $t_a = ht_c+(1-h)t_m$ استفاده می‌شود با این فرض که اگر داده داخل کش باشد (احتمال آن h است) داده را با زمان کش می‌خوانیم در غیر این صورت (با احتمال 1-h) داده را از حافظه اصلی می‌خوانیم.

۳ـ فرمول بهتر $t_a = t_c+ht_{cr}+(1-h)t_m$ در این فرمول t_{cr} زمان استعلام از کش و t_{cr} زمان خواندن از کش است یعنی ما همیشه ابتدا از کش استعلام می‌کنیم (به کش رجوع می‌کنیم یا زمان t_{cr}) اگر داده داخل کش باشد آن را می‌خوانیم (عبارت ht_{cr} در فرمول) ولی اگر نباشد از حافظه اصلی می‌خوانیم.

توجه: از سه فرمول فوق معمولاً در حل مسائل از فرمول (۱) استفاده می‌شود.

۴ـ اگر نوشتن را در نظر بگیریم و سیاست write-through باشد آنگاه: $t_a=t_c+(1-h)t_m+w(t_m-t_c)$

که w درصد نوشتن‌ها است $\left(W=\frac{\text{نوشتن‌ها}}{\text{کل مراجعات}}\right)$

مثلاً اگر $t_c=25ns$ و $t_m=200ns$ و $h=\%99$ و $w=\%20$ (یعنی ۲۰ درصد مراجعات نوشتن باشد.)

آنگاه: $t_a=25+(1-\%99)\times 200+\%20(200-25)=26ns$

۵ـ اگر سیاست نوشتن write-back باشد آنگاه: $t_m=t_c+(1-h)t_m+(1-h)t_m=t_c+2(1-h)t_m$

که یک عبارت $(1-h)t_m$ مربوط به آوردن داده از حافظه اصلی به کش و دیگری مربوط به بردن داده از حافظه کش به حافظه اصلی است. می‌دانیم در سیاست write-back هر داده فقط یک بار به حافظه اصلی نوشته می‌شود.)

۶ـ در سیاست write-back معمولاً داده‌هایی که تغییر می‌کنند به حافظه اصلی نوشته می‌شوند نه همه داده‌ها بنابراین اگر احتمال تغییر داده در کش را w_b بنامیم فرمول (۵) به صورت زیر اصلاح می‌شود:

$$\boxed{t_a=t_c+(1-h)t_m+w_b(1-h)t_m}$$

مثال:

در سلسله مراتب حافظه بیشترین زمان کارکرد به ترتیب در کدام نوع حافظه است؟

الف) Auxiliary, Main, Cache ب) Auxiliary, Cache, Main

ج) Cache, Main, Auxiliary د) Main, Cache, Auxiliary

جواب) گزینهٔ الف صحیح است، زیرا حافظهٔ سریع (Cache) به دلیل سرعت بالاتر و نزدیکی بیشتر به CPU، بیشتر مورد استفاده قرار می‌گیرد و تا هنگامی که اطلاعاتی در Cache وجود دارد مراجعه به حافظه اصلی (Main) به صرفه نمی‌باشد و همین گونه است در مورد حافظه‌های اصلی و حافظه کمکی (Storage).

مثال:

در یک سیستم کامپیوتری که اهمیت سرعت بیشتر از اهمیت حجم مصرف شده است، استفاده از کدام یک از حافظه‌های زیر را در کنار CPU پیشنهاد می‌نمایید؟

الف) ROM پویا و یا ROM ایستا فرقی نمی‌کند.

ب) RAM پویا

ج) RAM ایستا

د) سرعت حافظه عموماً از CPU خیلی بیشتر است. بنابراین نوع حافظه مهم نیست.

جواب) گزینهٔ ج صحیح است. گزینهٔ الف که اصلاً اشتباه است و ROM در دو نوع ایستا و پویا وجود ندارد، گزینهٔ (د) نیز

عبارت اشتباهی را بیان می‌کند زیرا سرعت حافظه محدودکنندهٔ سرعت CPU است نه برعکس و بین گزینه‌های ب و ج نیز چون RAMهای ایستا سیکل خواندن و نوشتن کوتاه‌تری دارند ولی در عوض ظرفیت یکسان با RAMهای پویا را در حجم بالاتری ارائه می‌کنند، ارجح استفاده از RAM ایستا می‌باشد. یعنی گزینه ج صحیح است.

مثال:

کدام عبارت در مورد حافظه‌های RAM و ROM صحیح نیست؟

الف) حافظه‌های RAM گذرگاه داده دو جهته دارند ولی در حافظه‌های ROM این گذرگاه یک جهته است.

ب) هر کدام از این حافظه‌ها می‌توانند چندین خط انتخاب داشته باشند (خط انتخاب منظور خط cs می‌باشد.)

ج) حافظه‌های ROM به دلیل اینکه از بیت‌های فقط خواندنی ساخته شده‌اند، سلول‌های حافظهٔ کم حجم‌تری دارند و ظرفیت یکسان با RAM را در حجم کمتری ارائه می‌نمایند.

د) تعداد خطوط گذرگاه آدرس در حافظه‌های RAM و ROM هیچ ارتباطی با ظرفیت حافظه ندارد.

جواب) گزینه د صحیح است زیرا در حافظه‌های RAM و ROM، تعداد کلمات حافظه با استفاده از خطوط گذرگاه آدرس و تعداد بیت‌های هر کلمه با استفاده از خطوط گذرگاه داده تعیین می‌شوند.

مثال:

برای ایجاد حافظه‌ای ۲۰۴۸ بایتی:

الف) ۸ عدد قطعه حافظهٔ RAM ۱۲۸×۸ یا ۸ خط آدرس مشترک نیاز است.

ب) ۱۶ عدد قطعه حافظهٔ RAM ۱۲۸×۸ با ۷ خط آدرس مشترک و یک رمزگشای ۱۶×۲ نیاز است.

ج) ۱۶ عدد قطعه حافظهٔ RAM ۱۲۸×۸ با ۸ خط آدرس مشترک و یک رمزگشای ۱۶×۲ نیاز است.

د) ۱۶ عدد قطعه حافظهٔ RAM ۱۲۸×۸ با ۷ خط آدرس و یک رمزگشای ۸×۲ نیاز است.

جواب) گزینهٔ ب صحیح است. زیرا ۱۶ قطعه حافظه ۱۲۸ بایتی مجموعاً ۲۰۴۸ بایت را تشکیل می‌دهد و چون $2^{11}=2048$، ۱۱ خط آدرس برای آدرس‌دهی نیاز است و از آنجا که $2^7=128$ در نتیجه ۷ خط آدرس، قابل اتصال به هر قطعه RAM می‌باشد و ۴ خط باقیمانده باید یکی از ۱۶ قطعه را انتخاب نمایند، پس رمزگشای ۱۶×۲ برای انتخاب قطعه‌های RAM مورد نیاز می‌باشد.

مثال:

برای ساختن حافظه‌ای با ظرفیت ۱۶KB می‌توان:

الف) ۱۶ عدد حافظهٔ ۱۰۲۴×۱ به کار برد. ب) ۱۶ عدد حافظه ۱۰۲۴×۸ به کار برد.

ج) ۱۶×۸ عدد حافظه ۱۰۲۴×۱ به کار برد. د) ۲ و ۳ صحیح هستند.

جواب) گزینهٔ د صحیح است. زیرا هر قطعه حافظه ۱۰۲۴×۸ یا 1KB را ذخیره می‌نماید که ۱۶ عدد آن ۱۶KB را تشکیل می‌دهد و هر قطعه حافظه ۱۰۲۴×۱، 1KB (۱ کیلوبیت) و ۸ عدد آن 1KB را می‌سازد که ۱۶ برابر آن ۱۶KB خواهد بود.

مثال:

یک قطعه ROM ۱۰۲۴×۸ و یک قطعه RAM ۱۰۲۴×۸ هر کدام با ۳ خط انتخاب ورودی، حداقل چه تعداد پایه بجز پایه‌های تغذیه نیاز دارند؟

تست‌ها

۱ـ فقط در یکی از موارد زیر هر سه جمله: الف، ب و ج نوشته شده در آن صحیح است. آنرا مشخص کنید؟

(مهندسی کامپیوتر ۷۴)

الف) ۱ـ در کامپیوترها Addressing Formatهای Zero addressing، One addressing، Two addr و Three Addr می‌تواند وجود داشته باشد.

۲ـ اصولاً هر Instruction Cycle به سه Cycle: Fetch، Decode و Execution تقسیم می‌شوند.

۳ـ Interruptهای موجود در کامپیوترها از سه نوع External Int، Internal Int و Cpu Int. خارج نمی‌باشند.

ب) ۱ـ اصولاً Registerهای مستقر در داخل Cpu کامپیوترها حداقل به یکی از روشهای Accumulator Organization، Stack Org و General Register Org طراحی شده‌اند.

۲ـ تعداد دستورالعمل‌های موجود در کامپیوترهای RISC بیشتر از تعداد دستورالعمل‌های موجود در کامپیوترهای CISC است.

۳ـ ساختمان سخت‌افزاری Cache Memory با استفاده از ساختار Associative - Full دارای بالاترین سرعت می‌باشد.

ج) ۱ـ Look - up Table مربوط به اولین مرور اسمبلی (First path) معمولاً با ساختمان Associative Memory طراحی می‌شود.

۲ـ در کامپیوترهای RISC از مکانیزم Microprogramming نسبت به کامپیوترهای CISC کمتر استفاده می‌شود.

۳ـ یک برنامه نویس با زبان اسمبلی اصولاً نیاز به اطلاعات: Instruction Format، Mode Addressing و Set of the cpu Register کامپیوتر مورد نظر دارد.

د) ۱ـ کلیه دستورالعمل‌ها در یک کامپیوتر از سه نوع: Data Transfer Instruction، Arithmatic & logic Instruction و I/O Instruction خارج نمی‌باشند.

۲ـ سرعت انتقال برنامه به داخل حافظه کامپیوترها در روش DMA بیشتر از روش I/O Programmed می‌باشد.

۳ـ برای حفظ آدرس بازگشت در زمان وقفه (Interrupt) بهتر است از Stack استفاده نمود.

۲ یک پردازنده دارای یک حافظهٔ Direct Mapped Cache با ۸ بلوک ۲ کلمه‌ای است. پردازنده، آدرس‌های صفر تا ۲۵۵ را به صورت متوالی و تکراری تولید می‌نماید. یعنی رشتهٔ آدرس:

...،۰،۲۵۵،۲۵۴،...،۱،۰،۲۵۵،۲۵۴،...،۱،۰

به صورت متوالی تولید می‌گردد. اگر این کار تا بی‌نهایت ادامه پیدا کند، Hit Rate برابر خواهد بود با:

(مهندسی کامپیوتر ۷۸)

(Hit Rate: عبارتست از درصد مواقعی که به Cache رجوع می‌شود و اطلاعات مورد نیاز در آن موجود است).

الف) ٪۸۷/۵ ب) ٪۷۵ ج) ٪۲۵ د) ٪۱۲/۵

۳ـ در یک حافظهٔ نهان شرکت‌پذیر مجموعه‌ای با دو بلوک در هر مجموعه (Two-way set associative cache) از بلوکهای چهار کلمه‌ای استفاده می‌شود. این حافظه نهان می‌تواند در مجموع ۲۰۴۸ کلمه از حافظه اصلی را در خود جای دهد. اندازهٔ حافظهٔ اصلی ۱۲۸K کلمه است. برچسب (Tag) در این حافظه چند بیت دارد؟ (مهندسی کامپیوتر ۷۹)

الف) ۶ ب) ۷ ج) ۸ د) ۹

۴ـ یک سیستم حافظه شامل حافظه نهان (cache)، حافظه اصلی و حافظه ثانویه است. اگر زمان دستیابی به یک حافظهٔ نهان برابر ۱۰ns و درصد مراجعه و پیداکردن اطلاعات مورد نظر (Hit Ratio) برابر ۰/۹۸ باشد و نیز اگر زمان دستیابی به حافظه اصلی ۱۰۰ns و Hit Ratio مربوط به آن ۰/۹ بوده و زمان دستیابی به حافظه ثانویه 1ms باشد، در این صورت زمان مؤثر دستیابی به اطلاعات حدود ..... است. (مهندسی کامپیوتر ۸۰)

الف) یک میلی‌ثانیه ب) ۲ میکروثانیه ج) ۱۲ نانوثانیه د) ۱۰۰ میکروثانیه

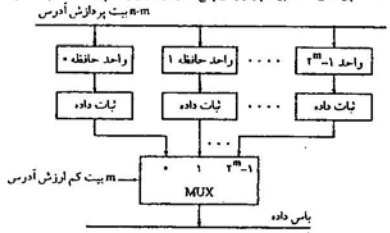
۵ـ تعداد 2^m واحد حافظه در شکل زیر دیده می‌شود. ساختاری که این واحدها به آن متصل است n خط آدرس دارد. چنانچه تأخیر دسترسی (t_{access}) برای هر یک از واحدهای حافظه ۱۰۰ns و تأخیر تسهیم‌کننده (MUX) برای هر بار پیکربندی 5ns باشد و از تأخیر ثباتها صرفنظر کنیم، مدت زمان مورد نیاز برای خواندن پنج آدرس متوالی تحت شرایطی که $2^m > 5$ بوده و n-m بیت پرارزش پنج آدرس فوق با هم مساوی باشد، چقدر است؟ (بر حسب ns) (مهندسی ۸۰)

الف) ۵۲۵

ب) ۵۰۵

ج) ۱۲۵

د) ۱۰۵



۶ یک پردازنده دارای n خط آدرس و J خط داده است فقط یک تراشهٔ حافظه با m خط آدرس و J خط داده را مستقیماً به پردازنده وصل کرده‌ایم (n>m). هر کلمهٔ J بیتی از این تراشه با چند آدرس قابل دسترسی است؟ (مهندسی کامپیوتر ۸۱)

الف) 2^n ب) 2^{n-m} ج) 2^{n-m-1} د) قابل محاسبه نیست.

۷ـ یک حافظهٔ اصلی به بزرگی ۲۵۶K کلمه و یک حافظهٔ نهان (Cache) به بزرگی ۲ بلوک ۴ کلمه‌ای موجود است. روش نگاشت مستقیم (Direct Map) استفاده می‌شود. با فرض خالی بودن Cach، نرخ برخورد (Hit rate) در انتهای صدور آدرسهای ذیل (از چپ به راست) از طرف پردازنده کدام است؟ (مهندسی کامپیوتر ۸۱)

۱۷۰، ۲۵۷، ۱۶۸، ۲۴۶، ۱۷۶، ۱۷۵، ۱۷۶، ۱۷۷، ۱۷۵، ۱۷۶، ۱۷۷ (شروع)

۱۷۵، ۱۷۶، ۱۷۷، ۱۷۶، ۱۷۵، ۱۷۴، ۱۷۳، ۱۷۲، ۱۷۱، ۱۷۰، ۱۶۹، ۱۶۸

۱۶۷، ۱۶۸، ۱۶۵، ۱۶۴

الف) $\frac{21}{27}$ ب) $\frac{20}{27}$ ج) $\frac{10}{27}$ د) $\frac{9}{27}$

الف) ROM ۸ خط خروجی، ۱۴ خط ورودی و RAM ۱۵ خط ورودی و ۸ خط ورودی - خروجی

ب) ROM ۸ خط خروجی، ۱۴ خط ورودی و RAM ۱۶ خط ورودی و ۸ خط ورودی - خروجی

ج) ROM ۱۰ خط خروجی، ۱۲ خط ورودی و RAM ۱۵ خط ورودی و ۸ خط ورودی - خروجی

د) ROM ۱۰ خط خروجی، ۱۲ خط ورودی و RAM ۱۶ خط ورودی و ۸ خط ورودی - خروجی

جواب) گزینهٔ الف صحیح است. در ROM، ۸ خط خروجی برای دادهٔ یک بایتی و ۱۰ خطّ آدرس ورودی برای ۱۰۲۴ بایت مورد نیاز است که به اضافه ۴ خط انتخاب، ۱۴ ورودی را تشکیل می‌دهند ولی در RAM چون هم خواندن و هم نوشتن مدنظر است، خطوط داده ۸ تایی هم ورودی و هم خروجی هستند و ۱۰ خط آدرس برای مشخص کردن ۱۰۲۴ بایت حافظه نیاز است. ۴ خط برای انتخاب و یک خط برای مشخص نمودن عمل خواندن و یا نوشتن کفایت می‌کند که مجموعاً ۱۵ خط ورودی را تشکیل می‌دهند.

مثال:

یک کامپیوتر از RAMهای ۸×۲۵۶ و ROMهای ۸×۱۰۲۴ استفاده می‌نماید. اندازهٔ گذرگاه آدرس ۱۶ بیت است. این کامپیوتر به ۲KB از RAM و ۲KB از ROM و چهار ثبات واسط نیاز دارد. در این کامپیوتر ساختار Memory Mapped I/O به کار رفته است. دو بیت بالارزشتر گذرگاه آدرس برای RAM مقدار ۰۰، برای ROM مقدار ۰۱ و برای ثبات‌های واسط مقدار ۱۰ را اعمال می‌نمایند، محدودهٔ آدرس برای RAM، ROM و ثبات‌ها به ترتیب عبارتند از:

الف) ۰۰۰۰H - ۰۰FFH و ۰۲۰۰H - ۰۲FFH و ۸FFFH - ۸۰۰۰H

ب) ۰۰۰۰H - ۲FFFH و ۲۰۰۰H - ۲FFFH و BFFFH - ۸۰۰۰H

ج) ۰۰۰۰H - ۲FFFH و ۰۲۰۰H - F۲FFH و ۸FFFH - ۸۰۰۰H

د) ۰۰۰۰H - ۲FFFH و ۲۰۰۰H - ۷FFFH و ۸FFFH - ۸۰۰۰H

جواب) گزینهٔ ب صحیح است. در مورد RAM ۸ بیت سمت راست که به RAM وصل می‌شوند (هر قطعه RAM، ۲۵۶ بایت است) از ۰۰H تا FFH را می‌پوشانند و از آنجا که دو بیت سمت چپ نیز می‌بایست ۰۰ باشند، برای RAM مجموعهٔ آدرس‌های ۰۰xx xxx۰ ۰۰۰۰ - ۰۰۰۰ xxxx ۱۱۱۱ ۱۱۱۱ را خواهیم داشت که از ۰۰۰۰H تا ۲FFFH را نشان می‌دهند. در مورد ROM، ۱۰ بیت سمت راست به طور مشترک به قطعات ROM متصل می‌شوند و دو بیت چپ نیز می‌بایست ۰۱ باشند.

مثال:

در یک حافظه سریع انجمنی با اندازهٔ گروه (Set) دوتایی (two-way) از بلوک‌هایی با تعداد کلمات ۴تایی استفاده می‌شود. این حافظه سریع می‌تواند ۲۰۴۸ کلمه از حافظه اصلی را در خود ذخیره نماید و اندازهٔ حافظه اصلی نیز ۱۲۸K×۳۲ می‌باشد، تعداد بیت هر کلمه از حافظه سریع و تعداد بیت فیلد tag در این حافظه به ترتیب عبارتند از:

الف) ۶۴ بیت و ۲ بیت ب) ۶۴ بیت و ۷ بیت ج) ۷۸ بیت و ۶ بیت د) ۷۸ بیت و ۷ بیت

جواب) گزینهٔ د صحیح است. از آنجا که هر حافظه اصلی دارای ۱۰۲۴×۱۲۸ کلمه است، توسط ۱۱ بیت آدرس‌دهی می‌گردد پس مجموع تعداد بیت‌های دو فیلد tag و index، برابر ۱۷ بیت می‌باشد. همچنین با توجه به اینکه حافظه سریع می‌تواند ۲۰۴۸ کلمه از حافظه اصلی را در خود ذخیره نماید و هر کلمه حافظه سریع دو کلمه از حافظه اصلی را شامل

می‌گردد (two-way) در نتیجه، حافظه سریع دارای ۱۰۲۴ کلمه می‌باشد و توسط ۱۰ بیت قابل آدرس‌دهی است و از این امر نتیجه می‌شود که فیلد index، ۱۰ بیتی بوده و ۷ بیت نیز برای tag باقی می‌ماند و چون هر کلمه حافظه سریع از دو کلمه حافظه اصلی به اضافه tagهای مربوطه تشکیل می‌گردد و از آنجا که هر کلمه حافظه اصلی ۳۲ بیتی است در نتیجه تعداد بیت هر کلمه در حافظه سریع ۷×۲+۲×۳۲ می‌گردد که همان عدد ۷۸ خواهد بود.

مثال:

یک کامپیوتر دیجیتال واحد حافظه‌ای با ابعاد ۱۶K×۶۴ و حافظه‌ای سریع با اندازهٔ 1K دارد. روشی که این حافظه سریع استفاده می‌نماید، نقش کردن مستقیم با اندازهٔ بلوک ۴ کلمه می‌باشد. تعداد بیت‌های فیلدهای tag و index و block و word در قالب آدرس به ترتیب عبارتند از:

الف) ۶، ۱۰، ۲ ب) ۶، ۱۰، ۴ ج) ۵، ۱۱، ۲ د) ۵، ۱۱، ۶، ۴

جواب) گزینهٔ الف صحیح است. زیرا حافظه با ابعاد ۶۴K کلمه، ۱۶ خط آدرس نیاز دارد و یک آدرس ۱۶ بیتی باید به آن اعمال شود و چون حافظه سریع دارای 1K کلمه است، فیلد index که آن را آدرس‌دهی می‌نماید باید ۱۰ بیتی باشد و از مجموعه قالب آدرس، ۶ بیت برای tag می‌ماند. فیلدهای block و word نیز در کنار یکدیگر فیلد index را تشکیل می‌دهند و چون هر بلوک ۴ کلمه‌ای است پس برای فیلد word ۲ بیت کفایت می‌کند و ۸ بیت باقیمانده از ۱۰ بیت نیز برای block در نظر گرفته می‌شود.

جدول (۲ـ۱)

دستورات ورودی/خروجی			دستورات ثباتی			دستورات حافظه‌ای			
توصیف	کد مبنای ۱۶	نام دستور	توصیف	کدمبنای ۱۶	نام دستور	توصیف	کدمبنای ۱۶ $I=0$	کدمبنای ۱۶ $I=1$	نام دستور
ورودی	F800	INP	پاک‌کردن‌آکومولاتور	7800	CLA	کلمه حافظه با AC	0xxx	8xxx	AND
خروجی	F400	OUT	پاک کردن E	7200	CLE	کلمه حافظه با AC	1xxx	9xxx	ADD
	F200	SKI[1]	مکمل کردن AC	7180	CMA	بار کردن کلمه حافظه با AC	2xxx	Axxx	LDA
	F100	SKO	مکمل کردن E	7100	CME	ذخیره AC به حافظه	3xxx	Bxxx	STA
	F080	ION[2]	چرخش به راست AC	7080	CIR	پرش بدون شرط	4xxx	Cxxx	BUN
	F040	IOF	چرخش به چپ AC	7040	CIL	پرش با ذخیره آدرس بازگشت	5xxx	Dxxx	BSA
			افزایش AC	7020	INC	افزایش یک واحد کلمه حافظه و پرش از دستور بعدی اگر نتیجه صفر شود.	6xxx	Exxx	ISZ
			پرش از دستور بعدی اگر AC مثبت	7010	SPA				
			پرش اگر AC منفی	7008	SNA				
			پرش اگر AC صفر	7004	SZA				
			پرش اگر E صفر	7002	SZE				
			توقف	7001	HLT				

کنترل و زمان‌بندی

زمان‌بندی تمام ثبات‌ها در کامپیوتر پایه توسط یک تولیدکننده پالس ساعت کنترل می‌شود. پالسهای ساعت به تمام ثبات‌ها و فلیپ فلاپ‌ها اعمال می‌شوند. پالسهای ساعت حالت یک ثبات را تغییر نمی‌دهند بلکه ثبات باید توسط یک «سیگنال کنترلی» فعال شده باشد. سیگنالهای کنترلی در واحد کنترل تولید می‌شوند و شامل ورودی‌های کنترل مالتی‌پلکسر در باس مشترک، ورودی‌های کنترلی ثباتها در پردازنده و ریزعملیات مربوط آکومولاتور هستند.

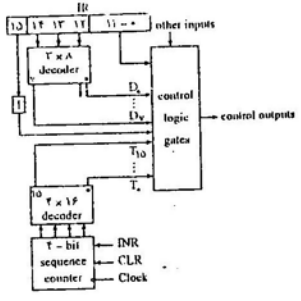
دو نوع ساختار کنترلی عمده وجود دارد: کنترل سیم‌بندی شده[3] و کنترل ریزبرنامه‌ریزی شده[4].

در کنترل سیم‌بندی شده، منطق کنترل توسط گیتها، فلیپ فلاپ‌ها، رمزگشاها، و دیگر مدارات دیجیتال پیاده‌سازی می‌شود. مزیت این نوع کنترل سرعت بالای آن است. در ساختار ریزبرنامه‌ریزی شده، اطلاعات کنترلی در یک حافظه کنترلی ذخیره می‌شود. حافظه کنترل طوری برنامه‌ریزی می‌شود که دنباله‌ای از ریزعملیات‌ها را تولید کند. مزیت این نوع کنترل انعطاف‌پذیری آن است. هر تغییر احتمالی یا به روز کردن اطلاعات حافظهٔ کنترلی، قابل اعمال است. در این بخش نوع اول بررسی می‌شود. شکل ۱.۷ بلوک دیاگرام واحد کنترل را نشان می‌دهد. دستوری که از حافظه خوانده می‌شود در ثبات IR قرار می‌گیرد. کد عملیاتی یعنی بیت‌های ۱۲ تا ۱۴ توسط یک رمزگشای ۳×۸ ترجمه شده و D_0 تا D_7 را تولید می‌کنند. بیت ۱۵ ثبات IR به فلیپ‌فلاپ I منتقل می‌شود. بیت‌های ۰ تا ۱۱ (بخش آدرس) نیز به واحد کنترل منتقل می‌شوند. شمارندهٔ متوالی[5] از ۰ تا ۱۵ شمرده و رمزگشای ۴×۱۶ سیگنالهای زمان‌بندی T_0 تا T_{15} را تولید می‌کند. اغلب اوقات شمارنده یکی یکی افزایش می‌یابد ولی بعضی مواقع نیاز است که شمارش را از صفر شروع کند که پایه clear شمارنده

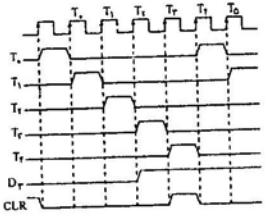
1- Skip on Input
2- Interrupt on
3- Hardwired
4- Micro - Programmed
5- Sequence Counter

این عمل را انجام می‌دهد یعنی ۰ ← SC می‌شود. شکل (۱.۸) دیاگرام زمانی وابستگی سیگنالهای کنترلی را نشان می‌دهد.

در لبهٔ مثبت هر کلاک شمارنده یک واحد می‌شمارد. اگر شمارنده پاک نشود تا T_{15} شمرده و مجدداً به T_0 باز می‌گردد ولی در شکل (۱.۸) با فعال شدن D_3 و T_4، CLR فعال شده و شمارنده از صفر شروع به شمارش می‌کند که این امر موجب تولید T_0 می‌شود.



شکل (۱ـ۷) واحد کنترل کامپیوتر پایه



شکل (۱ـ۸) مثالی از سیگنالهای زمانبندی

سیکل دستورالعمل

سیکل دستورالعمل طبق الگوریتم فن‌نیومن به ترتیب زیر است:

۱- واکشی دستور از حافظه

۲- رمزگشایی دستور

۳- پیدا کردن آدرس مؤثر در صورت غیرمستقیم بودن آدرس‌دهی

۴- اجرای دستور

ریزعملیات واکشی و رمزگشایی عبارتند از:

T_0: AR ← PC

T_1: IR ← M[AR], PC ← PC+۱

T_2: D_0, ..., D_7 ← Decod IR (۱۲ـ۱۴), AR ← IR(۰ـ۱۱), I ← IR(۱۵)

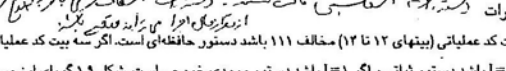
در کلاک اول T_0 تولید شده و در این کلاک محتویات PC روی باس قرار می‌گیرد ($S_2S_1S_0=010$) و ورودی LD ثبات AR فعال شده و PC از طریق باس به AR منتقل می‌شود. در T_1، دستور از حافظه خوانده شده و در IR قرار می‌گیرد و سپس PC به دستور بعد اشاره می‌کند ($PC \longleftarrow PC + 1$).

در کلاک بعد (T_2)، رمزگشایی صورت می‌گیرد یعنی بیتهای ۱۲ تا ۱۴ ثبات دستور ترجمه شده و سیگنالهای D_0 تا D_7 تولید می‌شوند و همچنین بیتهای ۰ تا ۱۱ ثبات دستور وارد AR می‌شوند و بیت شمارهٔ ۱۵ ثبات IR به فیلپ فلاپ I منتقل می‌شود.

انواع دستورات

دسته‌بندی اساسی ثابت نشده. دستورات اشکال زیادی با توجه به نوع شرط در دری که نیاز از دیکودرهای اجرا می‌برایند وجود دارد.

اگر سه بیت کد عملیاتی (بیتهای ۱۲ تا ۱۴) مخالف ۱۱۱ باشد دستور حافظه‌ای است. اگر سه بیت کد عملیاتی مساوی ۱۱۱ باشد و $I=0$ باشد دستور ثباتی و اگر $I=1$ باشد دستور ورودی خروجی است. شکل ۱.۹ گویای این مسئله است.



(a) دستور حافظه‌ای (b) دستور ثباتی (c) دستور ورودی/خروجی

شکل (۱-۹)

با توجه به اینکه در کلاکهای T_0 و T_1 و T_2، واکشی و رمزگشایی دستور صورت می‌گیرد. در T_3 یکی از عملیات زیر صورت می‌گیرد:

۱ـ اگر دستور حافظه‌ای مستقیم باشد ($I=0$) هیچ کاری صورت نمی‌گیرد: $D'_7I'T_3$: Nothing

۲ـ اگر دستور حافظه‌ای غیرمستقیم باشد ($I=1$) آدرس موثر به AR منتقل می‌شود: D'_7IT_3 : $AR \longleftarrow M[AR]$

۳ـ اگر دستور ثباتی باشد، دستور اجرا می‌شود: $D_7I'T_3$: Execute a register instruction

۴ـ اگر دستور ورودی / خروجی باشد، اجرا می‌شود: D_7IT_3 : Execute an I/O instruction

دستورات حافظه‌ای

جدول ۱-۲ هفت دستور حافظه‌ای را نشان می‌دهد. خروجی دیکد شده D_i ($i=0,1,\dots,6$) در جدول نشان داده شده است. این دستورات از سیگنال زمانی T_4 شروع به اجرا می‌شوند.

AND	D_0	$AC \longleftarrow AC \wedge M[AR]$
ADD	D_1	$AC \longleftarrow AC+M[AR],\ E \longleftarrow Cout$
LDA	D_2	$AC \longleftarrow M[AR]$
STA	D_3	$M[AR] \longleftarrow AC$
BUN	D_4	$PC \longleftarrow AR$
BSA	D_5	$M[AR] \longleftarrow PC,\ PC \longleftarrow AR+1$
ISZ	D_6	$M[AR] \longleftarrow M[AR]+1$, if $M[AR]+1=0$ then $PC \longleftarrow PC+1$

جدول (۱-۳) ریزعملیات مربوط به هر دستور حافظه‌ای را نشان می‌دهد.

AND	D_0T_4 : $DR \longleftarrow M[AR]$	BSA	D_5T_4 : $M[AR] \longleftarrow PC$ و
	D_0T_5 : $AC \longleftarrow AC \wedge DR,\ SC \longleftarrow 0$		$AR \longleftarrow AR+1$
ADD	D_1T_4 : $DR \longleftarrow M[AR]$		D_5T_5 : $PC \longleftarrow AR,\ SC \longleftarrow 0$
	D_1T_5 : $AC \longleftarrow AC+DR,\ E \longleftarrow Cout,\ SC \longleftarrow 0$	ISZ	
LDA	D_2T_4 : $DR \longleftarrow M[AR]$		D_6T_4 : $DR \longleftarrow M[AR]$
	D_2T_5 : $AC \longleftarrow DR,\ sc \longleftarrow 0$		D_6T_5 : $DR \longleftarrow DR+1$
STA	D_3T_4 : $M[AR] \longleftarrow AC,\ SC \longleftarrow 0$		D_6T_6 : $M[AR] \longleftarrow DR$ و
BUN	D_4T_4 : $PC \longleftarrow AR,\ SC \longleftarrow 0$		$IF(DR=0)$ then $(PC \longleftarrow PC+1)$,
			$SC \longleftarrow 0$

دستورات ثباتی

این مجموعه دستورات، توسط واحد کنترل، زمانی که $D_7=1$ و $I=0$ باشند، تشخیص داده می‌شوند و بیتهای (۱۱-۰) از کد دستورالعمل یکی از ۱۲ دستور ممکن را مشخص می‌نمایند. در این دستورات با اینکه مراجعه به حافظه نیاز نیست، ولی در سیکل زمانی T_3، محتوای $M[AR]$ به IR منتقل می‌شود. این دستورات در طی زمان T_3 اجرا می‌شوند. اجرای هر دستور ثباتی، تابع کنترلی $D_7I'T_3B_i$ را نیاز دارد و علاوه بر آن، با توجه به بیتهای ۰ تا ۱۱ از هر دستور چون تنها یک بیت از این ۱۲ بیت فعال است، همان بیت به تابع کنترلی وارد می‌شود. در جدول ۱.۲ ملاحظه می‌شود که مثلاً کد دستور برای دستور CLE برابر ۷۲۰۰ (مبنای ۱۶) می‌باشد که در آن بیت شمارهٔ ۱۰، یک می‌باشد و برای اجرای این دستور تابع کنترلی $D_7I'T_3B_{10}$ باید فعال باشد.

جدول (۱-۴) دستورات مراجعه به ثبات و ریزعملیات مربوطه را نشان می‌دهد. در پایان اجرای هر دستور باید $SC \longleftarrow 0$ شود تا سیکل دستور بعدی آغاز شود.

جدول (۱-۴) اجرای دستورات مراجعه به ثبات

$r : SC \longleftarrow 0$

CLA	rB_{11} : $AC \longleftarrow 0$	clear AC
CLE	rB_{10} : $E \longleftarrow 0$	clear E
CMA	rB_9 : $AC \longleftarrow \overline{AC}$	complement AC
CME	rB_8 : $E \longleftarrow \overline{E}$	complement E
CIR	rB_7 : $AC \longleftarrow shr\ AC,\ AC(15) \longleftarrow E,\ E \longleftarrow AC(0)$	circulate Right
CIL	rB_6 : $AC \longleftarrow shl\ AC,\ AC(0) \longleftarrow E,\ E \longleftarrow AC(15)$	circulate Left
INC	rB_5 : $AC \longleftarrow AC+1$	Increment AC
SPA	rB_4 : $IF(AC(15)=0)$ then $(PC \longleftarrow PC+1)$	skip if Positive AC

SNA rB_2 : IF(AC(۱۵)=۱) then (PC ⟵ PC+۱) skip if Negative AC

SZA rB_2 : IF(AC=۰) then PC ⟵ PC+۱ skip if AC Zero

SZE rB_1 : IF(E=۰) then PC ⟵ PC+۱ skip if E Zero

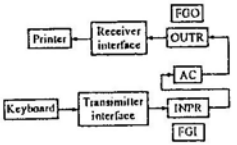
HLT rB_0 : S ⟵ ۰ (S is a start-stop flip-flop) Halt computer

$IR(i)=B_i$: یک بیت IR(۰-۱۱) که مشخص کنندهٔ دستور است و $D_7IT_3=r$ (مشترک برای همهٔ دستورات ثباتی)

دستورات ورودی و خروجی و وقفه

ورودی و خروجی:

اطلاعات سریال از صفحه کلید به ثبات ورودی INPR منتقل می‌شود. اطلاعات سریال برای پرینتر در ثبات خروجی OUTR ذخیره می‌شود. این دو ثبات با یک واسط به صورت سریال و با AC به صورت موازی[1] در ارتباط هستند. پیکربندی ورودی - خروجی در شکل ۱.۱۰ نشان داده شده است.



شکل (۱۰-۱) ورودی - خروجی

پرچم یک بیتی ورودی FGI[2] یک فیلپ فلاپ کنترلی است. وقتی اطلاعات جدیدی در ورودی حاضر است، FGI=۱ می‌شود و زمانی که اطلاعات توسط کامپیوتر پذیرفته می‌شود FGI=۰ می‌شود. این پرچم به منظور همسان‌سازی نرخ انتقال بین ورودی و کامپیوتر مورد نیاز است.

در ابتدا FGI=۰ می‌شود، زمانی که کلیدی روی صفحه کلید فشرده می‌شود، یک کد ۸ بیتی به INPR منتقل می‌شود و FGI=۱ می‌شود. تا زمانی که FGI=۱ باشد محتویات INPR با فشار کلید تغییر نمی‌کند. کامپیوتر FGI را چک می‌کند، اگر ۱ بود، اطلاعات از INPR به طور موازی به AC منتقل شده و FGI=۰ می‌گردد. زمانی که FGI=۰، اطلاعات جدید را می‌توان به INPR منتقل کرد.

در ابتدا، پرچم خروجی FGO یک است. کامپیوتر FGO را چک می‌کند؛ اگر یک بود، اطلاعات از AC به طور موازی به OUTR منتقل شده و FGO=۰ می‌شود. وسیله خروجی اطلاعات را می‌پذیرد. کاراکتر معادل را چاپ می‌کند و زمانی که عملیات کامل شد FGO=۱ می‌شود. زمانی که FGO=۰ کامپیوتر هیچ کاراکتر جدیدی به OUTR بار نمی‌کند.

دستورات ورودی - خروجی

به منظور انتقال اطلاعات (به / از) AC، چک کردن بیت‌های پرچم، و برای کنترل امکان وقفه، از دستورات ورودی - خروجی استفاده می‌شود. کد عملیات دستورات ورودی - خروجی ۱۱۱۱ می‌باشد که با $D_7=1$ و $I=1$ مشخص می‌شود.

1- Parallel 2- FlaG Input

بیتهای باقیمانده ثبات دستور، خود دستور را مشخص می‌کند.

توابع کنترلی و ریزعملیات مربوط به این دستورات در جدول ۵-۱ آمده است. این دستورات در سیگنال زمانی D_7IT_3 اجرا می‌شوند. هر دستور ورودی - خروجی برای اجرا به تابع کنترلی D_7IT_3 نیاز دارد که در جدول با P نشان داده شده است. یکی از بیتهای (۱۱-۶) IR که با B_i مشخص شده است، دستور را مشخص می‌کند. دستور INP اطلاعات ورودی را از INPR به هشت بیت مرتبه پایین AC منتقل کرده و FGI را صفر می‌کند. دستورات ION و IOF، فلیپ‌فلاپ توانساز وقفه IEN را یک و صفر می‌کنند.

جدول (۵-۱)

$D_7IT_3=P$, $IR(i)=B_i$		
	p : SC ⟵ ۰	clear SC
INP	pB_{11}: AC(۰-۷) ⟵INPR, FGI ⟵ ۰	Input character
OUT	pB_{10}: OUTR ⟵ AC(۰-۷), FGO ⟵ ۰	Output character
SKI	pB_9 : IF(FGI=۱) then PC ⟵ PC+۱	Skip on input flag
SKO	pB_8 : IF(FGO=۱) then PC ⟵ PC+۱	Skip on output flag
ION	pB_7 : IEN ⟵ ۱	Interrupt enable on
IOF	pB_6 : IEN ⟵ ۰	Interrupt enable off

وقفه

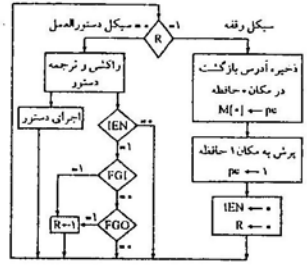
نوع ارتباطی که توضیح داده شد را انتقال تحت کنترل برنامه[1] گویند. کامپیوتر باید دائماً بیت پرچم را چک کند، و اگر بیت مورد نظر ست بود، انتقال اطلاعات را آغاز می‌کند. این روش وقت کامپیوتر را هدر می‌دهد.

روش دیگر این است که وسایل جانبی زمانی که برای انتقال اطلاعات آماده هستند، اعلام کنند. در عین حال کامپیوتر می‌تواند به کار دیگری مشغول باشد. این روش وقفه[2] است. زمانی که کامپیوتر در حال اجرای یک برنامه است، پرچم‌ها را چک نمی‌کند. هر گاه پرچمی ست شود، به کامپیوتر توسط وسیله جانبی وقفه داده می‌شود، کامپیوتر کار فعلی را قطع کرده و انتقال ورودی / خروجی (وقفه) را آغاز می‌کند. پس از پایان سرویس دهی به وقفه، به برنامه قبلی خود بازمی‌گردد.

فلوچارت شکل ۱.۱۱ روشی که وقفه بررسی می‌شود را نشان می‌دهد. یک فلیپ‌فلاپ وقفه به نام R وجود دارد. وقتی R=۰، کامپیوتر سیکل معمولی دستور را آغاز می‌کند. در ضمن فاز اجرای دستور، IEN چک می‌شود. اگر صفر بود یعنی برنامه‌نویس نمی‌خواهد به وقفه پاسخ دهد. اگر IEN=۱، کنترل، بیت‌های پرچم را چک می‌کند، اگر هر دو پرچم (FGI و FGO) صفر باشند، یعنی نه ورودی و نه خروجی برای انتقال آماده نیستند. اگر هر کدام از پرچم‌ها یک باشند و IEN=۱، فلیپ‌فلاپ R یک می‌شود. در انتهای فاز اجرا، کنترل، R را چک می‌کند و اگر ۱ بود، کامپیوتر به سیکل وقفه وارد می‌شود، نه به سیکل عادی دستور.

1- Programmed control transfer 2- Interrupt

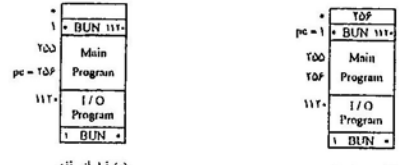
سیکل وقفه، پیاده‌سازیِ سخت‌افزاری دستور پرش و ذخیره آدرس بازگشت است. آدرس بازگشت که در PC است، در مکان بخصوصی ذخیره می‌شود که بتوان پس از بازگشت، دستوری که در آن وقفه رخ داده است را یافت. این مکان می‌تواند، ثباتی از پردازنده باشد یا پشته[1] باشد یا مکانی از حافظه. در کامپیوتر پایه، PC در آدرس ۰ حافظه ذخیره می‌شود. سپس ۱ ← PC شده و IEN و R هر دو صفر می‌شوند.



شکل (۱ـ۱۱)

شکل ۱.۱۲ نشان می‌دهد که هنگام وقوع وقفه چه اتفاقاتی رخ می‌دهد. هنگامی که کامپیوتر در حال اجرای دستور ۲۵۵ است، وقفه رخ می‌دهد. محتویات PC (۲۵۶) در مکان ۰ ذخیره می‌شود و ۱ ← PC و ۰ ← R می‌شود.

در ابتدای سیکل دستور بعدی، دستوری که خوانده می‌شود از مکان ۱ حافظه است. دستور پرش واقع در آدرس ۱ موجب می‌شود برنامه به روال ورودی / خروجی واقع در آدرس ۱۱۲۰ منتقل شود. این روال، پرچم‌ها را چک می‌کند و با توجه به اینکه ورودی یا خروجی رخ داده باشد، اطلاعات ورودی یا خروجی مورد نظر را منتقل می‌کند. پس از آن دستور IEN اجرا می‌شود تا ۱ ← IEN شود و برنامه به مکانی که قبل از وقوع وقفه داشت بازمی‌گردد (۱.۱۲ b).



(a) قبل از وقفه

شکل (۱ـ۱۲ a)

(b) پس از وقفه

شکل (۱ـ۱۲ b)

سیکل وقفه

فلیپ‌فلاپ R زمانی ۱ می‌شود که IEN=۱ و یکی از پرچم‌های FGI و FGO برابر ۱ باشند همچنین سیگنالهای زمانی

1- Stack

T_0, T_1 و T_2 فعال نباشند. یعنی:

$T'_0 T'_1 T'_2 (IEN)(FGI+FGO): R \longleftarrow 1$

اکنون می‌توان فازهای واکشی و ترجمه را با توجه به وقفه اصلاح کرد. بجای استفاده فقط از سیگنالهای زمانی T_0، T_1، T_2. این سیگنالها را باید با R'، AND کنیم. بنابراین فازهای واکشی و ترجمه با توابع کنترلی $R'T_0$، $R'T_1$، $R'T_2$ مشخص می‌شوند. یعنی پس از آنکه اجرای یک دستور خاتمه یافت و SC صفر شد، واحد کنترل تنها در صورتی وارد فاز واکشی می‌شود که R = ۰ در غیر این صورت (R = ۱) واحد کنترل سیکل وقفه را اجرا خواهد کرد. دنبالهٔ ریزعملیات زیر، سیکل وقفه را نشان می‌دهد.

$RT_0: AR \longleftarrow 0, TR \longleftarrow PC$

$RT_1: M[AR] \longleftarrow TR, PC \longleftarrow 0$

$RT_2: PC \longleftarrow PC+1, IEN \longleftarrow 0, R \longleftarrow 0, SC \longleftarrow 0$

با توجه به مطالب گفته‌شده فلوچارت کامل عملیات ماشین در شکل ۱.۱۳ آمده است.

جدول ۱ـ۶ نیز توابع کنترلی و ریزعملیات را برای کامپیوتر پایه نشان می‌دهد.

جدول (۱ـ۶) توابع کنترلی و ریزعملیات کامپیوتر پایه

Fetch	$R'T_0$:	$AR \longleftarrow PC$
	$R'T_1$:	$IR \longleftarrow M[AR], PC \longleftarrow PC+1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \longleftarrow Decode\ IR(12\text{-}14),$
		$AR \longleftarrow IR(0\text{-}11), I \longleftarrow IR(15)$
Indirect	$D'_7 I T_3$:	$AR \longleftarrow M[AR],$
Interrupt		
	$T'_0 T'_1 T'_2 (IEN)(FGI+FGO)$:	$R \longleftarrow 1$
	RT_0:	$AR \longleftarrow 0, TR \longleftarrow PC$
	RT_1:	$M[AR] \longleftarrow TR, PC \longleftarrow 0$
	RT_2:	$PC \longleftarrow PC+1, IEN \longleftarrow 0, R \longleftarrow 0, SC \longleftarrow 0$
Memory-reference:		
AND	$D_0 T_4$:	$DR \longleftarrow M[AR]$
	$D_0 T_5$:	$AC \longleftarrow AC \wedge DR, SC \longleftarrow 0$
ADD	$D_1 T_4$:	$DR \longleftarrow M[AR]$
	$D_1 T_5$:	$AC \longleftarrow AC+DR, E \longleftarrow C_{out}, SC \longleftarrow 0$
LDA	$D_2 T_4$:	$DR \longleftarrow M[AR]$
	$D_2 T_5$:	$AC \longleftarrow DR, SC \longleftarrow 0$
STA	$D_3 T_4$:	$M[AR] \longleftarrow AC, SC \longleftarrow 0$
BUN	$D_4 T_4$:	$PC \longleftarrow AR, SC \longleftarrow 0$
BSA	$D_5 T_4$:	$M[AR] \longleftarrow PC, AR \longleftarrow AR+1$
	$D_5 T_5$:	$PC \longleftarrow AR, SC \longleftarrow 0$
ISZ	$D_6 T_4$:	$DR \longleftarrow M[AR]$
	$D_6 T_5$:	$DR \longleftarrow DR+1$
	$D_6 T_6$:	$M[AR] \longleftarrow DR, if(DR=0) then (PC \longleftarrow PC+1), SC \longleftarrow 0$
Register-reference		

$D_7 I' T_3 = r$ (common to all register-reference instructions)

$IR(i) = B_i (i = 0, 1, 2, \ldots, 11)$