۴-۲ انتقال ثبات

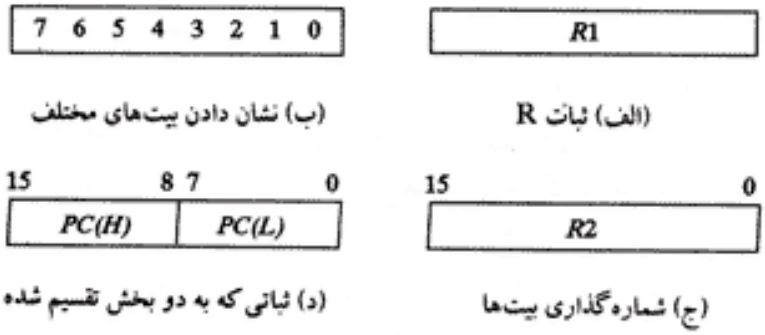
ثبات‌های کامپیوتر با حروف بزرگ الفبای انگلیسی (وگاهی عددهایی به دنبال آنها) برای نشان دادن کارثبات مشخص می‌شوند. مثلاً، ثباتی که آدرس را برای یک واحد حافظه نگمیدارد ثبات آدرس حافظه[1] نام داشته و با MAR مشخص می‌شود. سایر نام‌ها برای ثبات‌ها عبارتند از PC (برای شمارنده برنامه)[2]، IR (برای ثبات دستورالعمل)[3]، و R1 (برای ثبات پردازنده). فلیپ فلاپ‌های تشکیل دهنده یک ثبات n بیتی به طور متوالی از 0 الی n-1 از راست به چپ شماره گذاری می‌شوند. شکل ۱-۴ نمایش ثباتها را بصورت بلاک دیاگرام نشان می‌دهد. متداول‌ترین راه نمایش یک ثبات، استفاده از کادر مستطیل شکل همراه با نام ثبات در داخل آن است، شکل ۱-۴ (الف). بیت‌های مختلف ثبات مانند قسمت (ب) در شکل متمایز می‌شوند. شماره گذاری بیت‌ها در یک ثبات 16 بیتی ، مانند قسمت (ج)، در بالای آن مشخص می‌شود. یک ثبات 16 بیتی هم در قسمت (د) به دو قسمت تقسیم شده است. به بیت‌های 0 تا 7 علامت L (برای بایت کم ارزشتر) و به بیت‌های 8 تا 15 سمبل H (برای بایت باارزشتر) بکار می‌رود. نام ثبات 16 بیتی PC است. سمبل PC(0-7) به بایت کم ارزشتر اتلاق می‌شود و PC(8-15) یا PC(H) به بایت باارزشتر اشاره می‌کند.

انتقال اطلاعات از یک ثبات به ثبات دیگر بصورت سمبلیک با استفاده از عملگر جایگزینی مشخص می‌شود. عبارت:

$$R2 \leftarrow R1$$

بیانگر یک انتقال از ثبات R1 به ثبات R2 است. این عبارت جایگزین شدن محتوای R2 را با R1 مشخص می‌نماید. بنابه تعریف، محتوای ثبات R1 پس از انتقال تغییر نمی‌کند.

عبارتی که مشخص کننده انتقال ثباتها باشد دلالت براین دارد که مدارهایی از خروجی‌های ثبات مبدأ به ورودی‌های ثبات مقصد وجود دارند و ثبات مقصد دارای قابلیت بارشدن موازی است. معمولاً



شکل ۱-۴ بلاک دیاگرام ثبات

1- Memory Address Register 2- Program Counter 3- Instruction Register

می‌خواهیم که انتقال تحت شرایط کنترل از پیش تعیین شده‌ای انجام شود. این موضوع را می‌توان با یک عبارت مانند رابطه زیر نشان داد.

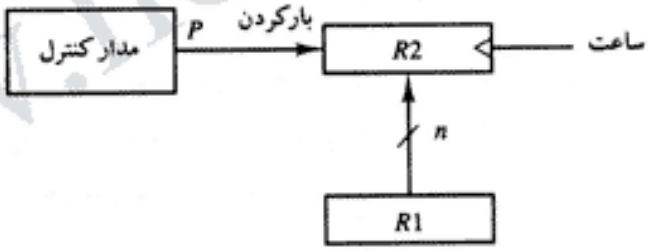
$$If\ (P = 1)\ then\ (R2 \leftarrow R1)$$

که در آن P یک سیگنال کنترلی است که در بخش کنترل تولید می‌شود. گاهی بهتر است که متغیرهای کنترل را با مشخص کردن یک تابع کنترل از عمل انتقال ثبات جدا کنیم. تابع کنترل یک متغیر بولی است که برابر 1 یا 0 است. تابع کنترل در عبارت بصورت زیر مشخص می‌شود

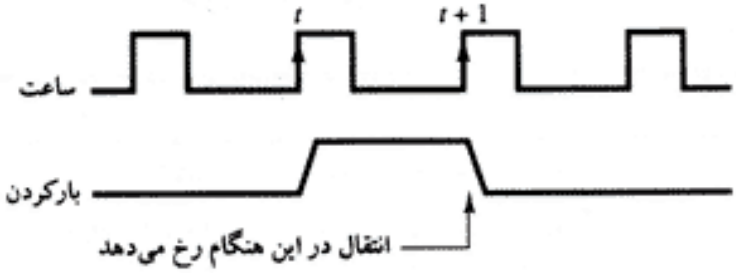
$$P:\quad P2 \leftarrow R1$$

شرط کنترل با علامت نقل[1] ، : ، خاتمه می‌یابد. این سمبل بیان می‌دارد که لازمه عمل انتقال توسط سخت افزار این است که P=1 باشد.

هر یک از عبارت‌های نوشته شده به زبان انتقال ثبات‌ها، دلالت بر وجود یک ساختمان سخت افزاری برای پیاده سازی انتقال دارد. شکل ۲-۴، انتقال از R1به R2 را بصورت بلاک دیاگرام نمایش می‌دهد. n خروجی ثبات R1 به n ورودی ثبات R2 متصل است. از حرف n برای نشان دادن تعداد بیت‌های ثبات استفاده شده است. اگر طول ثبات معین باشد، به جای n، یک عدد واقعی گذاشته



(الف) بلاک دیاگرام



(ب) دیاگرام زمانی

شکل ۲-۴ انتقال از ثبات R_1 به R_2 بهنگام $P = 1$

1- Colon

می‌شود. ثبات R2 دارای ورودی بارکردن است که بوسیله متغیر کنترل P فعال می‌شود. فرض براین است که متغیر کنترل با همان ساعتی که به ثبات اعمال می‌شود همگام[1] است. همانطور که در دیاگرام زمانبندی نشان داده شده است، P در بخش کنترل با لبه بالارونده پالس ساعت در زمان t فعال می‌شود. گذر مثبت بعدی ساعت در زمان t+1 ورودی بارکردن را فعال یافته و سپس داده‌های ورودی R2 بداخل ثبات بصورت موازی بار می‌شوند. در زمان t+1 کنترل P می‌تواند به 0 باز گردد؛ در غیراین صورت، عمل انتقال با هر گذر پالس ساعت مادامی که P فعال است تکرار می‌شود.

توجه کنید که پالس ساعت بعنوان یک متغیر در عبارت انتقال ثبات آورده نشده است. فرض براین است که تمام انتقال‌ها در یک لبه گذر پالس ساعت رخ می‌دهند. هرچند که شرایط کنترلی مانند P درست پس از زمان t فعال می‌شود، ولی عمل انتقال تا وقتی که ثبات با گذر مثبت بعدی ساعت در زمان t+1 فعال نشود رخ نمی‌دهد.

سمبل‌های اصلی زبان انتقال ثبات‌ها در جدول ۱-۴ لیست شده‌اند. ثبات‌ها با حروف بزرگ نشان داده می‌شوند و ممکن است اعدادی به دنبال این حروف آورده شوند. برای مشخص کردن بخشی از یک ثبات از پرانتز استفاده می‌شود و محدوده بیت‌ها یا یک نام سمبلیک برای آن بخش در داخل پرانتز ذکر می‌شود. علامت فلش بیانگر انتقال اطلاعات و جهت آن است. از کاما[2] برای جدا کردن دو یا چند عمل که همزمان انجام می‌شوند استفاده می‌شود. عبارت

$$T\text{: } R2 \leftarrow R1,\quad R1 \leftarrow R2$$

نشان دهنده عملی است که محتوای دو ثبات در طول یک پالس ساعت مشترک با یکدیگر تعویض می‌شوند، مشروط براین که T=1 باشد. این عمل همزمان با ثبات هایی امکان پذیر است که فلیپ فلاپ‌های حساس به لبه پالس داشته باشند.

جدول ۱-۴ سمبل‌های اصلی انتقال ثبات‌ها

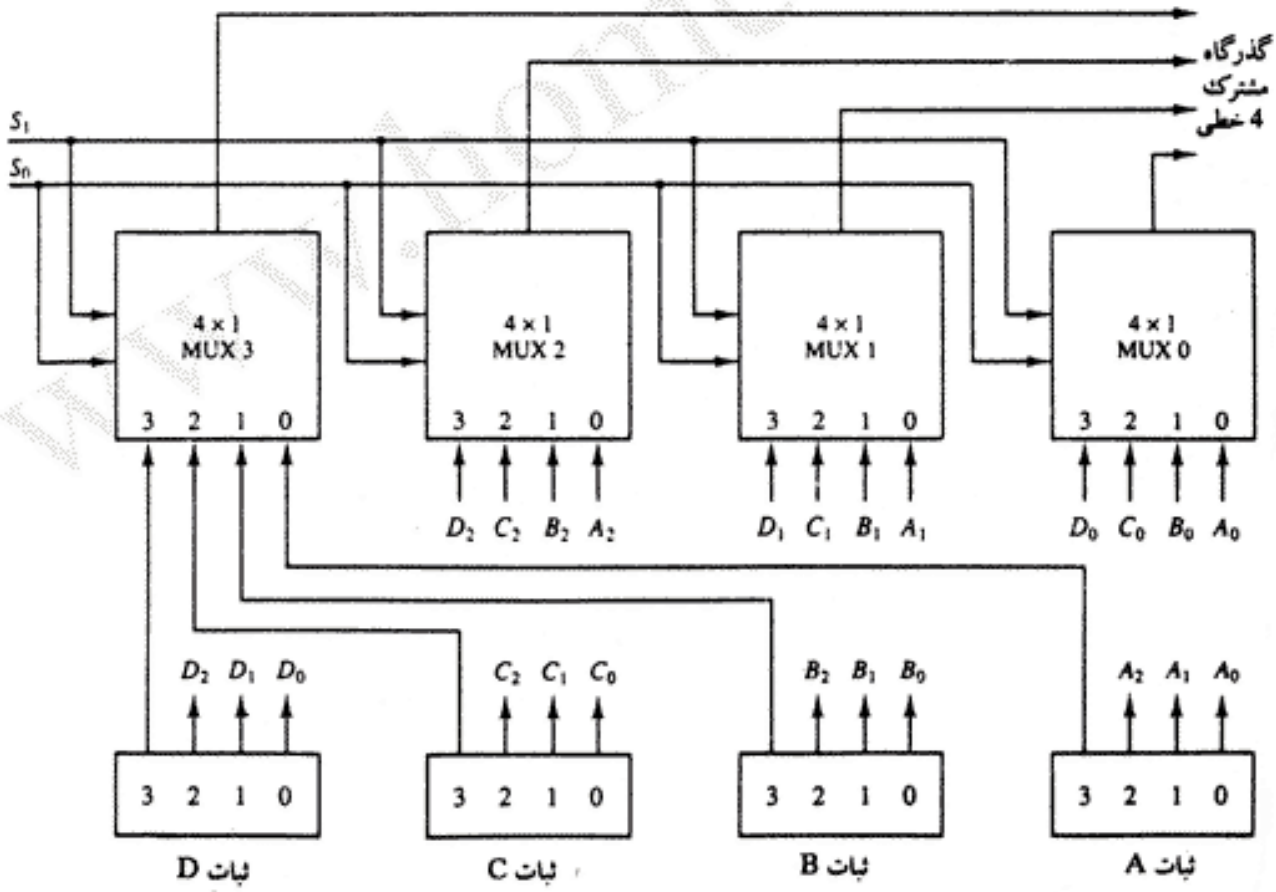
سمبل	توضیح	مثال
حروف (و ارقام)	یک ثبات را مشخص می‌کند	*MAR R2*
پرانتز ()	بخشی از یک ثبات را مشخص می‌کند	*R2(0-7) R2(L)*
پیکان ←	انتقال اطلاعات را مشخص می‌کند	*R2 ← R1*
کاما	دو ریزعمل را از هم جدا می‌کند	*R2 ← R1, R1 ← R2*

1- synchrom 2- Comma

۳-۴ انتقال‌های گذرگاهی و حافظه‌ای

یک کامپیوتر دیجیتال نوعاً ثبات‌های زیادی دارد و باید در آن مسیرهایی برای انتقال اطلاعات از یک ثبات به ثبات دیگر فراهم شود. اگر خطوط جداگانه‌ای بین هر ثبات و دیگر ثبات‌های سیستم به کار رود، تعداد سیم‌ها بیش از حد خواهد شد. روش کاراتر برای انتقال اطلاعات بین ثبات‌ها در یک آرایش چند ثباتی، سیستم گذرگاه مشترک است. ساختار یک گذرگاه از مجموعه‌ای از خطوط مشترک تشکیل می‌شود، که تعداد آنها، یک خط بازای هر یک از بیت‌های ثبات‌هاست، و از طریق آنها اطلاعات دودویی یکی یکی انتقال می‌یابند. سیگنال کنترل، تعیین کننده ثبات انتخاب شده بوسیله گذرگاه هستند.

یک راه ساخت یک سیستم گذرگاه مشترک استفاده ازمولتی پلکسر است. مولتی پلکسرها ثبات مبداء را انتخاب می‌کنند و سپس اطلاعات آنها روی گذرگاه قرار می‌گیرد. ساخت یک سیستم گذرگاه برای چهار ثبات در شکل ۳-۴ نشان داده شده است. هر ثبات دارای چهار بیت است، که از 0 تا 3 شماره‌گذاری شده‌اند. گذرگاه از چهارمولتی پلکسر 4×1 تشکیل شده که هریک چهار ورودی داده 0 تا 3 و دو ورودی انتخاب S_0 و S_1 دارند. برای جلوگیری از پیچیده شدن دیاگرام بعلت تقاطع 16 خط متقاطع،



شکل ۳-۴ سیستم گذرگاه برای چهار ثبات

برای نشان دادن اتصالات خروجی‌های ثبات‌ها به ورودی‌های مولتی پلکسرها از برچسب یا حروف استفاده می‌شود. مثلاً خروجی 1 در ثبات A به ورودی 0 از MUX1 متصل است زیرا این ورودی دارای برچسب A_1 است. با توجه به دیاگرام دیده می‌شود که بیت‌های هم ارزش در هر ثبات به ورودی‌های یک مولتی‌پلکسر متصل می‌شوند تا یک خط گذرگاه را تشکیل دهند. بنابراین MUX0 چهار بیت 0 ثباتها را راه‌دهی می‌کند، MUX1 چهاربیت 1 ثباتها را راه‌دهی می‌نماند، و برای دوبیت دیگر نیز بهمین ترتیب.

دو خط انتخاب S_1 و S_0 به ورودی‌های انتخاب هر چهار مولتی پلکسر متصل‌اند. خطوط انتخاب چهار بیت یک ثبات را انتخاب کرده و آنها را به گذرگاه مشترک منتقل می‌نمایند. اگر $S_1S_0=00$ باشد، ورودی داده 0 هرچهار مولتی پلکسر، انتخاب شده و به خروجی‌ها که گذرگاه را تشکیل می‌دهند اعمال می‌گردند. این موجب می‌شود تا خطوط گذرگاه محتویات ثبات A را دریافت دارند زیرا خروجی این ثبات به ورودی داده 0 مولتی پلکسرها متصل شده است. بطور مشابه، ثبات B بشرطی انتخاب می‌شود که $S_1S_0=01$ باشد والی آخر. جدول ۲-۴ ثباتی را نشان می‌دهد که بوسیله گذرگاه بازاء هریک از چهار مقدار دودویی ممکن خطوط انتخاب، برگزیده شده است.

بطور کلی، یک سیستم گذرگاه، k ثبات n بیتی را برای تشکیل یک گذرگاه مشترک n خطی راه‌دهی می‌نماید. تعداد مولتی پلکسرهای مورد نیاز برای ساخت گذرگاه برابر n است که همان تعداد بیت‌ها در هر ثبات است. اندازه هر مولتی پلکسر باید k×1 باشد زیرا k خط داده را راه‌دهی می‌نماید. مثلاً یک گذرگاه مشترک برای هشت ثبات 16 بیتی به 16 مولتی پلکسر احتیاج دارد که هر یک متعلق به یک خط در گذرگاه است. هر مولتی پلکسر باید دارای هشت خط ورودی داده و سه خط انتخاب برای راه‌دهی مجموعه بیت‌های هم ارزش در هشت ثبات باشد.

انتقال اطلاعات از یک گذرگاه بداخل یکی از چند ثبات مقصد بااتصال خطوط گذرگاه به ورودی‌های تمام ثبات‌های مقصد و فعال کردن کنترل بارکردن ثبات مقصد خاصی که انتخاب شده تحقق می‌یابد. گذرگاه ممکن است در عبارت سمبلیک انتقال گذرگاه ذکر شود و یا وجود آن در عبارت بطور ضمنی فرض گردد. وقتی که گذرگاه در عبارت قید شود انتقال ثبات بصورت زیر نشان داده می‌شود.

محتوای ثبات C روی گذرگاه قرار گرفته است و محتوای گذرگاه نیز با فعال کردن ورودی کنترل بارکردن

$$BUS \leftarrow C, \quad R1 \leftarrow BUS$$

جدول ۲-۴ جدول تابع برای گذرگاه شکل ۲-۴

S_1	S_0	ثباتی که انتخاب می‌شود
0	0	A
0	1	B
1	0	C
1	1	D

در ثبات R1 قرار می‌گیرد. اگر وجود گذرگاه در سیستم مسجل باشد، مناسبتر خواهد بود که فقط جهت انتقال نشان داده شود.

$$R1 \leftarrow C$$

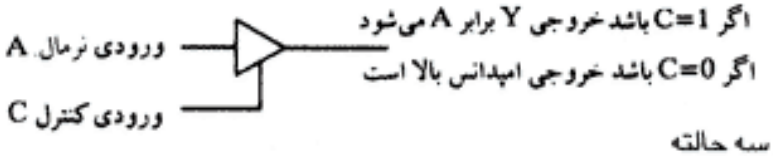
باتوجه به عبارت فوق طراح متوجه می‌شود که سیگنال‌های کنترلی باید فعال شوند تا انتقال از طریق گذرگاه انجام گردد.

بافرهای سه حالتی گذرگاه

گذرگاه سیستم می‌تواند بجای مولتی پلکسر از گیت‌های سه حالتی ساخته شود. گیت‌های سه حالته یک مدار دیجیتال است که سه حالت را در خروجی خود ایجاد می‌کند. دو تا از این حالات سیگنالهای معادل 1 و 0 منطقی همانند گیت‌های معمولی هستند. حالت سوم، حالت امپدانس بالاست. این حالت مانند مدار باز عمل می‌کند، یعنی خروجی قطع است و مفهوم منطقی ندارد. گیت‌های سه حالته می‌توانند هریک از انواع گیت‌های منطقی مانند AND یا NAND باشند. اما نوعی که بیش از همه در طراحی یک سیستم گذرگاه به کار می‌رود گیت بافر[1] (یا میانگیر) می‌باشد.

سمبل گرافیکی یک گیت بافر سه حالته در شکل ۴-۴ نشان داده شده است. وجود ورودی کنترل، علاوه بر ورودی عادی، آن را از میانگیرهای عادی متمایز می‌کند. اگر ورودی کنترل 1 باشد، خروجی فعال می‌شود و گیت مانند هر بافر معمولی کار کرده و خروجی آن برابر ورودی نرمال* است. وقتی ورودی کنترل 0 باشد، خروجی غیرفعال شده و گیت به حالت امپدانس بالا می‌رود و به ورودی نرمال بستگی نخواهد داشت. حالت امپدانس بالای یک گیت سه حالته ویژگی خاصی را فراهم می‌آورد که در گیت‌های دیگر موجود نیست. بعلت این ویژگی، خروجی تعداد زیادی گیت سه حالتی را می‌توان با سیم‌هایی بهم متصل نموده و بدون اینکه خطر تأثیر بارشدن پیش آید یک گذرگاه مشترک را تشکیل داد.

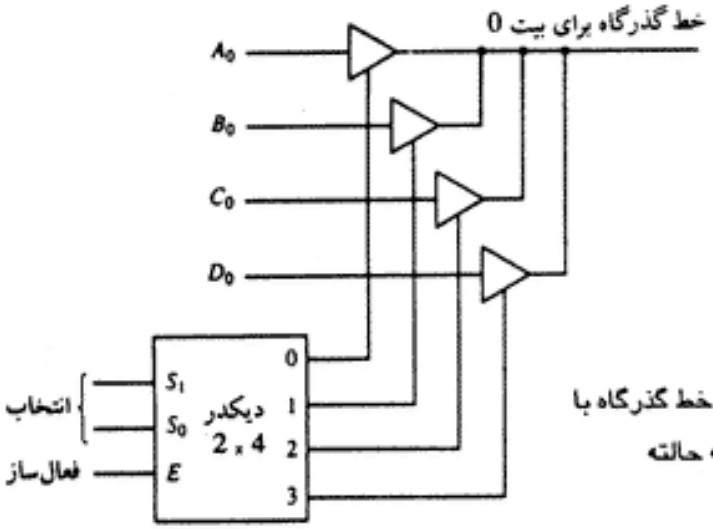
ساختن یک سیستم گذرگاه با میانگیرهای سه حالتی در شکل ۴-۵ نشان داده شده است. خروجی‌های چهار بافر به هم متصل شده‌اند تا یک خط گذرگاه بوجود آید. (دقت داشته باشید که این نوع اتصالات بر روی گیت‌هایی که فاقد خروجی سه حالتی هستند امکان پذیر نیست.) ورودی‌های کنترل بافرها تعیین می‌کنند که کدام یک از چهار ورودی نرمال با خط گذرگاه ارتباط برقرار کند. در هر



شکل ۴-۴ سمبل گرافیکی بافر سه حالته

1- Buffer

* - منظور از بکارگیری کلمه "نرمال" تفکیک ورودی از ورودی کنترل است.



شکل ۵-۴ خط گذرگاه با بافرهای سه حالته

لحظه از زمان فقط یک بافر می‌تواند فعال باشد. بافرهای متصل به خط باید طوری کنترل شوند که فقط یک بافر سه حالته به گذرگاه دسترسی داشته و سایر بافرها در حالت امپدانس بالا باشند.

برای تضمین این که در هر لحظه از زمان فقط یک ورودی کنترل فعال باشد از یک دیکدر مطابق شکل ۵-۴ استفاده می‌شود. وقتی که ورودی توانا‌ساز دیکدر 0 است، هر چهار خروجی آن 0 بوده و خط گذرگاه در حالت امپدانس بالاست زیرا تمام چهار بافر ناتوان شده‌اند. وقتی که ورودی توانا ساز فعال باشد، یکی از بافرهای سه حالته، بسته به مقدار دودویی در ورودی‌های انتخاب دیکدر، فعال می‌شود. بررسی دقیق شکل ۵-۴ نشان می‌دهد که این روش راه دیگری برای ساختن یک مولتی پلکسر 4×1 است زیرا این مدار می‌تواند جایگزین مولتی پلکسر شکل ۳-۴ گردد.

برای ایجاد یک گذرگاه مشترک برای چهار ثبات n بیتی با استفاده از بافرهای سه حالته، ما به n مدار، که در هر کدام چهار بافر سه حالته مانند شکل ۵-۴ وجود داشته باشد نیاز داریم. هر گروه چهار بافره یک بیت هم ارزش را از چهار ثبات دریافت می‌کند. هر خروجی مشترک یکی از خطوط را برای گذرگاه مشترک ایجاد می‌کند که در مجموع n خط بوجود خواهد آمد. برای انتخاب بین چهار ثبات فقط یک دیکدر لازم است.

انتقال‌های حافظه‌ای

عملکرد حافظه در بخش ۷-۲ شرح داده شد. انتقال اطلاعات از یک کلمه حافظه به محیط خارج، "خواندن"[1] نامیده می‌شود. انتقال اطلاعات جدیدی که در حافظه ذخیره می‌شود، "نوشتن"[2] نام دارد. یک کلمه حافظه را با حرف M نشان خواهیم داد. انتخاب یک کلمه حافظه خاص از میان چندین حافظه موجود بوسیله آدرس حافظه در حین انتقال صورت می‌گیرد. وقتی که عمل نوشتن انجام می‌شود لازم

1- Read 2- Write

است آدرس حافظه مشخص شود. این عمل با قرار دادن آدرس در کروشه‌ای که دنبال حرف M آمده است انجام می‌گردد.

واحد حافظه‌ای را در نظر بگیرید که آدرس را از یک ثبات، که آن را ثبات آدرس می‌نامیم، و با AR نشان داده می‌شود، دریافت می‌کند. داده‌ها به ثبات دیگری بنام ثبات داده، DR، منتقل می‌شوند. این عمل خواندن را با عبارت زیر می‌توان بیان کرد.

$$\text{Read:} \quad DR \leftarrow M[AR]$$

این عبارت موجب انتقال اطلاعات از کلمه حافظه M، که بوسیله آدرس موجود در AR انتخاب شده، به DR می‌گردد.

عمل نوشتن محتوای یک ثبات داده را به حافظه M، انتخاب شده بوسیله آدرسی، انتقال می‌دهد. فرض کنید که داده ورودی در ثبات R1 و آدرس در AR باشد. عمل نوشتن بصورت زیر می‌تواند بیان شود.

$$\text{Write:} \quad M[AR] \leftarrow R1$$

عبارت فوق موجب انتقال اطلاعات از R1 به کلمه حافظه M، انتخاب شده بوسیله آدرس موجود در AR ، می‌گردد.

۴-۴ ریزعمل‌های حسابی

ریزعمل[1] یک عمل جزیی است که روی داده‌های ذخیره شده در ثبات‌ها انجام می‌شود. ریزعمل‌هایی که بیش از همه در کامپیوترهای دیجیتال با آن مواجه می‌شویم به چهاردسته زیر طبقه بندی می‌شوند:

۱- ریزعمل‌های انتقال ثبات، اطلاعات دودویی را از یک ثبات به ثبات دیگر منتقل می‌کنند.

۲- ریزعمل‌های حسابی عملیات محاسباتی را روی داده‌های عددی ذخیره شده در ثبات‌ها انجام می‌دهند.

۳- ریزعمل‌های منطقی عملیات دستکاری بیت‌ها را روی داده‌های غیرعددی ذخیره شده در ثباتها انجام می‌دهند.

۴- ریزعمل‌های شیفت[2] اعمال شیفت (جابجایی) را روی داده‌های ذخیره شده در ثبات‌ها اجرا می‌کنند.

ریزعمل انتقال ثبات، در بخش ۲-۴ معرفی شد. این نوع ریزعمل محتوای اطلاعاتی را، هنگامی که از ثباتهای مبدأ به ثبات مقصد منتقل می‌شود، تغییر نمی‌دهد. سه نوع دیگر ریزاعمال محتوای اطلاعاتی را حین انتقال تغییر می‌دهند. در این بخش ما مجموعه‌ای از ریزعمل‌های حسابی را معرفی

1- Microoperation 2- Shift

خواهیم کرد. در دو بخش بعد ما ریزعمل‌های منطقی و شیفت را ارائه خواهیم داد.

ریزعمل‌های حسابی پایه عبارتند از جمع، تفریق، افزایش، کاهش و شیفت. شیفت‌های حسابی بعداً همراه با ریزعمل‌های شیفت توضیح داده خواهد شد. ریزعمل حسابی زیر

$$R3 \leftarrow R1 + R2$$

بیانگر ریزعمل "جمع" است. این عبارت بیان می‌کند که محتوای ثبات R1 با محتوای ثبات R2 جمع شده وحاصل جمع به ثبات R3 انتقال می‌یابد. برای پیاده سازی این عبارت با سخت افزار، ما به سه ثبات و قطعات منطقی که عمل جمع را انجام دهند نیاز داریم. سایر ریزعمل‌های اصلی حسابی در جدول ۳-۴ لیست شده‌اند. تفریق غالباً با استفاده از متمم سازی و جمع پیاده سازی می‌شود. در عوض بکارگیری عملگر "منها"، ما تفریق را بوسیله عبارت زیر مشخص می‌کنیم.

$$R3 \leftarrow R1 + \overline{R2} + 1$$

$\overline{R2}$ سمبل متمم 1 ثبات R2 است. با جمع 1 با متمم 1 متمم 2 تولید می‌شود. جمع محتوای R1 با متمم 2 محتوای ثبات R2، برابر ست با R1-R2.

ریز اعمال افزایش و کاهش را بترتیب بوسیله عمل‌های جمع با 1 و تفریق 1 انجام می‌دهیم. این ریزاعمال با مدارهای ترکیبی یا با شمارنده‌های بالا - پایین[1] پیاده سازی می‌شوند.

اعمال حسابی ضرب و تقسیم در جدول ۳-۴ لیست نشده‌اند. این دو عمل، اعمال حسابی معتبری هستند ولی در مجموعه دستورات پایه منظور نشده‌اند. تنها جایی که این اعمال بعنوان ریزعمل در نظر گرفته می‌شوند، در یک سیستم دیجیتال است که در آن این اعمال بصورت مدارهای ترکیبی پیاده سازی می‌شوند. در چنین حالتی، سیگنال‌هایی این اعمال را در طول گیت‌ها منتشر می‌نمایند و نتیجه

جدول ۳-۴ ریزعمل‌های حسابی

شرح	نمایش سمبلیک
محتوای R1 بعلاوه R2 به R3 منتقل می‌شود	$R3 \leftarrow R1 + R2$
محتوای R1 منهای R2 به R3 منتقل می‌گردد	$R3 \leftarrow R1 - R2$
محتوای R1 متمم می‌شود (متمم 1)	$R2 \leftarrow \overline{R2}$
محتوای R2 متمم می‌شود (منفی می‌شود)	$R2 \leftarrow \overline{R2} + 1$
R1 بعلاوه متمم 2 از R2 (تفریق)	$R3 \leftarrow R1 + \overline{R2} + 1$
یک واحد افزایش در محتوای R1	$R1 \leftarrow R1 + 1$
یک واحد کاهش در محتوای R1	$R1 \leftarrow R1 - 1$

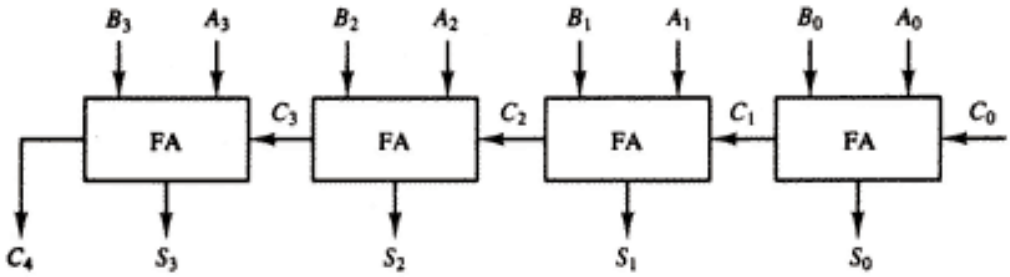
1- UP-Down counter

عمل می‌تواند به محض اینکه سیگنال خروجی از مدارترکیبی عبور کرد با یک پالس ساعت به داخل ثبات مقصد انتقال یابد. در اکثر کامپیوترها، عمل ضرب با رشته‌ای از ریزعمل‌های جمع و شیفت اجرا می‌شود. تقسیم با رشته‌ای از ریزعمل‌های تفریق و شیفت صورت می‌گیرد. مشخص کردن سخت افزار برای این موارد مستلزم استفاده از لیست عباراتی است که در آنها ریزعمل‌های جمع و تفریق و شیفت بکار رفته‌اند.

جمع کننده دودویی

برای پیاده سازی ریز عمل جمع بوسیله سخت افزار، ما به ثبات‌هایی که داده را نگهدارند و نیز قطعات یا مؤلفه‌های دیجیتالی که جمع حسابی را اجرا نمایند، نیازداریم. مدار دیجیتالی که مجموع حسابی دو بیت و رقم نقلی قبلی را تشکیل می‌دهد تمام جمع کننده[1] نامیده می‌شود (شکل ۱۷-۱ ملاحظه شود). مدار دیجیتالی که جمع دوعدد دودویی باهر طول را اجرا می‌نماید جمع کننده دودویی[2] خوانده می‌شود. جمع کننده دودویی از مدارهای تمام جمع کننده که بصورت سری بهم متصلند ساخته می‌شود، باین ترتیب که رقم نقلی خروجی یک تمام جمع کننده به ورودی رقم نقلی تمام جمع کننده بعدی متصل است. شکل ۶-۴ اتصالات درونی چهار تمام جمع کننده (FA) را برای فراهم کردن یک جمع کننده دودویی چهاربیت نشان می‌دهد. بیت‌های مضاف A و مضاف الیه B بوسیله زیرنویس‌هایی از راست به چپ مشخص شده‌اند و در آن زیرنویس 0 متعلق به کم ارزشترین بیت است .ارقام نقلی، بصورت زنجیره‌ای بین تمام جمع کننده‌ها کشیده شده‌اند. رقم نقلی ورودی به جمع کننده دودویی، C_0 و رقم نقلی خروجی C_4 است. خروجی S تمام جمع کننده‌ها بیت‌های حاصل جمع موردنظر را تولید می‌نمایند.

یک جمع کننده دودویی n بیتی به n تمام جمع کننده نیاز دارد. رقم نقلی خروجی از هر تمام جمع کننده به ورودی نقلی تمام جمع کننده مرتبه بالاتر بعدی متصل است. n بیت داده برای ورودی های A از یک ثبات (مانند R1) می‌آیند، و n بیت داده برای ورودی‌های B نیز از ثبات دیگری (مانند R2) گرفته



شکل ۶-۴ جمع کننده دودویی 4 بیتی

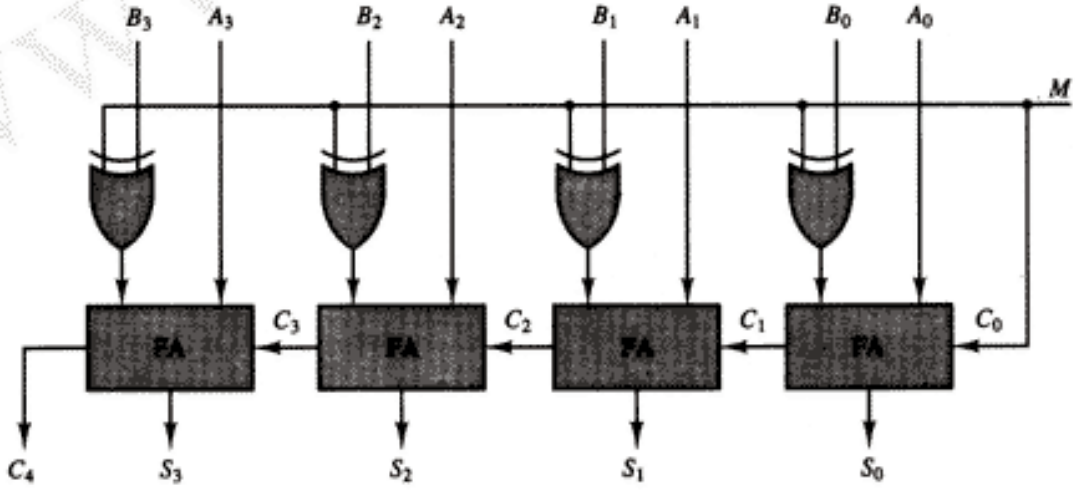
1- Full Adder
2- Binary Adder

می‌شوند. مجموع می‌تواند به ثبات سومی و یا بیکی از ثبات های مبدأ (R1یا R2) منتقل شده و جایگزین محتوای قبلی آن گردد.

جمع و تفریق کننده دودویی

تفریق اعداد دودویی را می‌توان به بهترین وجه با استفاده ازمتمم طبق آنچه در بخش ۲-۳ بحث شده انجام داد . به خاطر بیاورید که تفریق A−B با بدست آوردن متمم B و جمع آن با A انجام می‌شود. متمم 2 ، از متمم 1 و جمع کم ارزشترین جفت بیت‌ها با 1 حاصل می‌گردد. متمم 1 را هم با گیت‌های معکوس کننده می‌توان ایجاد کرد و جمع کردن یک واحد را نیز می‌توان از طریق ورودی نقلی انجام داد.

اعمال جمع و تفریق را می‌توان با افزودن یک گیت OR انحصاری با هم ترکیب کرده و بصورت یک مدار مشترک درآورد . یک مدار جمع و تفریق کننده چهاربیتی در شکل ۷-۴ نشان داده شده است. ورودی حالت[1] M عمل راکنترل می‌کند. اگر M=0 باشد، مدار یک جمع کننده و هر وقت M=1 باشد مدار تبدیل به یک تفریق گر می‌شود. هر گیت OR انحصاری ورودی M و یکی از ورودی‌های B را دریافت می‌کند. اگر M=0 باشد داریم B⊕0=B . تمام جمع کننده‌ها مقدار B و رقم نقلی 0 را دریافت کرده و مدار عمل A بعلاوه B را انجام می‌دهد. اگر M=1 باشد، داریم B⊕1=B′ و C_0=1 . ورودی‌های Bکلاً متمم شده و مقدار 1 از طریق نقلی ورودی با آن جمع می‌شود. مدار عمل A بعلاوه متمم 2 عدد B را انجام می‌دهد. برای اعداد بدون علامت، این عمل اگر A≥B باشد برابر A−B و اگر A<B باشد متمم 2 مقدار B−A خواهد بود. برای اعداد علامت دار نتیجه A−B می‌باشد، بشرطی که سرریزی وجود نداشته باشد.



شکل ۷-۴ جمع و تفریق کننده 4 بیتی

1- Mode Input

افزایشگر دودویی

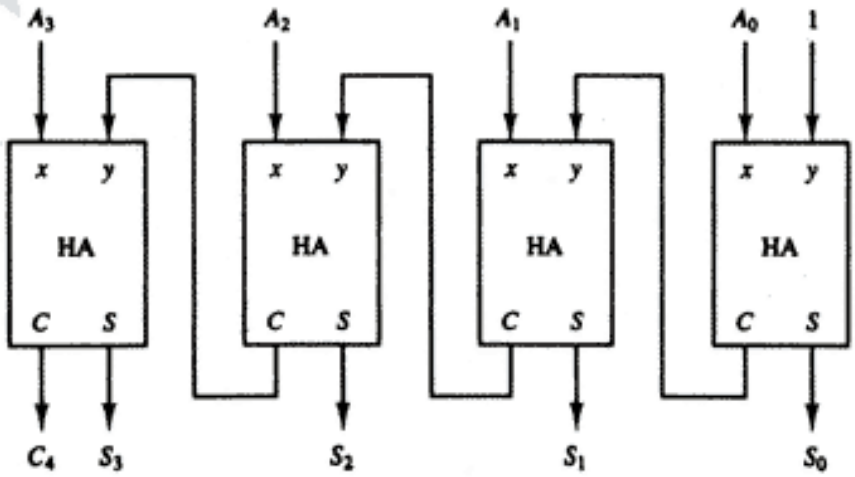
ریزعمل افزایش، به عدد داخل یک ثبات یک واحد اضافه می‌کند. مثلاً اگر یک ثبات 4 بیتی دارای یک مقدار دودویی 0110 باشد، پس از یک افزایش به 0111 بدل خواهد شد. این ریزعمل بسادگی با یک شمارنده (شکل ۱۰-۲ ملاحظه شود) پیاده سازی می‌شود. هر دفعه که تواناساز شمارش فعال شود، لبه پالس ساعت محتوای ثبات را یک واحد افزایش می‌دهد. مواردی هم پیش می‌آید که ریزعمل افزایش باید با یک مدار ترکیبی و مستقل از یک ثبات انجام شود. این کار با استفاده از نیم جمع کننده‌هایی که با هم متوالیاً بسته شده‌اند (شکل ۱۶-۱) امکان پذیر است.

دیاگرام مدار ترکیبی افزایشگر چهاربیتی در شکل ۸-۴ نشان داده شده است. یکی از ورودی‌های نیم‌جمع‌کننده[1] با ارزش کم‌تر به 1 متصل شده است و ورودی دیگر به بیت کم ارزشتر عددی که قرار است افزایش یابد وصل می‌شود. رقم نقلی خروجی از یک نیم جمع کننده به یکی از ورودی‌های نیم جمع‌کننده مرتبه بالاتر بعدی اتصال یافته است. مدار، چهار بیت A_0 تا A_3 را دریافت کرده، یک واحد به آن اضافه می‌کند، و خروجی افزایش یافته را در S_0 تا S_3 تولید می‌نماید. رقم نقلی خروجی C_4 فقط پس از حالت دودویی 1111 برابر 1 خواهد شد. در این حالت S_0 تاS_3 هم 0 می‌شوند.

مدار شکل ۸-۴ می‌تواند به یک افزایشگر n بیتی گسترش یابد و این باگسترش دیاگرام توسط n نیم جمع کننده امکان پذیر است. یکی از بیت‌های کم ارزشتر باید به 1 منطقی متصل گردد. سایر ورودی‌ها، عددی راکه باید افزایش یابد و یا رقم نقلی مرحله قبلی را دریافت می‌کنند.

مدار حسابی

ریزعمل‌های لیست شده در جدول ۳-۴ را می‌توان با یک مدار حسابی تکمیل شده‌ای پیاده سازی



شکل ۸-۴ افزایشگر دودویی 4 بیتی

1- Half Adder

کرد. قطعه اصلی یک مدار حسابی، جمع کننده موازی است. با کنترل ورودی‌های داده به جمع کننده،می‌توان انواع اعمال حسابی مختلف را بدست آورد.

دیاگرام یک مدار حسابی 4 بیتی در شکل ۹-۴ نشان داده شده است. در این دیاگرام چهار مدار تمام کننده وجود دارد که جمع چهار بیت را بوجود می‌آورد و چهار مولتی پلکسر نیز اعمال مختلف را انتخاب می‌کنند. دو ورودی چهار بیت A و B و یک خروجی 4 بیت نیز وجود دارند. چهار ورودی A مستقیماً به ورودی‌های X جمع کننده دودویی می‌روند. هر یک از چهار ورودی B هم به ورودی‌های داده مولتی پلکسرها متصلند. ورودی‌های داده مولتی پلکسرها متمم B را نیز دریافت می‌نمایند. دو ورودی دیگر داده به 0 و 1 منطقی وصل شده‌اند. 0 منطقی یک سطح ولتاژ ثابتی است (0 ولت در مدارهای مجتمع TTL) و سیگنال 1 منطقی می‌تواند توسط یک معکوس کننده که ورودی آن در 0 است تولید گردد. چهار مولتی پلکسر بوسیله دو خط انتخاب S_1 و S_0 کنترل می‌شوند. رقم نقلی C_{in} به ورودی نقلی FA در مکان کم ارزشتر وارد می‌شود. سایر ارقام نقلی از یک مرحله به مرحله بعد متصل می‌شوند.

خروجی جمع کننده دودویی از جمع حسابی زیر محاسبه می‌شود

$$D = A + Y + C_{in}$$

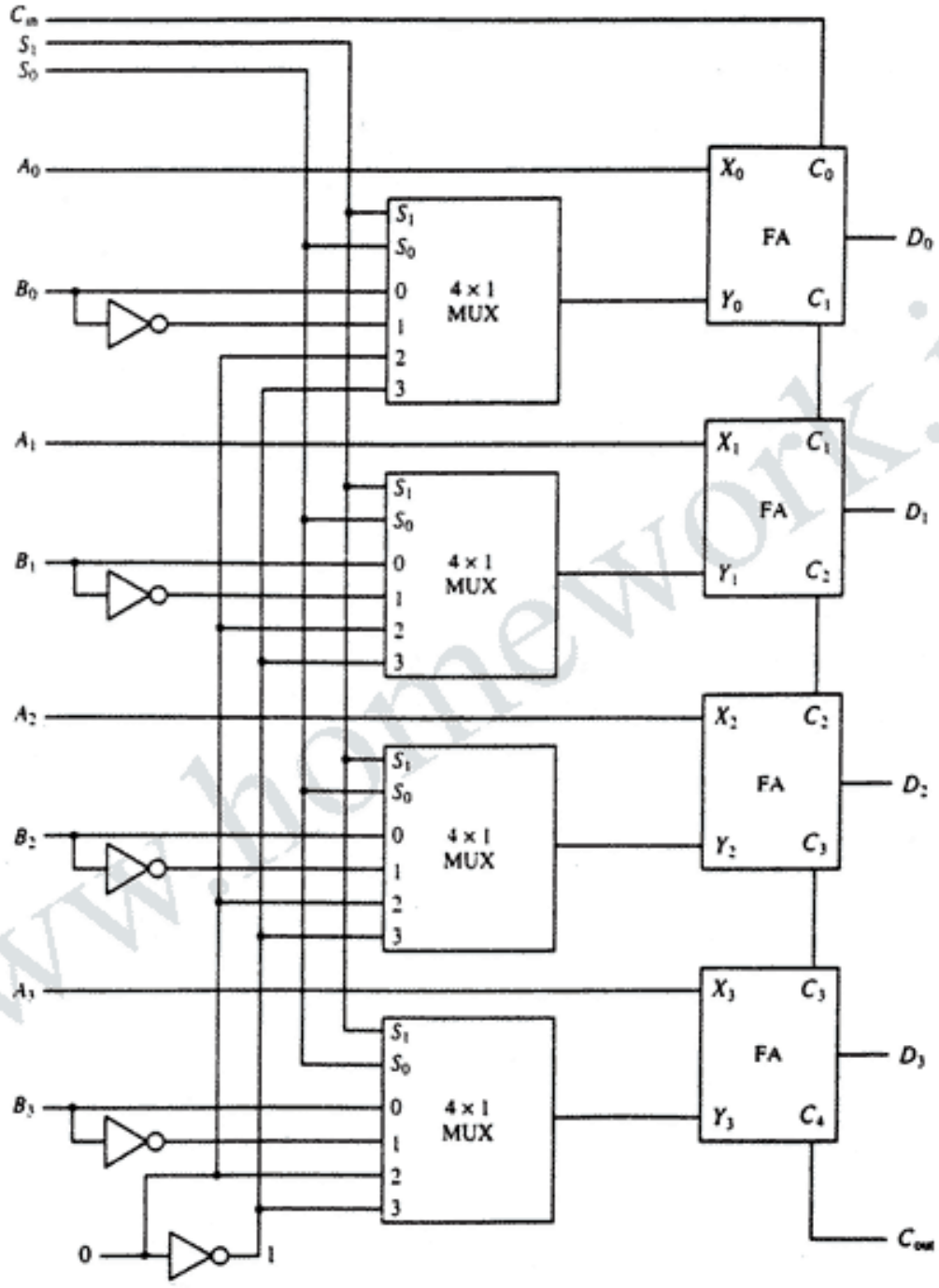
که در آن A یک عدد دودویی 4 بیت در ورودی‌های X، و Y یک عدد دودویی 4 بیت در ورودی‌های Y جمع کننده دودویی است. C_{in} رقم نقلی ورودی است که می‌تواند 0 یا 1 باشد. با کنترل مقدار Y توسط دو ورودی S_0 و S_1 و 0 یا 1 کردن C_{in} می‌توان هشت ریزعمل حسابی لیست شده در جدول ۴-۴ را تولید کرد.

اگر $S_1S_0 = 00$ باشد، مقدار B به ورودی‌های Y جمع کننده اعمال می‌شود. اگر $C_{in} = 0$ باشد، خروجی $D = A + B$ است. اگر $C_{in} = 1$ باشد خروجی $D = A + B + 1$ خواهد بود. هر دو حالت ریزعمل جمع را با و یا بدون نقلی ورودی انجام می‌دهند.

اگر $S_1S_0 = 01$ باشد، متمم B به ورودی‌های Y جمع کننده اعمال می‌گردد. اگر $C_{in} = 1$ باشد سپس $D = A + \bar{B} + 1$ خواهد بود. این عمل A بعلاوه متمم 2 عدد B را که معادل تفریق A−B است تولید می‌نماید. وقتی که $C_{in} = 0$ باشد سپس $D = A + \bar{B}$ خواهد شد. این عمل معادل با تفریق بدون قرض کردن می‌باشد، یعنی، $A - B - 1$.

در $S_1S_0 = 10$، از ورودی‌های B چشم پوشی می‌شوند، ودر عوض، به ورودی‌های Y تماماً 0 اعمال می‌گردد. خروجی برابر خواهد بود با $D = A + 0 + C_{in}$. در این حال اگر $C_{in} = 0$ باشد داریم $D = A$ و اگر $C_{in} = 1$ باشد داریم $D = A + 1$. در حالت اول انتقال مستقیمی از ورودی A به خروجی D وجود دارد. در حالت دوم مقدار A به میزان 1 واحد افزایش یافته است.

اگر $S_1S_0 = 11$ باشد، بداخل ورودی‌های Y جمع کننده تماماً 1 وارد می‌شود تا عمل کاهش 1 واحد، وقتی که $C_{in} = 0$ است، یعنی $D = A - 1$، صورت گیرد. دلیل این مطلب این است که عددی که تماماً از 1 ساخته شده باشد متمم 2 عدد 1 است (متمم 2 عدد 0001 عدد 1111 است). جمع عدد A با



شکل ۹-۴ مدار حساب 4 بیتی

متمم 2 عدد 1 برابرست با A−1 . اگر $C_{in} = 1$ باشد، سپس D=A-1+1=A خواهد شد، که انتقال مستقیمی را از ورودی A به خروجی D ایجاد می‌نماید. توجه کنید که ریزعمل D=A دوبار تولید شده است. بنابراین تنها هفت ریزعمل متمایز از هم در مدار حسابی وجود دارد.

جدول ۴-۴ جدول تابع مدار حساب

انتخاب			ورودی	خروجی	
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	ریزعمل
0	0	0	B	$D = A + B$	جمع
0	0	1	B	$D = A + B + 1$	جمع با نقلی
0	1	0	$\overline{B}$	$D = A + \overline{B}$	تفریق با قرض
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	تفریق
1	0	0	0	$D = A$	انتقال A
1	0	1	0	$D = A + 1$	افزایش A
1	1	0	1	$D = A - 1$	کاهش A
1	1	1	1	$D = A$	انتقال A

۴-۵ ریزعمل‌های منطقی

ریزعمل‌های منطقی اعمال دودویی رابرای رشته‌ای از بیت‌های ذخیره شده در ثبات‌ها مشخص می‌نمایند. اعمال هر بیت ثبات را بطور جداگانه در نظر گرفته و با آنها بعنوان متغیر دودویی رفتار می‌کند. مثلاً ریزعمل OR انحصاری[1] را با محتوای دو ثبات R1 و R2 با عبارت زیر نشان می‌دهیم.

$$P: \quad R1 \leftarrow R1 \oplus R2$$

این عبارت یک ریزعمل منطقی را روی بیت‌های متمایز ثبات‌ها مشخص می‌کند بشرطی که متغیر کنترل P = 1 باشد. بعنوان یک مثال عددی، فرض کنید که هر ثبات دارای چهار بیت باشد. اجازه بدهید محتوای R1 برابر 1010 و محتوای R2 برابر 1100 باشد. ریز عمل OR انحصاری مشخص شده بالا محاسبات منطقی زیر را انجام خواهد داد

محتوای R1	1010
محتوای R2	1100
محتوای R1 پس از P=1	0110

محتوای R1 پس از اجرای ریزعمل برابرست با OR انحصاری بیت به بیت بر روی جفت ثبات‌های R2 و مقادیر اولیه R1. ریزعمل‌های منطقی به ندرت در محاسبات علمی بکار می‌روند، ولی برای دستکاری داده‌های دودویی و تصمیم‌گیری‌های منطقی بسیار مفیدند.

برای تفکیک ریزعمل‌های OR و AND سمبل‌های خاصی بکار خواهند رفت تا از سمبل‌های بکار رفته از توابع بول متمایز شوند. سمبل ∨ برای ریز عمل OR و سمبل ∧ برای ریزعمل AND بکار خواهد رفت. ریزعمل متمم سازی همان متمم 1 بوده و با یک خط که روی نام ثبات قرار می‌گیرد

1– Exclusive OR (XOR, EXOR)

شناخته می‌شود. با بکارگیری سمبل‌های متفاوت می‌توان یک ریزعمل منطقی را از یک تابع کنترل (یا تابع بول) تفکیک کرد. دلیل دیگر برای استفاده از دو مجموعه سمبل این است که سمبل + را وقتی بمعنی جمع حسابی بکار می‌رود، از عمل OR منطقی متمایز نمائیم. هر چند + دارای دو مفهوم است، ولی‌می‌توان بین آنها با توجه به اینکه سمبل در کجا آمده است تفاوت قائل شد. وقتی سمبل + در یک ریزعمل واقع شود، بمعنی جمع حسابی است. وقتی که در تابع کنترل واقع شود (یا تابع بول)، بمعنی عمل OR است. ما هرگز از این علامت برای ریزعمل OR استفاده نخواهیم کرد. مثلاً در عبارت

$$P + Q: \quad R1 \leftarrow R2 + R3, \quad R4 \leftarrow R5 \vee R6$$

علامت + بین P و Q یک عمل OR بین دو متغیر دودویی یک تابع کنترل است. علامت + در بین R2 و R3 یک ریزعمل جمع را مشخص می‌نماید. ریزعمل OR بین R5 و R6 با سمبل ∨ تعریف شده است.

لیست ریزعمل‌های منطقی

شانزده عمل منطقی وجود دارد که می‌توان با دو متغیر دودویی انجام داد. این عمل‌ها را از روی جدول درستی کلی که می‌توان با دو متغیر، مطابق جدول ۵-۴ ، بدست آورد تعریف کرد. در این جدول هر یک از شانزده ستون F_0 تا F_{15} یک جدول درستی را برای هر تابع بول دو متغیر x و y نمایش می‌دهند. توجه کنید که تابع از شانزده ترکیب ممکن که برای F می‌تواند تخصیص یابد بدست آمده است. 16 تابع بول دو متغیر x و y بصورت جبری در جدول ۶-۴ آورده شده‌اند. 16 ریزعمل منطقی با جایگزینی متغیر x با محتوای دودویی ثبات A و متغیر y با محتوای ثبات B حاصل شده‌اند. دانستن این نکته که توابع بولی لیست شده در اولین ستون جدول ۶-۴ رابطه‌ای را بین دو متغیر x و y نشان می‌دهند، اهمیت دارد. ریزعمل‌های لیست شده در دومین ستون رابطه‌ای را بین محتوای دودویی ثبات‌های A و B بیان می‌دارند. با هر بیت از ثبات بصورت یک متغیر دودویی رفتار شده و ریزعمل روی رشته‌ای از بیت‌های ذخیره شده در ثبات انجام می‌گیرد.

پیاده سازی سخت افزاری

پیاده سازی سخت افزاری ریزعمل‌های منطقی مستلزم این است که برای همه بیت‌ها یا جفت

جدول ۵-۴ جدولهای ارزش برای 16 تابع دو متغیره

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

جدول ۴-۶ شانزده ریزعمل منطقی

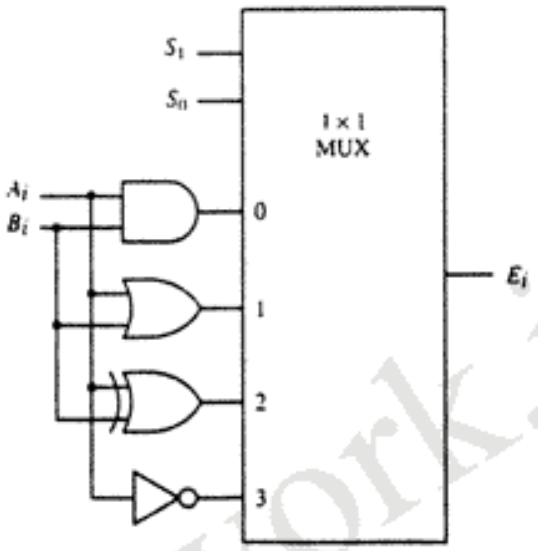
توضیح	ریزعمل	تابع بولی
صفرکردن	$F \leftarrow 0$	$F_0 = 0$
AND	$F \leftarrow A \wedge B$	$F_1 = xy$
	$F \leftarrow A \wedge \overline{B}$	$F_2 = xy'$
انتقال A	$F \leftarrow A$	$F_3 = x$
	$F \leftarrow \overline{A} \wedge B$	$F_4 = x'y$
انتقال B	$F \leftarrow B$	$F_5 = y$
OR انحصاری	$F \leftarrow A \oplus B$	$F_6 = x \oplus y$
OR	$F \leftarrow A \vee B$	$F_7 = x + y$
NOR	$F \leftarrow \overline{A \vee B}$	$F_8 = (x + y)'$
NOR انحصاری	$F \leftarrow \overline{A \oplus B}$	$F_9 = (x \oplus y)'$
متمم کردن B	$F \leftarrow \overline{B}$	$F_{10} = y'$
	$F \leftarrow A \vee \overline{B}$	$F_{11} = x + y'$
متمم کردن A	$F \leftarrow \overline{A}$	$F_{12} = x'$
	$F \leftarrow \overline{A} \vee B$	$F_{13} = x' + y$
NAND	$F \leftarrow \overline{A \wedge B}$	$F_{14} = (xy)'$
همه بیت‌ها در وضعیت 1	F←all 1's	$F_{15} = 1$

بیت‌های ثبات‌ها گیت‌های منطقی گذاشته شود تا تابع منطقی مورد نظر را اجرا نماید. اگرچه 16 ریزعمل منطقی وجود دارد اغلب کامپیوترها فقط از چهارریزعمل AND، OR، XOR (OR انحصاری) و متمم استفاده می‌کنند، و سایر ریزعمل‌ها را نیز می‌توان از این چهار عمل بدست آورد.

شکل ۴-۱۰ یک طبقه از مدار راکه چهار ریزعمل اصلی منطقی را اجرا می‌نماید، نشان می‌دهد. این طبقه متشکل از چهار گیت و یک مولتی پلکسر است. هر یک از چهار عمل منطقی از طریق یک گیت که منطق لازم را انجام می‌دهد تولید می‌شود. خروجی گیت‌ها به ورودی‌های داده مولتی پلکسر اعمال می‌شود. دو ورودی انتخاب S_1 و S_0 یکی از ورودی‌های داده به مولتی پلکسر را انتخاب کرده و مقدار آن را به خروجی هدایت می‌نماید. دیاگرام، نمونه‌ای از طبقات راکه با اندیس i مشخص شده نشان می‌دهد. برای یک مدار منطقی n بیتی، دیاگرام باید n بار برای $i=0, 1, 2, \ldots, n-1$ تکرار شود. متغیرهای انتخاب به تمام طبقات اعمال می‌شوند. جدول تابع شکل ۴-۱۰ (ب) ریزعمل‌های منطقی حاصل برای هر ترکیبی از متغیرهای انتخاب را لیست نموده است.

چند کاربرد

ریزعمل‌های منطقی در دستکاری بیت‌های متمایز یا بخشی از کلمات ذخیره شده در یک ثبات بسیار مفیدند. از این ریزعمل‌ها می‌توان برای تغییر مقادیر بیت‌ها ، پاک کردن گروهی از بیت‌ها، یا واردکردن بیت‌های جدید در ثبات استفاده کرد. مثال‌های زیر چگونگی دستکاری بیت‌های یک ثبات (فرضاً A) را با ریزعملی که تابعی از بیت‌های ثبات دیگر است (فرضاً B) نشان می‌دهد. در یک نمونه از



(الف) نمودار منطقی

S_1	S_0	خروجی	عمل
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \bar{A}$	متمم

(ب) جدول تابع

شکل ۱۰-۴ یک طبقه از مدار منطق

کاربرد، ثبات A می‌تواند ثبات یک پردازشگر و بیت‌های ثبات B عملوندی منطقی حاصل از حافظه باشد که در ثبات B قرار گرفته است.

عمل نشاندن انتخابی[1] 1هایی را در بیت‌های ثبات A که متناظرشان در ثبات B برابر 1 هستند، می‌نشاند. این عمل مکان‌هایی را که متناظرشان در B برابر 0 باشد تغییر نمی‌دهد. مثال عددی زیر این عمل را واضحتر بیان می‌دارد.

A قبل از عمل	1010
B (عملوند منطقی)	1100
A بعداز عمل	1110

دو بیت سمت چپ B ، 1 هستند، بنابراین بیت‌های متناظرشان در A ، 1 خواهند شد. یکی از این دو بیت قبلاً 1 بوده و دیگری از 0 به 1 تبدیل شده است. دو بیت A که نظیرشان در B برابر 0 می‌باشند بلاتغییر باقی مانده‌اند. مثال فوق مانند یک جدول درستی است زیرا تمام چهار بیت ترکیب دو متغیر دودویی را داراست. از جدول درستی درمی‌یابیم که بیت A پس از عمل OR منطقی بیت‌ها در ثبات B و مقادیر قبلی A بدست می‌آیند. بنابراین، ریزعمل OR می‌توانند برای 1کردن بیت‌های انتخاب شده یک ثبات بکار روند.

1- Selective set

عمل متمم سازی انتخابی[1] بیت‌های A را که متناظرشان در B، برابر 1 باشد متمم می‌نماید. این عمل مکان‌هایی را که نظیرشان در B برابر 0 باشد تغییر نمی‌دهد. مثلاً

A قبل ازعمل	1010
عملوند منطقی	1100
A بعداز عمل	0110

مجدداً دو بیت منتهاالیه سمت چپ B، 1 می‌باشند، بنابراین بیت‌های متناظر در A متمم شده‌اند. این مثال هم مانند یک جدول درستی است که از آن می‌توان نتیجه گرفت که عمل متمم کردن انتخابی دقیقاً همان ریزعمل OR انحصاری است. بنابراین از ریزعمل OR انحصاری می‌توان برای متمم کردن انتخابی بیت‌های یک ثبات استفاده کرد.

عمل پاک کردن انتخابی[2] فقط بیت‌هایی از A را، که بیت‌های متناظر آنها در B مقدار 1 دارد، 0 می‌کند مثلاً

A قبل ازعمل	1010
عملوند منطقی	1100
A بعداز عمل	0010

باز هم دو بیت منتهاالیه چپ B مقدار 1 دارند، بنابراین بیت‌های متناظر آنها در A برابر 0 شده‌اند. می‌توان نتیجه گرفت که عمل بولی انجام شده روی بیت‌ها $A\bar{B}$ است. ریزعمل منطقی مربوطه عبارتست از

$$A \leftarrow A \wedge \overline{B}$$

عمل ماسک[3] (پوشش) مشابه عمل پاک کردن انتخابی است بجز اینکه بیت‌های A فقط در صورتی که نظیرشان در B مقدار 0 باشد، پاک می‌شوند. عمل ماسک، همانطور که از مثال عددی زیر دیده می‌شود. یک ریزعمل AND است.

A قبل از عمل	1010
B عملوند منطقی	1100
A پس از ماسک	1000

دو بیت منتهاالیه سمت راست A پاک شده‌اند زیرا بیت‌های متناظرشان در B، 0 هستند. دو بیت سمت چپ بلاتغییر مانده‌اند زیرا بیت‌های نظیرشان در B، 1 می‌باشند. عمل ماسک از عمل پاک کردن انتخابی مناسبتر است زیرا اکثر کامپیوترها دارای دستور AND می‌باشند، و آنهایی که دستوراتی رابرای اجرای ریزعمل پاک کردن انتخابی دارا هستند اندکند.

1- Selective Complement 2- Selective Clear 3- Mask

عمل درج[1]، یک مقدار جدیدی را در گروهی وارد می‌کند. این عمل ابتدا با پوشش بیت‌ها و سپس OR کردن آنها با مقدار موردنظر انجام می‌شود. مثلاً فرض کنید که ثبات A حاوی هشت بیت 01101010 باشد. برای جایگزینی چهار بیت سمت چپ با 1001، ابتدا چهار بیت ناخواسته را ماسک می‌کنیم:

A قبل ازعمل	0110 1010
B (ماسک)	0000 1111
A پس ازعمل	0000 1010

و سپس مقدار جدید را درج می‌نمائیم

A قبل ازعمل	0000 1010
B (درج)	1001 0000
A بعدازعمل	1001 1010

عمل ماسک یک ریزعمل AND و عمل درج یک ریزعمل OR می‌باشد

عمل پاک کردن[2]، دو کلمه A و B را مقایسه کرده و اگر دو عدد مساوی باشند نتیجه تمام 0 را تولید می‌نماید. این عمل توسط ریزعمل OR انحصاری تحقق می‌یابد و بوسیله مثال زیر نشان داده شده است.

A	1010
B	1010
$A \leftarrow A \oplus B$	0000

اگر A و B مساوی باشند، بیت‌های نظیر به نظیر هر دو یا 0 و یا 1 می‌باشند. در هر یک از حالات عمل OR انحصاری، 0 تولید خواهد کرد. نتیجه تمام 0 سپس برای تعیین مساوی بودن دو عدد چک می‌شود.

۴-۶ ریزعمل‌های شیفت

ریزعمل‌های شیفت برای انتقال یا جابجایی سری داده‌ها بکار گرفته می‌شوند. آنها همچنین همراه با عمل‌های حسابی، منطقی و سایر اعمال داده پردازی بکار می‌روند. محتوای یک ثبات می‌تواند به چپ یا به راست شیفت پیدا کند. در همان زمانی که بیت‌ها شیفت داده می‌شوند، اولین فلیپ فلاپ اطلاعات دودویی خود را از ورودی سری دریافت می‌کند. در حین شیفت به چپ، ورودی سری یک بیت را به سمت راست ترین مکان منتقل می‌نماید. در حین عمل شیفت به راست، ورودی سری یک بیت را به سمت چپ‌ترین مکان انتقال می‌دهد. اطلاعاتی که از طریق ورودی سری منتقل می‌گردد تعیین کننده نوع شیفت است. سه نوع شیفت وجود دارد: منطقی، چرخشی و حسابی.

1- Insert 2- Clear

شیفت منطقی [1] مقدار 0 را از طریق ورودی سری انتقال می‌دهد. ما سمبل shl و shr را بترتیب برای ریزعمل‌های شیفت به چپ و شیفت به راست برمی‌گزینیم. مثلاً

$$R1 \leftarrow \text{shl } R1$$
$$R2 \leftarrow \text{shr } R2$$

دو نمونه ریزعمل می‌باشند که محتوای ثبات R1 را یک بیت به چپ و محتوای ثبات R2 را یک بیت به راست شیفت می‌دهند. سمبل ثبات در هر دو طرف فلش باید یکسان باشند. فرض بر این است که حین شیفت، بیت انتقالی به مکان انتهایی از طریق ورودی سری، 0 باشد.

شیفت چرخشی (که عمل چرخش [2] نیز خوانده می‌شود) بیت‌های ثبات را از طریق دوانتها بدون از دست دادن هرگونه اطلاعات می‌چرخاند. این عمل با اتصال خروجی سری به ورودی سری ثبات تحقق می‌یابد. ما سمبل‌های cil و cir را بترتیب برای چرخش به چپ و چرخش به راست بکار خواهیم برد. سمبل‌های بکار رفته وریزعمل‌های شیفت در جدول ۷-۴ نشان داده شده‌اند.

شیفت حسابی [3] ریزعملی است که که یک عدد دودویی علامت دار را به چپ یا راست شیفت می‌دهد. شیفت حسابی به چپ، یک عدد دودویی علامت دار را در 2 ضرب می‌کند. یک شیفت حسابی به راست نیز عدد را بر 2 تقسیم می‌نماید. شیفت‌های حسابی نباید بیت علامت را تغییر دهند زیرا وقتی عدد را در 2 ضرب یا تقسیم می‌کنیم علامت همچنان باقی می‌ماند. سمت چپ‌ترین بیت درثبات بیت علامت را نگه‌می‌دارد، و بقیه بیت‌ها عدد را حفظ می‌کنند. بیت علامت برای اعداد مثبت ، 0 و برای منفی ، 1 است. اعداد منفی در فرم متمم 2 هستند. شکل ۱۱-۴ نمونه‌ای از یک ثبات n بیت را نشان می‌دهد. بیت R_{n-1} در سمت چپ ترین مکان بیت علامت را نگه‌می‌دارد. بیت R_{n-2}، با ارزشترین بیت و R_0 کم ارزشترین بیت است. شیفت به راست حسابی بیت علامت را عوض نمی‌کند و همه بیت‌ها (از جمله علامت) را به راست شیفت می‌دهد. بنابراین R_{n-1} تغییر نمی‌کند، R_{n-2} بیت R_{n-1} را دریافت می‌نماید و برای سایر بیت‌های ثبات نیز بهمین ترتیب. بیت واقع در R_0 از دست می‌رود.

جدول ۷-۴ ریزعمل‌های شیفت

توضیح	سمبل نشاندهنده
شیفت ثبات R به چپ	$R \leftarrow \text{shl } R$
شیفت ثبات R به راست	$R \leftarrow \text{shr } R$
شیفت چرخش ثبات R به چپ	$R \leftarrow \text{cil } R$
شیفت چرخش ثبات R به راست	$R \leftarrow \text{cir } R$
شیفت حسابی ثبات R به چپ	$R \leftarrow \text{ashl } R$
شیفت حسابی ثبات R به راست	$R \leftarrow \text{ashr } R$

1- Logical Shift 2- Rotate 3- Arthmetic Shift



شکل ۱۱-۴ شیفت حسابی به راست

شیفت حسابی به چپ یک 0 وارد R_0 می‌نماید، و کلیه بیت‌های دیگر را به چپ شیفت می‌دهد. بیت اولیه R_{n-1} از دست رفته و با بیت R_{n-2} جایگزین می‌شود. اگر بیت واقع در R_{n-1} پس از شیفت عوض شود، علامت معکوس شده است. این هنگامی رخ می‌دهد که ضرب در 2 سبب سرریز گردد. سرریز در شیفت به چپ هنگامی رخ می‌دهد که قبل از شیفت R_{n-1} و R_{n-2} مساوی نباشد. یک فلیپ فلاپ سرریز V_s برای کشف یک سرریز حاصل از شیفت حسابی به چپ می‌تواند مورد استفاده قرار گیرد.

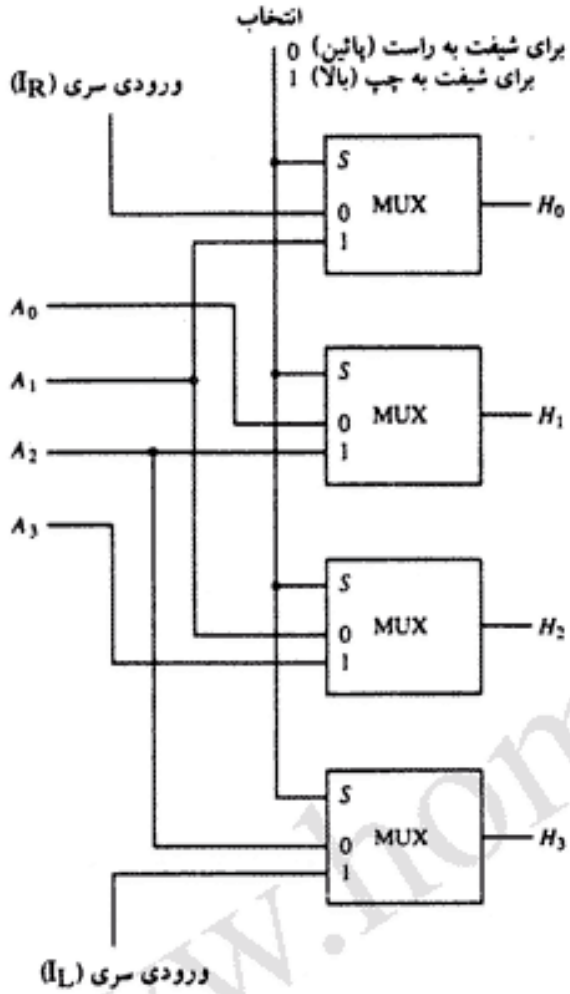
$$V_s = R_{n-1} \oplus R_{n-2}$$

اگر $V_s = 0$ باشد، سرریز وجود ندارد، ولی اگر $V_s = 1$ گردد، سرریز وجود داشته و پس از شیفت علامت عوض خواهد شد. V_s باید با همان پالس ساعت شیفت ثبات وارد فلیپ فلاپ سرریز گردد.

پیاده سازی سخت افزاری

یکی از انتخاب‌های ممکن برای واحد شیفت، ثبات شیفت دوطرفه با بارشدن موازی است (شکل ۹-۲ ملاحظه شود). اطلاعات را می‌توان به صورت موازی وارد ثبات نمود و سپس آنرا به چپ یا راست شیفت داد. دراین نوع آرایش، یک پالس ساعت برای بارشدن داده بداخل ثبات موردنیاز است و پالس دیگری برای شیفت دادن لازم است. در یک واحد پردازنده با ثبات‌های متعدد، بهتر است برای کارایی بیشتر، عمل شیفت را با مدار ترکیبی پیاده سازی کنیم. در این روش محتوای ثباتی که قرار است شیفت کند ابتدا روی گذرگاه مشترک قرار گرفته و خروجی آن به یک شیفت دهنده ترکیبی متصل می‌شود ، وسپس عدد شیفت داده شده به ثبات بازگردانده می‌شود. این نیز تنها یک پالس ساعت برای بارکردن مقدار شیفت یافته بداخل ثبات نیاز دارد.

یک شیفت دهنده ترکیبی را می‌توان مانند شکل ۱۲-۴ با مولتی پلکسر ساخت. شیفت دهنده 4 بیتی دارای چهار ورودی داده A_0 الی A_3 و چهار خروجی داده H_0 الی H_3 است. دو ورودی سری نیز وجود دارد، یکی برای شیفت به چپ (I_L) و دیگری برای شیفت به راست (I_R). وقتی که ورودی انتخاب S=0 باشد، داده ورودی به راست شیفت داده می‌شود (در دیاگرام به سمت پائین). اگر S=1 باشد، داده ورودی به چپ شیفت می‌یابد (به سمت بالای دیاگرام). با n ورودی و خروجی داده به n مولتی پلکسر نیاز دارد. دو ورودی سری بوسیله مولتی پلکسر دیگری قابل کنترل‌اند تا سه نوع شیفت ممکن را فراهم آورند.



جدول تابع

انتخاب	خروجی			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

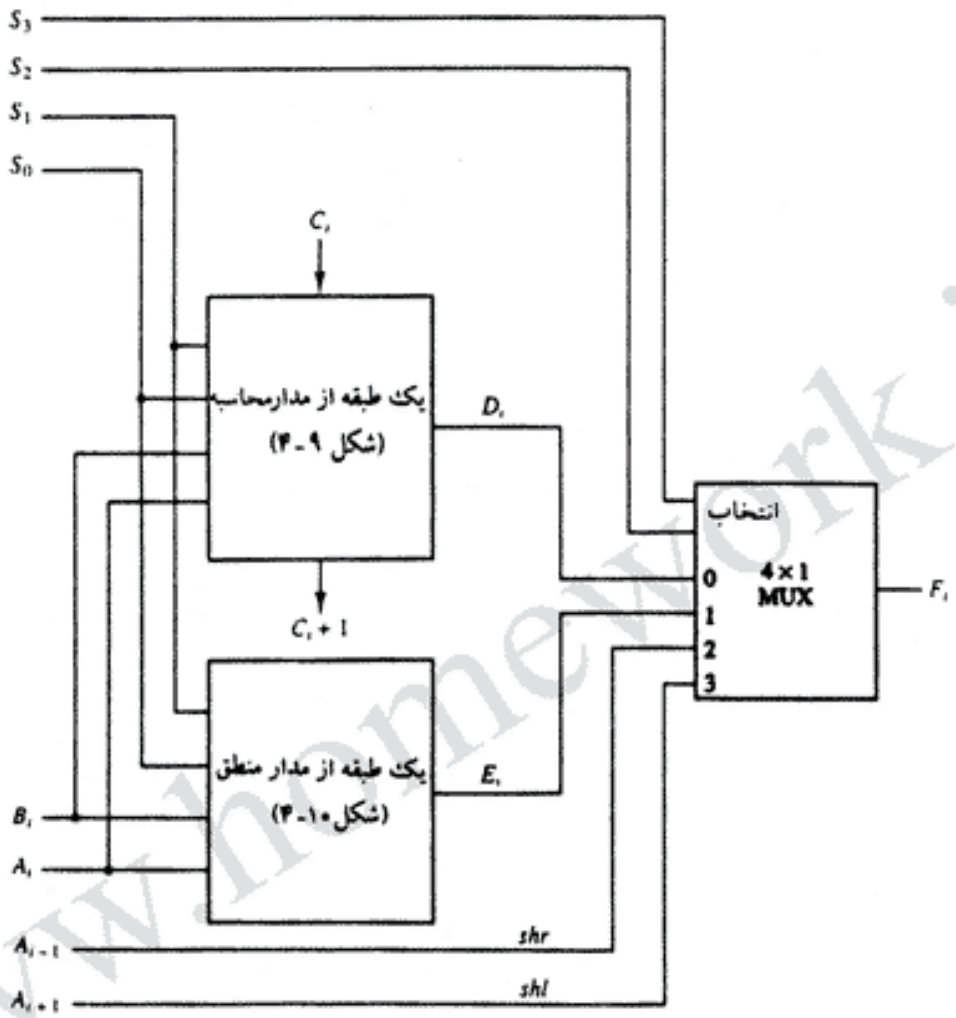
شکل ۱۲-۴ شیفت ترکیبی 4 بیتی

۴-۷ واحد حساب، منطق و شیفت

در سیستم‌های کامپیوتری به جای اینکه ثباتهای مختلف ریزعملها را مستقیماً اجرا کنند، از تعدادی ثبات ذخیره سازی استفاده می‌شود که به یک واحد عملیاتی مشترک بنام واحد حساب ومنطق[1] متصل‌اند. برای اجرای یک ریزعمل، محتوای ثبات خاصی در ورودی ALU مشترک قرار می‌گیرد. ALU عملی را انجام داده و سپس نتیجه را به ثبات مقصد ارسال می‌نماید. چون ALU یک مدار ترکیبی است بنابراین عمل انتقال ثبات از ثبات مبدأ به ALU واز آنجا به ثبات مقصد در یک پریود پالس ساعت صورت می‌گیرد. ریز عمل شیفت اغلب در یک واحد جدا انجام می‌شود، و گاهی نیز واحد شیفت جزئی از ALU می‌باشد.

مدارهای حساب، منطق و شیفت که در بخش قبل معرفی شدند را می‌توان بصورت ALU واحد با متغیرهای انتخاب مشترک با هم ترکیب کرد. یک طبقه از واحد حساب، منطق و شیفت در شکل ۱۳-۴

1- Arithmetic and Logic Unit



شکل ۱۳-۴ یک طبقه از واحد حساب، منطق و شیفت

دیده می‌شود. زیرنویس i نمایانگر یک طبقه نمونه است. ورودی‌های A_i و B_i به هر دو مدار حساب و منطق اعمال شده‌اند. یک ریزعمل خاصی با ورودی‌های S_1 و S_0 انتخاب می‌شوند. یک مولتی‌پلکسر 4×1 در خروجی، خروجی حسابی E_i یا منطقی H_i را برمی‌گزینند. داده‌ها در این مولتی پلکسر با S_2 و S_3 انتخاب می‌شوند. دو ورودی دیگر داده به مولتی پلکسر، A_{i-1} را برای عمل شیفت به راست و A_{i+1} را برای عمل شیفت به چپ دریافت می‌کند. توجه کنید که دیاگرام تنها یک طبقه را نشان می‌دهد. برای یک ALU با ساختمان n بیتی مدار شکل ۱۳-۴ باید n بار تکرار شود. خروجی نقلی C_{i+1} برای یک طبقه ALU مفروض باید به رقم نقلی C_i طبقه سری بعدی وصل شود. رقم نقلی ورودی به اولین طبقه، C_{in} است که یک متغیر انتخاب برای اعمال حسابی خواهد بود.

مداری که یک طبقه‌اش در شکل ۱۳-۴ مشخص شد. یک عمل حسابی هشت بیتی، چهار عمل

منطقی، و دو عمل شیفت را اجرا می‌نماید. هر عمل توسط پنج متغیر S_3، S_2، S_1، S_0 و C_{in} انتخاب می‌گردد. رقم نقلی ورودی فقط برای انتخاب اعمال حسابی بکار می‌رود.

جدول ۸-۴ چهارده عمل ALU را لیست نموده است. هشتای اول اعمال حسابی هستند. (جدول ۴-۴ را ببینید) و توسط $S_3S_2=00$ انتخاب می‌شوند. چهار عمل بعدی منطقی هستند (جدول ۱۰-۴ را ببینید) و بوسیله $S_3S_2=01$ انتخاب می‌گردند. ورودی نقلی در طول اعمال منطقی هیچ اثری ندارد و با علامت بی‌اهمیت X مشخص شده است. دو عمل آخر اعمال شیفت هستند که با $S_3S_2=10$, 11 انتخاب می‌شوند. سه ورودی انتخاب دیگر تاثیری برروی شیفت ندارند.

جدول ۸-۴ جدول تابع برای واحد حساب، منطق و شیفت

انتخاب کننده عمل						
S_3	S_2	S_1	S_0	C_{in}	عمل	تابع
0	0	0	0	0	$F = A$	انتقال
0	0	0	0	1	$F = A + 1$	افزایش
0	0	0	1	0	$F = A + B$	جمع
0	0	0	1	1	$F = A + B + 1$	جمع با رقم نقلی
0	0	1	0	0	$F = A + \overline{B}$	تفریق با قرض
0	0	1	0	1	$F = A + \overline{B} + 1$	تفریق
0	0	1	1	0	$F = A - 1$	کاهش A
0	0	1	1	1	$F = A$	انتقال A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	متمم کردن A
1	0	×	×	×	$F =$ shr A	شیفت A به راست و به داخل F
1	1	×	×	×	$F =$ shl A	شیفت A به چپ و به داخل F

مسائل

۴-۱ بلاک دیاگرام سخت افزاری مشابه شکل ۴-۲ (الف) که عبارت انتقال ثبات زیر را پیاده سازی کنید را رسم کنید:

$$yT_2: \quad R2 \leftarrow R1 \quad , R1 \leftarrow R2$$

۴-۲ خروجی چهارثبات R0، R1، R2 و R3 از طریق یک مولتی پلکسر 4×1 به ورودی‌های ثبات پنجم R5 وصل شده‌اند. هر ثبات هشت بیتی است. انتقالات لازم توسط چهار متغیر زمانبندی T_0 تا T_3 مطابق زیر تعیین می‌شود

$$T_0 : R5 \leftarrow R0$$

$$T_1 : R5 \leftarrow R1$$
$$T_2 : R5 \leftarrow R2$$
$$T_3 : R5 \leftarrow R3$$

متغیرهای زمانی دو به دوازهم جدا هستند. یعنی در هرزمان معین فقط یک متغیر 1 است، در حالی که سه متغیر دیگر 0 هستند. بلاک دیاگرامی رسم کنید که نشان دهنده سخت افزار پیاده سازی انتقال ثبات باشد. اتصالات لازم برای چهار متغیر زمانبندی به خطوط انتخاب مولتی پلکسر و ورودی بارکردن ثبات R5 را نیرنشان دهید.

۳-۴ عبارت کنترل شرطی زیر را بوسیله دو عبارت انتقال ثبات با توابع کنترل نشان دهید.

$$\text{If}(P=1)\ \text{then}\ (R1 \leftarrow R2)\ \text{else if}\ (Q=1)\ \text{then}\ (R1 \leftarrow R3)$$

۴-۴ برای اینکه سیستم گذرگاه شکل ۳-۴ اطلاعات را از هر ثبات به هر ثبات دیگر منتقل کند چه باید کرد؟ خصوصاً اتصالاتی راکه باید برای تهیه یک مسیر از خروجی‌های ثبات C به ورودی‌های ثبات A وصل نمود نشان دهید.

۵-۴ یک سیستم گذرگاه مانند آنچه در شکل ۳-۴ دیده شد رسم کنید، ولی از بافرهای سه حالته و دیکدر بجای مولتی پلکسر استفاده نمائید.

۶-۴ یک کامپیوتر دیجیتال دارای سیستم گذرگاه مشترک برای 16 ثبات 32 بیتی است. گذرگاه با مولتی‌پلکسر ساخته شده است.

الف) در هر مولتی پلکسر چند ورودی انتخاب وجود دارد.

ب) اندازه مولتی پلکسر چیست

ج) چند مولتی پلکسر در گذرگاه است

۷-۴ عبارات زیر انتقال در یک حافظه را مشخص می‌کنند. در هر حالت عمل حافظه را توضیح دهید.

الف) $R2 \leftarrow M[AR]$ ب) $M[AR] \leftarrow R3$ ج) $R5 \leftarrow M[R5]$

۸-۴ بلاک دیاگرام سخت افزاری را رسم کنید که عبارات زیر را پیاده سازی کند.

$$x+yZ : AR \leftarrow AR+BR$$

که AR و BR دو ثبات n بیت و x و y و z متغیرهای کنترل‌اند. برای تابع کنترل گیت‌ها را نیز نشان دهید. (بیاد داشته باشید که سمبل + در بخش کنترل عمل OR و در ریزعمل جمع بمعنی بعلاوه است.)

۹-۴ سخت افزاری رانشان دهید که عبارت زیر را پیاده سازی کند. برای تابع کنترل گیت‌ها و برای شمارنده دودویی با ورودی فعال‌ساز (توانا‌ساز) شمارش، بلاک دیاگرام را رسم کنید.

$$xyT_0 + T_1 + y' T_2 : AR \leftarrow AR+1$$

۴-۱۰ عبارت انتقال ثبات زیر را برای دو ثبات 4 بیتی R1 و R2 ملاحظه کنید

$$xT: R1 \leftarrow R1 + R2$$
$$x'T: R1 \leftarrow R2$$

هربار که T=1 است، اگر x=1 باشد. محتوای R2 به R1 اضافه می‌شود و اگر x=0 باشد R2 به R1 انتقال می‌یابد. دیاگرامی رسم کنید که پیاده سازی سخت افزاری دو عبارت را نشان دهد. برای دو ثبات 4 بیتی، یک جمع کننده 4بیت، و یک مولتی پلکسر چهارتایی 2:1 که ورودی‌ها R1 را انتخاب می‌کند از بلاک دیاگرام استفاده کنید. در دیاگرام نشان دهید که متغیرهای کنترلی x و T چگونه ورودی‌های مولتی‌پلکسر و ورودی بارکردن ثبات R_1 را انتخاب می‌کنند.

۴-۱۱ بااستفاده از یک شمارنده 4 بیتی با بارشدن موازی مانند شکل ۱۱-۲ و یک جمع کننده 4 بیتی مانند شکل ۶-۴، یک بلاک دیاگرام رسم کرده و چگونگی پیاده سازی عبارت‌های زیررا نشان دهید.

$$x: R_1 \leftarrow R_1 + R_2 \quad R_2 \text{را به } R1 \text{ اضافه کن}$$
$$x'y: R_1 \leftarrow R_1 + 1 \quad R_1 \text{ را افزایش بده}$$

که R1 یک شمارنده با بارشدن موازی و R2 یک ثبات 4 بیت است.

۴-۱۲ مدار جمع - تفریق کننده شکل ۴-۷ دارای مقادیر زیر برای ورودی مد M[1] و ورودی‌های داده A و B می‌باشد. در هر حالت، مقادیر خروجی S_3، S_2، S_1، S_0 و C_4 را معین کنید.

	M	A	B
الف	0	0111	0110
ب	0	1000	1001
ج	1	1100	1000
د	1	0101	1010
ه	1	0000	0001

۴-۱۳ یک مدار ترکیبی کاهش گر چهاربیت با چهار مدار تمام جمع کننده بسازید.

۴-۱۴ فرض کنید که مدار چهاربیت شکل ۹-۴ در یک مدار مجتمع قرار گرفته باشد. اتصالات لازم برای دو IC از این نوع را برای ساختن یک مدار حسابی 8 بیتی نشان دهید.

۴-۱۵ یک مدار حسابی با یک متغیر انتخاب s و دو خط ورودی n بیتی A و B طراحی کنید. مدار چهار عمل حسابی زیر را با توجه به نقلی ورودی C_{in} تولید می‌کند. بلاک دیاگرام را برای دو طبقه اول مدار رسم کنید.

1- Mode

S	$C_{in} = 0$	$C_{in} = 1$	
0	D=A+B(جمع)	D = A+1	(افزایش)
1	D=A−1(کاهش)	$D = A+\bar{B}+1$	(تفریق)

۴-۱۶ مدار ترکیبی انتخاب و تولید هر یک از 16 تابع منطقی جدول ۴-۵ را رسم کنید.

۴-۱۷ یک مدار دیجیتال که چهار عمل منطقی OR انحصاری، NOR انحصاری ، NOR و NAND را انجام دهد طراحی کنید. دو متغیر انتخاب بکار برید. دیاگرام منطقی یک طبقه نمونه را نشان دهید .

۴-۱۸ ثبات A عدد هشت بیتی 11011001 را حفظ می‌کند. عملوند B و ریزعمل منطقی لازم را برای تغییر A به هریک از حالات زیر معین کنید

الف) 01101101 (ب) 11111101

۴-۱۹ ثبات‌های 8 بیتی AR ، BR ، CR و DR ابتداً دارای مقادیر زیرند

AR = 11110010

BR = 11111111

CR = 10111001

DR = 11101010

محتوای هشت بیتی هر یک از ثبات‌ها را پس از اجرای هر یک از رشته ریز عمل‌ها مشخص‌کنید.

BR را با هم جمع کن AR ← AR+BR

DR و CR راه AND کن، BR را افزایش بده CR ← CR∧ DR , BR ← BR+1

CR را از AR کم کن AR ← AR − CR

۴-۲۰ یک ثبات هشت بیتی حاوی عدد دودویی 10011100 است. پس از یک شیفت به راست حسابی مقدار ثبات چقدر است؟ با شروع از مقدار اولیه 10011100، مقدار ثبات را پس از یک شیفت به چپ حسابی معین کنید، و بگوئید آیا سرریز وجود دارد؟

۴-۲۱ با شروع از مقدار اولیه R=11011101 رشته مقادیر دودویی را در R پس از یک شیفت به چپ منطقی و بدنبال آن یک شیفت به راست چرخشی، و سپس با یک شیفت به راست منطقی و نهایتاً یک شیفت به چپ چرخشی تعیین کنید.

۴-۲۲ مقدار H در شکل ۴-۱۲ چیست بشرطی که A=1001، S=1، I_R=1 و I_L=0 باشد؟

۴-۲۳ چه چیزی در عبارات انتقال ثبات زیر غلط است

الف) xT: AR ← $\overline{AR}$, AR ← 0 ب) yT: R1 ← R2 , R1 ← R3

ج) zT: PC ← AR , PC ← PC + 1

۵ سازمان و طراحی یک کامپیوتر پایه

۵-۱ کدهای دستورالعمل‌ها
۵-۲ ثبات‌های کامپیوتر
۵-۳ دستورالعمل‌های کامپیوتر
۵-۴ زمانبندی و کنترل
۵-۵ سیکل دستورالعمل
۵-۶ دستورالعمل‌های ارجاع به حافظه
۵-۷ ورودی - خروجی و وقفه
۵-۸ تشریح کامل کامپیوتر
۵-۹ طراحی کامپیوتر پایه
۵-۱۰ طراحی مدار منطقی انباره

۵-۱ کدهای دستورالعمل‌ها

در این فصل یک کامپیوتر پایه[1] را معرفی می‌کنیم و نشان خواهیم داد که چگونه می‌توان عملکرد آن را توسط عبارات انتقال ثبات مشخص نمود. سازمان کامپیوتر بوسیله ثبات های داخلی‌اش، زمانبندی و ساختار کنترل، و مجموعه دستوراتی که به کار می‌برد تعریف می‌گردد. پس از آن طراحی کامپیوتر مشروحاً انجام شده است. هرچند که کامپیوتر پایه‌ای که در این فصل ارائه شده در مقایسه با کامپیوترهای تجاری بسیار کوچک است، ولی مزیت سادگی آن ما را در ارائه روند طراحی بدون برخورد با اشکالات متعدد، قادر می‌سازد.

سازمان داخلی یک سیستم دیجیتال با رشته ریزعملهایی[2] که روی داده‌های ذخیره شده در ثباتهایش انجام می‌دهد تعریف می‌شود. کامپیوتر دیجیتال همه منظوره[3] قادر به اجرای انواع ریزعملها بوده و بعلاوه می‌توان اجرای عملیاتی با توالی خاص را به آن دیکته کرد. کاربر یک کامپیوتر می‌تواند کنترل پردازش را با استفاده از یک برنامه بدست بگیرد. برنامه هم مجموعه‌ای از دستورالعمل‌هاست که اعمال، عملوندها و توالی رخداد پردازش را مشخص می‌کند. کار پردازش داده را ممکن است بوسیله برنامه جدیدی با دستورات متفاوت و یا با همان دستورات ولی با داده‌های مختلف تغییر داد.

1- Basic Computer 2- Microoperation
3- Generol Purpose Digital Computer

دستورالعمل کامپیوتر یک کد دودویی است که رشته‌ای از ریزاعمال را برای کامپیوتر مشخص می‌کند. کدهای دستورات همراه با داده‌هایشان در حافظه ذخیره می‌شوند. کامپیوتر هر دستور را ازحافظه خوانده و آن را در یک ثبات کنترل قرار می‌دهد. پس از آن واحد کنترل کد دودویی دستورالعمل را تفسیر می‌کند و بدنبال آن با صادر کردن رشته‌ای از ریزعمل‌ها آن را اجرا می‌نماید. هر کامپیوتر مجموعه دستورالعمل‌های خاص خود را دارد. توانایی در ذخیره و اجرای دستورات، یا مفهوم توانایی برنامه ذخیره شده، مهمترین خاصیت یک کامپیوتر همه منظوره است.

یک کد دستورالعمل مجموعه‌ای از بیت‌هاست که انجام یک عمل خاص را به کامپیوتر فرمان می‌دهد. این کد معمولاً به دوقسمت تقسیم می‌شود و هر یک تفسیر خاص خود را داراست. اصلی ترین بخش کد دستور بخش عمل آنست. کد عمل در یک دستور گروهی ازبیت‌هاست که اعمالی مانند جمع ، تفریق، ضرب، شیفت و متمم سازی را تعریف می‌نمایند. تعداد بیت‌های مربوط به عمل یک دستور به تعداد کل اعمال موجود در کامپیوتر بستگی دارد. برای 2^n عمل مستقل از هم (یا کمتر)، کد عمل حداقل n بیتی است. بعنوان توضیحی بیشتر در این مورد، کامپیوتری را با 64 عمل جدا از هم در نظر بگیرید که یکی از آنها عمل ADD باشد. کد عمل شش بیتی و ترکیب 110010 به ADD اختصاص یافته است. وقتی که این کد در واحد کنترل دیکد شود، کامپیوتر سیگنال‌های کنترل خواندن عملوند را از حافظه صادر و آنرا با یک ثبات پردازشگر جمع می‌نماید.

در این مقطع ما باید ارتباط بین یک عمل و یک ریزعمل را در یک کامپیوتر بدانیم. عمل بخشی از دستور ذخیره شده در حافظه است. این بخش که یک کد دودویی است انجام عملی خاص را به کامپیوتر دستور می‌دهد. واحد کنترل دستورالعمل را از حافظه دریافت و بیت‌های عمل را تفسیر می‌کند. سپس رشته‌ای از سیگنال کنترل را برای آغاز اجرای ریزاعمال در ثبات‌های داخلی کامپیوتر صادر می‌نماید. واحد کنترل برای هر کد عمل، رشته‌ای از ریزاعمال لازم را برای تحقق سخت افزاری عمل صادر می‌کند. به همین علت گاهی یک کد عمل را درشت عمل [1] (کلان) هم می‌گویند زیرا مجموعه‌ای از ریزعمل‌ها را مشخص می‌نماید.

بخش عمل در یک کد دستورالعمل، کاری راکه قراراست انجام بگیرد معین می‌کند. این عمل باید برروی داده‌هائی که در ثبات‌های پردازشگر و یا حافظه ذخیره شده‌اند صورت گیرد. بنابراین در کد دستور نه تنها عمل بلکه ثبات و یا حافظه‌هایی که عملوندها در آن‌ها می‌توانند یافت شوند و یا در آنها ذخیره شوند نیز مشخص می‌شود. کلمات حافظه توسط آدرسشان در دستورالعمل معین می‌گردند. ثبات‌های پردازشگر نیز با تخصیص k بیت دودویی دیگر که 2^k ثبات را مشخص می‌کند تعیین می‌شوند. برای تخصیص یک کد دودویی برای دستورات انتخاب‌های متعددی وجود دارد و هر کامپیوتر ساختار کد دستورالعمل خاص خود را داراست. قالب کدهای دستورتوسط طراحان که معماری آن را مشخص می‌نمایند تعیین می‌شود. در این فصل ما قالب کدهای خاصی را برای تشریح سازمان وطراحی کامپیوترهای دیجیتال برمی‌گزینیم .

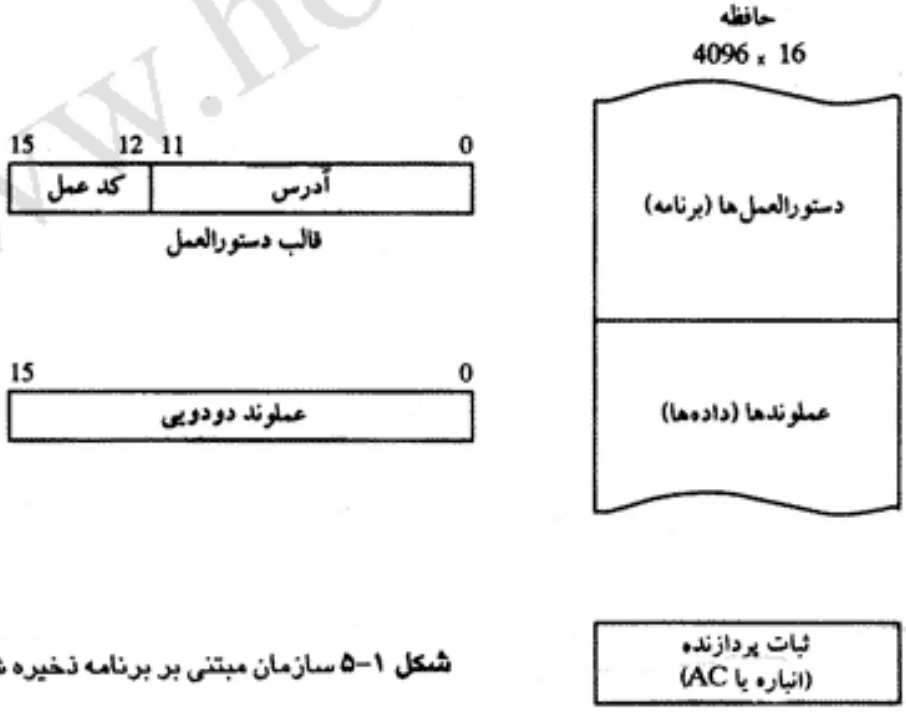
1- Macrooperation

سازمان مبتنی بر برنامه ذخیره شده

ساده‌ترین راه سازماندهی یک کامپیوتر داشتن یک ثبات پردازنده و قالب کد دستورالعملی متشکل از دو بخش است. قسمت اول عملی را که انجام خواهد شد مشخص می‌کند و قسمت دوم آدرس را معین می‌نماید. آدرس به کنترل، محل عملوند[1] را در حافظه نشان می‌دهد. این عملوند از حافظه خوانده شده و بر روی آن بعنوان داده ذخیره شده در ثبات پردازشگر عمل می‌شود.

شکل ۱-۵ این نوع سازماندهی را نشان می‌دهد. دستورالعمل‌ها در یک بخش دیگری ذخیره می‌شوند. برای واحد حافظه‌ای با 4096 کلمه به 12 بیت برای مشخص کردن آدرس نیاز است، زیرا $4096=2^{12}$ می‌باشد. اگر هر کد دستورالعمل در یک کلمه 16 بیتی حافظه جای گیرد، چهار بیت برای کد عمل لازم است تا یکی از 16 عمل ممکن را مشخص نماید و نیز 12 بیت برای مشخص کردن آدرس عملوند خواهیم داشت. کنترل یک کد 16 بیتی دستورالعمل را از آن قسمت از حافظه که برنامه در آن است می‌خواند. سپس از بخش 12 بیتی آدرس در دستورالعمل برای خواندن یک عملوند 16 بیتی از ناحیه داده‌ها استفاده می‌شود. آنگاه عملی را که کد عمل مشخص می‌نماید اجرا می‌کند. در کامپیوترهایی که فقط یک ثبات پردازش دارند نام انباره[2] را به آن اختصاص می‌دهند. اعمال در این کامپیوتر معمولاً بر روی عملوند و محتویات انباره AC صورت می‌گیرد.

اگر عمل مربوط به یک کد دستورالعمل نیاز به عملوند نداشته باشد بیت‌های مربوطه برای اهداف



شکل ۱-۵ سازمان مبتنی بر برنامه ذخیره شده

1- Operand 2-Accumulator

دیگری بکار گرفته می‌شوند. مثلاً، اعمالی مانند پاک کردن AC[1]، متمم کردن AC و افزایش AC روی داده ذخیره شده در AC عمل می‌کنند. این اعمال به عملوندی از حافظه نیازی ندارند. برای این گونه اعمال، بخش دوم کد دستور (بیت‌های 0 تا 11) برای تعیین آدرس حافظه مورد نیاز نیستند و می‌توانند برای اعمال دیگری در کامپیوتر بکار روند.

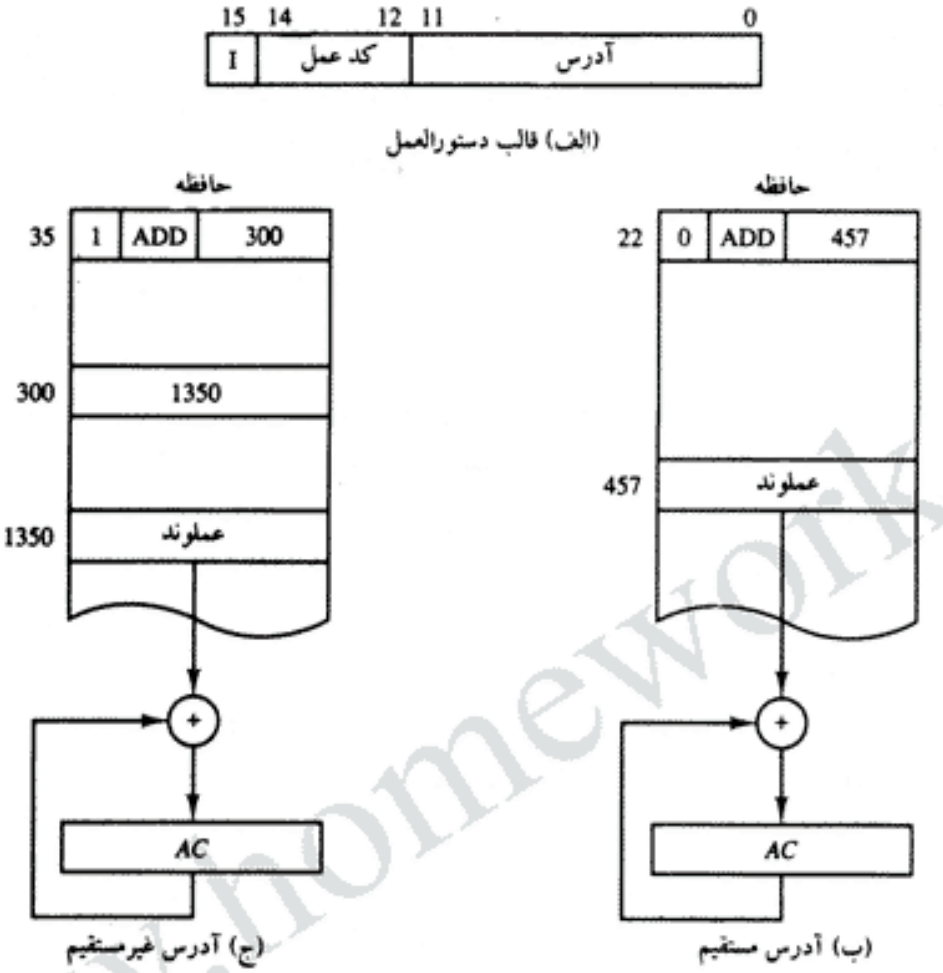
آدرس غیرمستقیم

گاهی اوقات مناسبتر است از بیت‌های آدرس دستورالعمل نه بعنوان آدرس بلکه عملوند واقعی استفاده شود. در اینصورت گوئیم دستورالعمل دارای عملوند بلافصل[2] است. هر گاه این بخش، آدرس عملوند را مشخص کند گوئیم دستور دارای آدرس مستقیم[3] است. دلیل این نام وجودامکان سومی است که آدرس غیرمستقیم[4] نامیده می‌شودو در آن بیت‌های بیت بخش دوم، آدرس کلمه حافظه‌ای را مشخص می‌کنند که آدرس عملوند در آن قرار دارد. یکی از بیت‌های کد دستور را برای تفکیک آدرس مستقیم و غیرمستقیم می‌توان بکار برد.

به منظور توضیح بیشتر، قالب کد دستورالعمل شکل ۲-۵ (الف) را در نظر بگیرید. این دستورمتشکل است از سه بیت کد عملیات، 12 بیت آدرس و یک بیت مربوط به روش آدرس دهی مستقیم یا غیرمستقیم، که با I نشان داده شده است. بیت I برای آدرس دهی مستقیم برابر 0 و برای آدرس دهی غیرمستقیم 1 می‌باشد. در شکل ۲-۵ (ب) دستورالعملی با آدرس مستقیم نشان داده شده است. این دستورالعمل در آدرس 22 از حافظه قرار دارد. بیت I برابر 0 است بنابراین دستور از نوع آدرس مستقیم است. کد عمل مشخص کننده دستور ADD است و بخش آدرس یک عدد دودویی معادل با 457 می‌باشد. واحد کنترل عملوند را در آدرس 457 یافته و آن را با محتوای AC جمع می‌کند. دستورالعمل موجود در آدرس 35 در شکل ۲-۵ (ج) دارای بیت I=1 است. بنابراین این دستور بعنوان یک دستورالعمل با آدرس غیرمستقیم شناخته می‌شود. بخش آدرس عدد دودویی معادل 300 است. کنترل به آدرس 300 مراجعه می‌نماید تا آدرس عملوند را بیابد. آدرس عملوند در این مورد 1350 است. سپس عملوند حاصل از آدرس 1350 با محتوای AC جمع می‌شود. لذا دستورالعمل‌ها با آدرس غیرمستقیم برای برداشتن عملوند نیاز به دو ارجاع به حافظه دارند. اولین ارجاع برای خواندن آدرس عملوند؛ و دومی برای خواندن خود عملوند است. ما آدرس موثر را در دستورات محاسباتی همان آدرس عملوند و در دستورالعمل‌های انشعاب آدرس هدف یا مقصد[5] تعریف می‌کنیم. بنابراین آدرس موثر در دستورالعمل شکل ۲-۵ (ب) عبارتست از 457 و در دستورشکل ۲-۵ (ج) برابر 1350 می‌باشد.

روش‌های آدرس دهی مستقیم و غیرمستقیم در کامپیوتری که دراین بخش ارائه شده بکار رفته‌اند. کلمه‌ای از حافظه برای نگهداری آدرس عملوند در دستورالعملی که با آدرس غیرمستقیم است بعنوان

1- Clear
2- Immediate
3- Direct Address
4- Indirect Address
5- Destination



شکل ۲-۵ نمایش آدرس مستقیم و غیرمستقیم

اشاره‌گری به آرایه داده بکار می‌رود. همانطور که در کامپیوترهای تجاری مرسوم است این اشاره‌گر را می‌توان بجای قرار دادن در یک حافظه، در یک ثبات پردازشگر قرار داد.

۲-۵ ثبات‌های کامپیوتر

دستورالعمل‌های کامپیوتر معمولاً در مکان‌هایی از حافظه بطور متوالی ذخیره شده و هر یک بنوبت اجرا می‌گردند. واحد کنترل یک دستور را از آدرس خاصی در حافظه خوانده و آن را اجرا می‌کند. سپس بهمین ترتیب با خواندن دستوربعدی و اجرای آن، روند را ادامه می‌دهد. اینگونه توالی در خواندن و اجرای دستورات، به شمارنده نیاز دارد تا آدرس دستورالعمل بعدی را پس از اتمام اجرای دستور جاری محاسبه نماید. همچنین یک ثبات در واحد کنترل برای ذخیره کد دستورالعمل، پس از خواندن آن از حافظه، لازم است. بعلاوه کامپیوتر به ثبات‌هایی در پردازشگر جهت دستکاری داده‌ها و نیز یک ثبات برای ذخیره‌سازی آدرس حافظه نیاز دارد. این نیازها، وجود ثبات‌های شکل ۳-۵ را در سیستم الزامی می‌دارد.

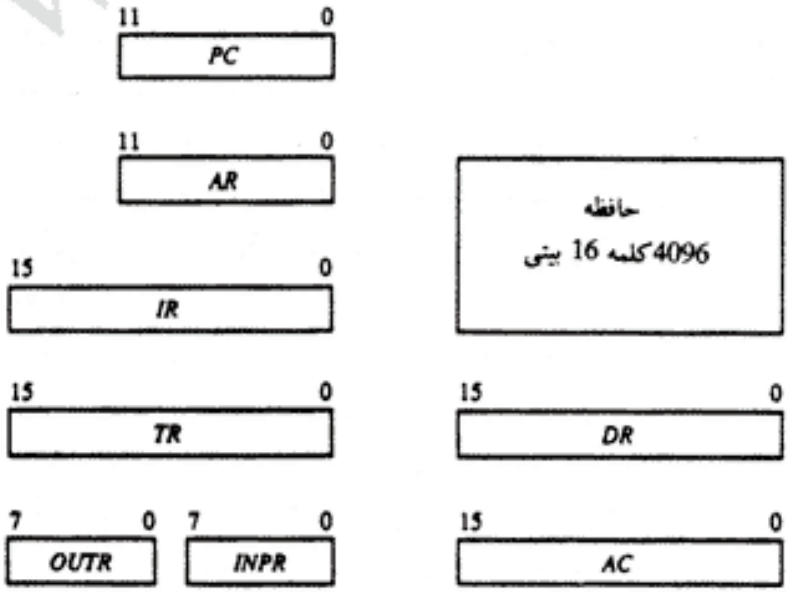
این ثبات‌ها در جدول ۱-۵ هم همراه با شرحی مختصر از کارشان و تعداد بیت‌های آنها آورده شده‌اند.

واحد حافظه 4096 کلمه‌ی 16 بیتی ظرفیت دارد. دوازده بیت از کلمه برای مشخص کردن آدرس عملوند لازم است. لذا سه بیت برای بخش عمل دستور و یک بیت برای تعیین مستقیم یا غیرمستقیم بودن آدرس باقی می‌ماند. ثبات داده (DR) عملوند خوانده شده از حافظه را نگه می‌دارد. ثبات انباره (AC) یک ثبات همه منظوره در پردازش است. دستور خوانده شده از حافظه درثبات دستورالعمل (IR) قرار می‌گیرد. ثبات موقت (TR) برای نگهداری موقت داده‌ها در حین پردازش به کار می‌رود.

ثبات آدرس حافظه (AR) 12 بیتی است زیرا عرض آدرس حافظه برابر دوازده است. شمارنده برنامه (PC) نیز 12 بیت دارد و آدرس دستورالعمل بعدی را که پس از اجرای دستور جاری خوانده خواهد شد

جدول ۱-۵ لیست ثابت‌های کامپیوتر ساده

وظیفه	نام ثبات	تعداد بیت‌ها	سمبل ثبات
نگهداری عملوند حافظه	ثبات داده	16	*DR*
نگهداری آدرس حافظه	ثبات آدرس	12	*AR*
ثبات پردازنده	انباره	16	*AC*
نگهداری کد دستور	ثبات دستورالعمل	16	*IR*
نگهداری آدرس دستور	شمارنده برنامه	12	*PC*
نگهداری داده‌های موقت	ثبات موقت	16	*TR*
نگهداری کاراکتر ورودی	ثبات ورودی	8	*INPR*
نگهداری کاراکتر خروجی	ثبات خروجی	8	*OUTR*



شکل ۳-۵ ثبات‌ها و حافظه کامپیوتر پایه

نگه‌می‌دارد. PC یک رشته شمارشی را طی می‌نماید و سبب می‌شود تا کامپیوتر دستورات از قبل ذخیره شده در حافظه را متوالیاً بخواند. کلمات دستورالعمل یکی پس از دیگری خوانده شده و اجرا می‌شوند مگر اینکه به دستورانشعاب در برنامه برخورد شود. دستورانشعاب موجب انتقال به یک دستور غیرمتوالی در برنامه که در مکانی دیگر است می‌گردد. در تحقق این انتقال، بخش آدرس دستورانشعاب به PC منتقل می‌شود تا آدرس دستور قابل اجرای بعدی باشد. برای خواندن یک دستورالعمل، محتوای PC به عنوان آدرس به حافظه ارسال شده و بدین ترتیب سیکل خواندن از حافظه آغاز می‌شود. سپس PC به میزان یک واحد افزایش می‌یابد تا آدرس دستور بعدی تولید گردد.

دو ثبات برای ورودی و خروجی بکار رفته‌اند. ثبات ورودی (INPR) یک کاراکتر هشت بیتی را از یک وسیله ورودی دریافت می‌کند. ثبات خروجی (OUTR) یک کاراکتر هشت بیتی را برای یک وسیله خروجی نگه می‌دارد.

سیستم گذرگاه مشترک

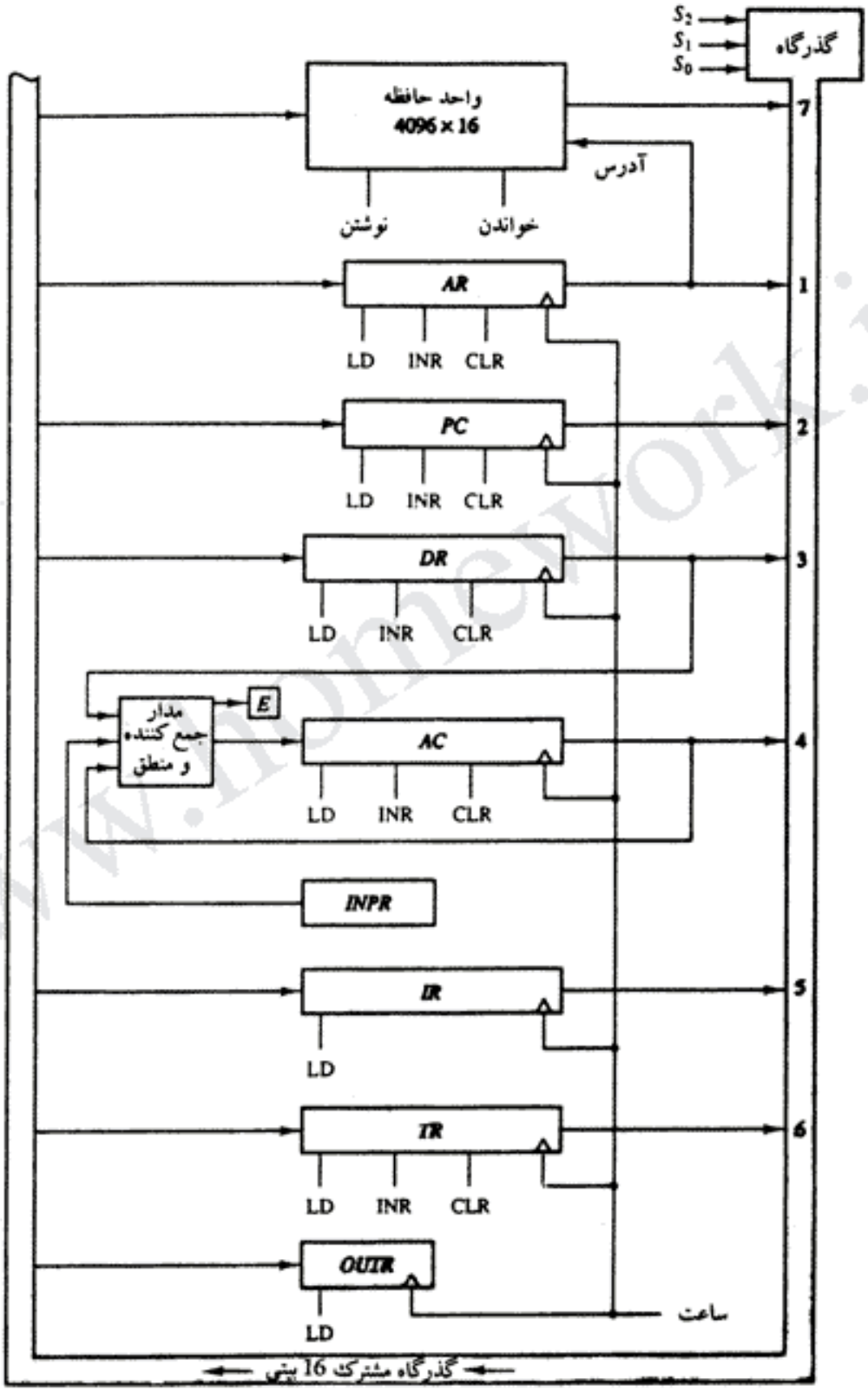
کامپیوتر مورد نظر دارای هشت ثبات، یک واحد حافظه و یک واحد کنترل، (که در بخش ۴-۵ ارائه خواهد شد) می‌باشد. برای انتقال اطلاعات از یک ثبات به ثبات دیگر و یا بین حافظه و ثبات‌ها، مسیرهایی باید ایجاد شود. اگر بین خروجی هر ثبات با ورودی ثبات دیگری اتصال برقرار شود، تعداد سیم‌ها بیش از حد زیاد خواهد شد. روش بهتر در انتقال اطلاعات در یک سیستم که دارای تعداد قابل توجهی ثبات است استفاده از گذرگاه مشترک است. دربخش ۳-۴ نشان دادیم که چگونه با استفاده از مولتی پلکسر یا بافرهای سه حالته می‌توان یک گذرگاه سیستم ساخت. اتصالات ثبات‌ها و حافظه‌های کامپیوتر مورد نظر به گذرگاه مشترک فوق در شکل ۴-۵ نشان داده شده است.

خروجی‌های هفت ثبات و حافظه‌ها به گذرگاه مشترک متصل شده‌اند. در هر مقطع زمانی، خروجی خاصی که برای خطوط گذرگاه انتخاب می‌شود توسط مقادیر دودویی انتخاب[1] S_0، S_1 و S_2 معین می‌گردد. مقادیر دهدهی که در امتداد هر خروجی نوشته شده معادل عدد دودویی لازم را برای انتخاب نشان می‌دهد. مثلاً در کنار خروجی DR عدد 3 نوشته شده است، لذا 16 بیت خروجی DR هنگامی روی گذرگاه قرار می‌گیرد که $S_2S_1S_0 = 011$ باشد زیرا این مقدار معادل دودویی عدد 3 است. خطوط گذرگاه مشترک به ورودی‌های همه ثبات‌ها و ورودی داده حافظه‌ها نیز وصل شده‌اند.

هر ثبات خاصی که ورودی LD آن فعال باشد، داده‌ها را با پالس بعدی از گذرگاه دریافت می‌کند. حافظه هم بهنگام فعال شده ورودی "نوشتن"[2] محتوای گذرگاه را دریافت می‌نماید. هنگام فعال شدن ورودی "خواندن"[3] حافظه، خروجی آن روی گذرگاه قرار می‌گیرد.

چهار ثبات DR، AC، IR و TR هر یک 16 بیتی هستند. دو ثبات AR و PC هر یک 12 بیت دارند زیرا آدرس حافظه را نگه می‌دارند. وقتی که محتوای AR یا PC به گذرگاه مشترک 16 بیتی اعمال شود،

1- Select 2- Write 3- Read



شکل ۴-۵ ثبات‌های کامپیوتر پایه متصل به یک گذرگاه مشترک

چهار بیت بالا رتبه 0 می‌شوند. وقتی که AR یا PC اطلاعات را از گذرگاه دریافت می‌کنند، تنها 12 بیت پائین رتبه گذرگاه به داخل ثبات منتقل می‌گردد.

ثبات ورودی INPR و OUTR هر یک هشت بیت داشته و با هشت بیت پایین رتبه در گذرگاه ارتباط برقرار می‌کنند. INPR برای گذرگاه، اطلاعات را از خارج تهیه می‌کند ولی OUTR فقط اطلاعات را از گذرگاه دریافت می‌نماید. دلیل این است که INPR کاراکتری را از وسیله خارجی دریافت می‌کند که باید بعداً به AC انتقال یابد. بالعکس OUTR کاراکتر را از AC دریافت و به وسیله خارجی تحویل می‌دهد. ضمناً هیچگونه انتقالی از OUTR به ثبات‌های دیگر صورت نمی‌گیرد.

شانزده خط گذرگاه مشترک، اطلاعات را از شش ثبات و حافظه دریافت می‌کنند. خطوط گذرگاه به ورودی شش ثبات و حافظه متصلند. پنج ثبات دارای سه ورودی کنترل می‌باشند: LD (بارشدن)[1] INR (افزایش)[2] و CLR (پاک شدن)[3]. این نوع ثبات معادل یک شمارنده دودویی با بارشدن موازی و نیز پاک کننده همزمان[4] مطابق شکل ۱۱-۲ است. عمل افزایش با فعال شدن ورودی "شمارش"[5] شمارنده میسر است. دو ثبات هر یک فقط دارای یک ورودی LD می‌باشند. این نوع ثبات در شکل ۷-۲ نشان داده شد.

ورودی داده‌ها و خروجی داده‌های حافظه به گذرگاه مشترک متصل‌اند. ولی ورودی آدرس حافظه به AR متصل است. بنابراین از AR همیشه باید برای معین کردن آدرس حافظه استفاده شود. با استفاده از یک ثبات واحد برای آدرس، نیاز به یک گذرگاه آدرس که در غیراینصورت لازم بود از بین می‌رود. هنگام عمل نوشتن، محتوای هر ثبات می‌تواند بعنوان داده ورودی حافظه مشخص شود. به طور مشابه، هر ثباتی، بجز AC، می‌تواند داده‌های حافظه را پس از عمل خواندن دریافت کند.

شانزده ورودی AC از یک مدار جمع کننده و منطقی می‌آیند. این مدار جمع کننده دارای سه مجموعه ورودی است. یک مجموعه از ورودی‌ها از خروجی AC می‌آید. این خطوط برای پیاده سازی ریزعمل‌هایی[6] مانند متمم سازی AC و شیفت AC بکارمی‌روند. مجموعه ورودی 16 بیتی دیگری از ثبات DR می‌آید. ورودی‌هایی که از DR و AC می‌آیند برای ریزعملهای حسابی و منطقی مانند جمع DR با AC یا AND کردن DR با AC بکار می‌روند. نتیجه جمع به AC و رقم نقلی نهایی[7] به فلیپ فلاپ E (بیت گسترش[8] AC) منتقل می‌شوند. سومین مجموعه هشت بیتی به ثبات ورودی INPR وصل است. عملکرد INPR و OUTR در بخش ۷-۵ شرح داده خواهد شد.

توجه کنید که دریک سیکل ساعت، می‌توان هم محتوای هر ثبات را روی گذرگاه قرار داد و هم بطور همزمان یک عمل را در جمع کننده و مدار منطقی اجرا نمود. لبه پالس سیکل ساعت محتوای گذرگاه را به ثبات معین شده و خروجی جمع کننده و مدار منطقی را به AC منتقل می‌نماید. مثلاً دو

1- Load
2- Increment
3- Clear
4- Synchronous
5- Count
6- Microperation
7- End carry out
8- Extended Ac bit

ریزعمل

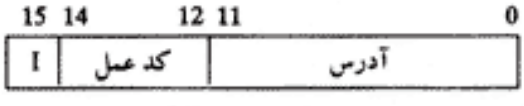
$$DR \leftarrow AC \text{ و } AC \leftarrow DR$$

می‌توانند هم زمان اجرا شوند. این اعمال با قرار گرفتن محتوای AC بر روی گذرگاه ($S_2S_1S_0$=100یا)، فعال شدن LD (بارشدن) در ورودی DR، و همزمان با آن انتقال محتوای DR از طریق مدار جمع کننده و منطقی، و فعال شدن LD در ورودی AC انجام می‌شوند. دو انتقال فوق با وقوع لبه پائین رونده پالس ساعت رخ می‌دهد.

۵-۳ دستورالعمل‌های کامپیوتر

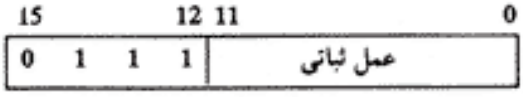
کامپیوتر پایه مورد بحث دارای سه قالب دستورالعمل مطابق شکل ۵-۵ می‌باشد. این قالب‌ها 16 بیتی هستند. بخش عمل کد دستور سه بیتی است و مفهوم بقیه بیت‌ها به نوع کد وابسته است. دستورالعمل‌های ارجاع به حافظه از 12 بیت برای تعیین آدرس و 1 بیت برای مشخص کردن روش آدرس دهی I استفاده می‌کنند. برای آدرس دهی مستقیم، I برابر 0 و برای آدرس غیرمستقیم I مساوی 1 است، شکل ۲-۵. دستورالعمل‌های ارجاع به ثبات با کد عملی 111 و یک بیت 0 در منتها الیه سمت چپ آن (بیت 15) قابل تشخیص‌اند. دستورالعمل‌های ثباتی عمل را برروی AC و یا هر تستی برروی آن را مشخص می‌نمایند. در این دستورات عملوندی از حافظه موردنیاز نیست؛ بنابراین از 12 بیت باقیمانده برای مشخص کردن عمل و یا تست موردنظر استفاده می‌شود. بطور مشابه، دستورات ورودی - خروجی نیز نیاز به ارجاع به حافظه ندارند و با کد عملیاتی 111 و 1 در بیت منتهاالیه سمت چپ دستور قابل تشخیص‌اند. بقیه 12 بیت نوع عمل ورودی - خروجی و یا تستی که باید انجام شود را مشخص می‌سازد.

واحد کنترل کامپیوتر نوع دستورالعمل را با توجه به بیت‌های مکان‌های 12 تا 15 دستورالعمل تشخیص می‌دهد. اگر سه بیت کد عمل در مکان‌های 12 تا 14 برابر 111 نباشد دستور از نوع ارجاع به حافظه است و بیت 15 بعنوان بیت روش آدرس دهی I در نظر گرفته می‌شود. اگر سه بیت کد عمل 111



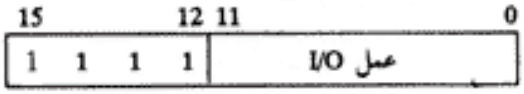
کد عمل از 000 تا 110

(الف) دستورالعمل‌های حافظه‌ای



کد عمل برابر 111، I برابر 0

(ب) دستورالعمل‌های ثبات



کد عمل برابر 111، I برابر 1

(ج) دستورالعمل‌های ورودی - خروجی

شکل ۵-۵ قالب دستورالعمل‌ها در کامپیوتر پایه

باشد، واحد کنترل بیت مکان 15 رابررسی می‌نماید. اگر این بیت 0 باشد، دستور ازنوع ارجاع به ثبات است. اگر بیت مذکور 1 باشد، دستور از نوع ورودی - خروجی است. هنگامی که کد عمل 111 باشد بیت مکان 15 با I مشخص می‌شود ولی بعنوان بیت روش آدرسی دهی بکار نمی‌رود.

گفتیم که برای کد عمل تنها سه بیت به کار رفته است. لذا ممکن است به نظر برسد که کامپیوتر به حداکثر هشت عمل محدود شده است. با این وجود، چون دستورات ارجاع به ثبات و ورودی خروجی از 12 بیت باقیمانده کد عمل استفاده می‌کنند، کل دستورالعمل‌ها از هشت تجاوز می‌کند. در واقع در این حال، دستورالعمل‌های مورد استفاده در کامپیوتر 25 خواهد بود

دستورالعمل‌های کامپیوتر در جدول ۲-۵ ملاحظه می‌شوند. سمبل مشخص کننده دستورالعمل‌ها از یک کلمه سه حرفی تشکیل شده و خلاصه عمل را برای برنامه نویس و کاربر مشخص می‌کنند. کد مبنای

جدول ۲-۵ دستورالعمل‌های کامپیوتر پایه

سمبل	کد شانزده شانزدهی $I = 0$	$I = 1$	شرح
AND	0xxx	8xxx	AND کردن کلمه حافظه با AC
ADD	1xxx	9xxx	جمع کردن کلمه حافظه با AC
LDA	2xxx	Axxx	بار کردن کلمه حافظه در AC
STA	3xxx	Bxxx	ذخیره محتوای AC در حافظه
BUN	4xxx	Cxxx	انشعاب نامشروط
BSA	5xxx	Dxxx	انشعاب و ضبط آدرس بازگشت
ISZ	6xxx	Exxx	افزایش و گذر در صورت نتیجه صفر
CLA	7800		پاک کردن AC
CLE	7400		پاک کردن E
CMA	7200		متمم کردن AC
CME	7100		متمم کردن E
CIR	7080		چرخش AC و E به راست
CIL	7040		چرخش AC و E به چپ
INC	7020		افزایش AC
SPA	7010		گذر از دستور بعدی اگر AC مثبت باشد
SNA	7008		گذر از دستور بعدی اگر AC منفی باشد
SZA	7004		گذر از دستور بعدی اگر AC صفر باشد
SZE	7002		گذر از دستور بعدی اگر E صفر باشد
HLT	7001		توقف کامپیوتر
INP	F800		دریافت کاراکتر و انتقال آن به AC
OUT	F400		برداشتن کاراکتر از AC و انتقال آن به خروجی
SKI	F200		گذر مبتنی بر پرچم ورودی
SKO	F100		گذر مبتنی بر پرچم خروجی
ION	F080		فعال‌کردن وقفه‌ها
IOF	F040		غیرفعال کردن وقفه‌ها

شانزده برابر با عدد معادل شانزده کد دودویی بکار رفته در دستورالعمل است. با استفاده ازمعادل شانزده،ما 16 بیت کد دستورالعمل را به چهار رقم کاهش می‌دهیم که هر رقم معادل چهار بیت است. دستورالعمل ارجاع به حافظه دارای بخش آدرس 12 بیتی است. این بخش از دستور با سه X مشخص شده و به مفهوم سه رقم در مبنای شانزده است. آخرین بیت دستورالعمل با نماد I مشخص گردیده است. هرگاه I=0 باشد چهار بیت آخر دستور یک رقم مبنای شانزده معادل با 0 تا 6 است. وقتی که I=1 باشد چهار بیت آخر دستور یک رقم مبنای شانزده معادل با 8 تا E می‌باشد.

دستورات ارجاع به ثبات برای مشخص کردن عملوند از 16 بیت استفاده می‌نمایند. چهاربیت سمت چپ همیشه 0111 هستند. که معادل 7 درمبنای شانزده می‌باشد. سه رقم دیگر معادل دودویی 12 بیت باقیمانده را در مبنای شانزده نشان می‌دهند. دستورات ورودی - خروجی هم از تمام 16 بیت برای مشخص کردن عمل استفاده می‌کنند. چهاربیت آخر همیشه 1111 می‌باشند که معادل F در مبنای شانزده است.

کامل بودن مجموعه دستورات

قبل از بررسی عمل‌هایی که دستورات انجام می‌دهند، اجازه بدهید درباره نوع دستوراتی که باید در کامپیوتر در نظر داشت بحث کنیم. یک کامپیوتر باید دستورالعمل‌هایی را داشته باشد که کاربر بتواند بکمک آنها برنامه‌هایی بزبان ماشین برای محاسبه توابعی که قابل محاسبه هستند بنویسد. اگر کامپیوتر دارای دستورالعمل‌های کافی در هر یک از رسته‌های زیر باشد مجموعه دستورات آن کافی تلقی می‌گردد:

۱- دستورالعمل‌های حسابی، منطقی و شیفت

۲- دستورالعمل‌هایی برای تبادل اطلاعات با حافظه و ثبات‌های پردازشگر

۳- دستورات کنترل برنامه همراه با دستورالعمل‌هایی که شرایط وضعیتی[1] راچک می‌کنند

۴- دستورات ورودی و خروجی

دستورات حسابی منطقی قابلیت‌های محاسباتی را برای پردازش داده‌هایی که کاربر مایل است بکار برد فراهم می‌آورند. بخش اعظم اطلاعات دودویی در یک کامپیوتر دیجیتال در حافظه ذخیره می‌شود، ولی تمام محاسبات در ثبات‌های‌پردازشگر انجام می‌گردد. بنابراین کاربر باید امکان انتقال اطلاعات را بین دو واحد داشته باشد. تصمیم‌گیری، از جمله قابلیت‌های مهم در کامپیوترهای دیجیتال است. بعنوان مثال دوعدد را می‌توان مقایسه کرد و اگر اولی بزرگتر از دومی باشد ادامه کار ممکن است با حالتی که دومی از اولی بزرگتر است متفاوت باشد. در اینحالت دستورالعمل‌های کنترل برنامه مانند دستورات انشعاب برای تغییر ترتیب اجرای برنامه بکار می‌روند.دستورات ورودی - خروجی برای ارتباط بین کامپیوتر و کاربر لازمند. برنامه‌ها و داده‌ها باید به حافظه منتقل گردند و نتایج محاسبات باید به کاربر بازگردانده شود.

1- Status Condition

دستورات لیست شده در جدول ۲-۵ از حداقل مجموعه‌ای تشکیل یافته که قادر است قابلیت‌های ذکر شده دربالا را فراهم آورد. در این جدول یک دستورالعمل حسابی ADD، دو دستور مرتبط با اعمال حسابی منطقی، یعنی متمم کردن AC (CMA) و افزایش AC (INC) وجود دارد. بااین سه دستور می‌توان عددهای دودویی را جمع و تفریق کرد بشرطی که اعداد منفی بشکل متمم 2 علامت دار نشان داده شوند. با استفاده از دستورالعمل های چرخش CIR و CIL می‌توان شیفت‌های حسابی و یاهر گونه شیفت دیگررا انجام داد. ضرب و تقسیم با استفاده از جمع و تفریق امکان پذیر است. سه عمل منطقی وجود دارد: AND، متمم سازی AC (CMA) و پاک کردن AC (CLA). دستور AND و مکمل با هم امکان NAND را فراهم می‌کنند. می‌توان نشان داد که با عمل NAND انجام کلیه اعمال منطقی دو متغیره امکان پذیر است (جدول ۶-۴).

انتقال اطلاعات از حافظه به AC با دستور بارکردن AC (LDA) انجام می‌شود. ذخیره اطلاعات از AC در حافظه هم توسط دستور ذخیره کردن در AC (STA) صورت می‌گیرد. دستورالعمل‌های انشعاب BUN و BSA و ISZ همراه با چهار دستور گذر[1] امکان کنترل برنامه و بررسی شرایط وضعیتی بوجود می‌آورند. دستورالعمل‌های ورودی INP و خروجی OUT موجب انتقال اطلاعات بین کامپیوتر و وسایل خارجی می‌گردند.

هر چند مجموعه دستورالعمل‌ها برای کامپیوتر پایه نسبتاً کافی است ولی کارا نیست زیرا دستوراتی که کراراً بکار می‌روند بسرعت اجرا نمی‌شوند. مجموعه کارایی از دستورات باید شامل دستورالعمل‌های تفریق، ضرب، OR و XOR[2] باشد. در کامپیوتر پایه برای این اعمال باید برنامه نویسی کرد. این برنامه‌ها برای کامپیوتر پایه به همراه مثالهای برنامه نویسی دیگری در فصل ۶ آورده شده است. با استفاده از تعداد محدودی از دستورالعمل‌ها می‌توان جزئیات طراحی منطقی کامپیوتر را نشان داد. مجموعه کامل‌تری از دستورات سبب می‌شود تا طراحی بیش از حد پیچیده شود. با روش انتخابی، ما اصول سازمان کامپیوتر را نشان داده و طراحی را بدون بحث‌های مفصل دنبال خواهیم کرد. در فصل ۸ لیست کاملی از دستورالعمل‌های کامپیوتر که در اغلب کامپیوترهای تجاری، وجود دارد را ارائه خواهیم نمود.

وظیفه هر دستور در جدول ۲-۵ لیست شده است و ریزعمل‌های لازم برای اجرای هر یک از آنها در بخش ۵-۵ تا ۵-۷ آورده شده است. ما فعلاً بدلیل اولویت در آشنایی با واحد کنترل و سازمان آن، این بحث را بتأخیر می‌اندازیم.

۵-۴ زمانبندی و کنترل

زمانبندی همه ثبات‌ها در کامپیوتر پایه بوسیله یک مولد پالس ساعت اصلی کنترل می‌شود. پالس‌های ساعت به همه فلیپ‌فلاپ‌ها و ثبات‌ها، از جمله آنهایی که در واحد کنترل سیستم قرار دارند

1- Skip 2- Exclusive OR (XOR)

اعمال می‌شوند. پالس‌های ساعت حالت هیچ ثباتی را عوض نمی‌کنند مگر اینکه ثبات توسط یک سیگنال کنترل فعال شود. سیگنال‌های کنترل در واحد کنترل تولید می‌شوند و ورودی‌های کنترل را برای مولتی پلکسر در گذرگاه مشترک، ورودی‌های کنترل در ثبات‌های پردازشگر و ریزاعمال در انباره AC فراهم می‌آورند.

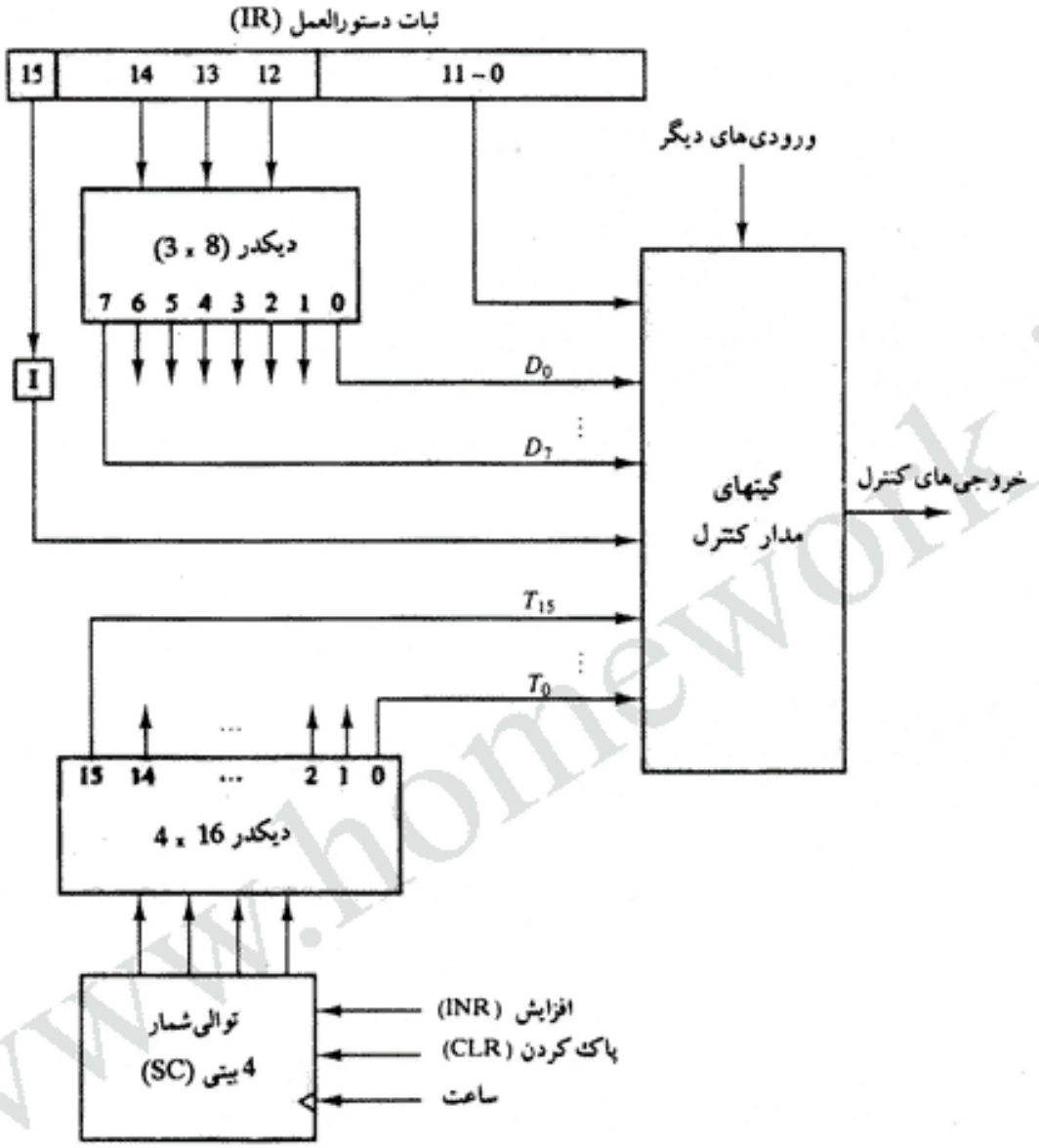
دو نوع سازمان کنترل عمده وجود دارد که عبارتند از: کنترل سخت افزاری و کنترل ریز برنامه نویسی شده. در نوع سخت افزاری، مدار منطقی کنترل با گیت‌ها، فلیپ فلاپ‌ها، دیکدرها و سایر مدارهای دیجیتال پیاده‌سازی می‌شود. حسن این روش داشتن سرعت بالا است که با بهینه کردن آن حاصل می‌شود. در سازمان ریز برنامه نویسی شده اطلاعات کنترل در یک حافظه کنترل ذخیره شده است. حافظه کنترل به صورتی برنامه نویسی می‌شود که رشته ریز عمل‌های لازم را اجرا کند. در کنترل سخت‌افزاری، همانطور که از نامش پیداست، در صورت نیاز به تغییر یا تصحیح آن، باید سیم بندی بین قطعات مختلف را تغییر داد. در کنترل ریزبرنامه‌نویسی شده هر گونه تغییر تنها با تصحیح ریز برنامه در حافظه کنترل امکان پذیر است. در این بخش یک واحد کنترل ریز برنامه نویسی برای کامپیوتر پایه ارائه می‌کنیم. واحد کنترل ریزبرنامه نویسی شده برای کامپیوتری مشابه در فصل ۷ آمده است.

بلاک دیاگرام واحد مزبور در شکل ۶-۵ دیده می‌شود. این واحد از دو دیکدر، یک توالی شمار SC و چند گیت تشکیل شده است. هر دستوری که از حافظه خوانده شود در ثبات دستورالعمل[1] (IR) قرار می‌گیرد. مکان این ثبات در سیستم گذرگاه مشترک در شکل ۴-۵ مشخص شده است. ثبات دستورالعمل که مجدداً در شکل ۶-۵ آورده شده به سه بخش تقسیم شده است: بیت I، کد عمل، و بیت 0 تا 11. کد عمل واقع در بیت‌های 12 تا 14، توسط یک دیکدر 3×8 دیکد می‌شود. هشت خروجی دیکدر با D_0 تا D_7 مشخص شده‌اند. زیر نویس دهدهی مبین مقدار دودویی کد عمل است. بیت 15 دستورالعمل به فلیپ فلاپ I منتقل می‌شود. بیت‌های 0 تا 11 به مدار کنترل اعمال می‌گردند. شمارنده چهاربیتی می‌تواند از 0 تا 15 متوالیاً بشمارد. خروجی‌های این شمارنده به صورت 16 سیگنال زمانبندی T_0 تا T_{15} دیکد می‌شود. مدار داخلی گیت‌های کنترل بعداً بهنگام بررسی مشروحتر طراحی کامپیوتر بدست خواهد آمد.

شمارنده SC را می‌توان بطور همزمان[2] افزایش داد یا پاک کرد (شمارنده شکل ۱۱-۲ ملاحظه شود). اغلب اوقات، شمارنده افزایش می‌یابد تا رشته سیگنالهای زمانبندی متوالی را از خروجی دیکدر ایجاد کند. هر از گاه یکبار این شمارنده 0 می‌شود و سبب می‌گردد تا سیگنال زمانبندی بعدی T_0 شود. مثلاً حالتی را در نظر بگیرید که SC افزایش داده شود و تا سیگنال‌های T_0، T_1، T_2، T_3، T_4 را بطور متوالی فراهم نماید. در T_4، اگر خروجی دیکدر D_3 فعال باشد، SC صفر می‌شود. این موضوع بصورت نمادی با عبارت زیر نشان داده می‌شود:

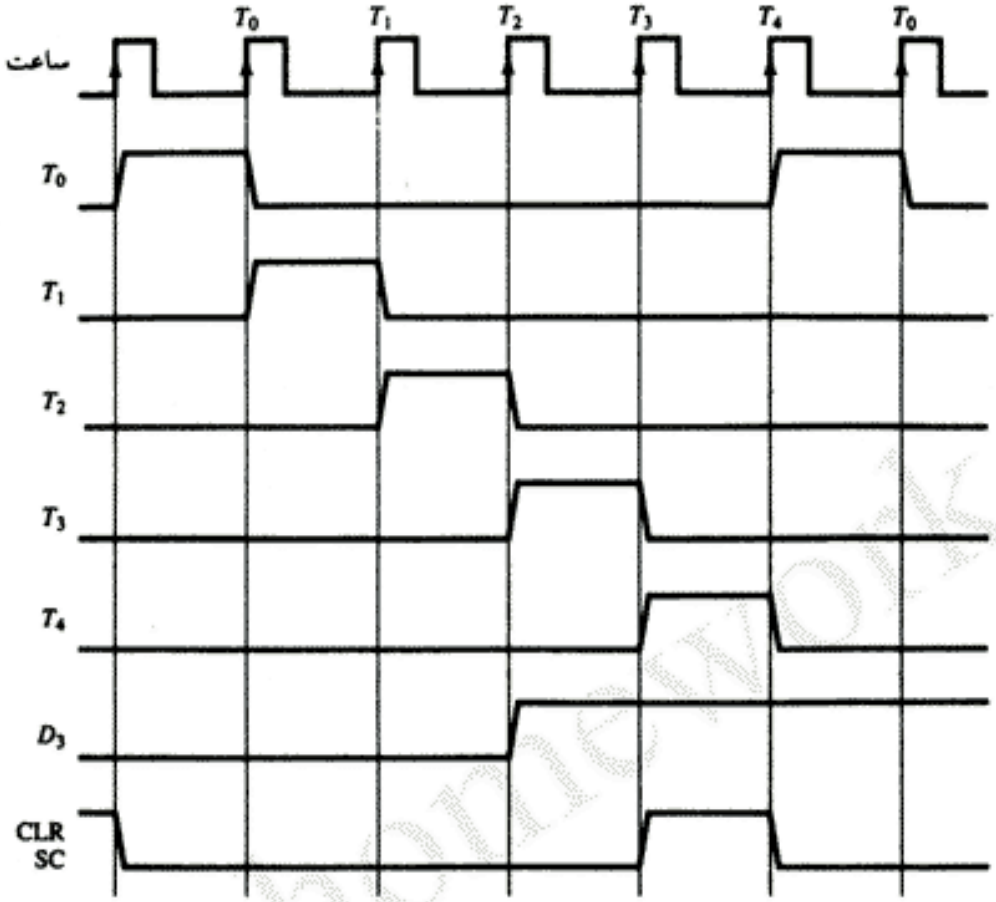
$$D_3\ T_4: SC \leftarrow 0$$

1- Instruction Register 2- Synchronous



شکل ۶-۵ واحد کنترل کامپیوتر

دیاگرام زمانی شکل ۷-۵ ارتباط زمانی سیگنال‌های کنترل را نشان می‌دهد. شمارنده SC بهنگام لبه مثبت پالس ساعت واکنش نشان داده و عمل می‌کند. در آغاز، ورودی CLR شمارنده SC فعال است. اولین لبه مثبت پالس ساعت، SC را 0 می‌کند که این بنوبه خود سیگنال زمانبندی T_0 را که از دیکدر خارج می‌شود فعال می‌نماید. T_0 در طول یک سیکل پالس ساعت فعال باقی می‌ماند. لبه مثبت پالس T_0 فقط آن دسته از ثباتها را فعال می‌کند که ورودی کنترلشان به سیگنال زمانبندی T_0 متصل باشد. SC با هر لبه مثبت پالس ساعت یک واحد افزایش می‌یابد، مگر اینکه CLR آن فعال باشد. این افزایش سبب تولید سیگنال‌های T_0، T_1، T_2، T_3، T_4 و غیره مطابق شکل می‌گردد (به رابطه بین سیگنال



شکل ۷-۵ مثالی از سیگنال‌های زمانبندی واحد کنترل

زمانبندی و لبه مثبت پالس مربوطه توجه کنید). اگر SC صفر نشود، سیگنال‌های زمانبندی با T_5 و T_6 تا T_{15} ادامه یافته و مجدداً به T_0 باز می‌گردند.

سه موج آخر در شکل ۷-۵ نشان می‌دهد که چگونه SC، وقتی که $D_3T_4=1$ است، پاک می‌شود. خروجی D_3 هنگامی فعال می‌شود که سیگنال T_2 پایان یافته باشد. وقتی سیگنال T_4 فعال می‌شود خروجی گیت AND بکار رفته در تابع کنترل D_3T_4 فعال می‌گردد. این به ورودی CLR مربوط به SC اعمال شده است. در لبه مثبت پالس بعد از T_3، یعنی T_4 شمارنده پاک می‌شود. این به نوبه خود سبب می‌شود تا در عوض T_5 که می‌باید با افزایش SC فعال می‌شد، سیگنال T_0 بعلت پاک شدن شمارنده SC تولید شود.

سیکل خواندن یا نوشتن با لبه بالارونده لبه سیگنال زمانبندی آغاز می‌شود. فرض براین است که سیکل زمانی عملکرد حافظه کمتر از سیکل پالس ساعت باشد. با این فرض، سیکل خواندن یا نوشتن توسط یک سیگنال زمانبندی آغاز و در لبه پالس ساعت بعدی خاتمه می‌یابد. سپس لبه پالس برای بارکردن حافظه در یک ثبات بکارمی‌رود. این ارتباط زمانی دربسیاری از کامپیوترها برقرار نیست زیرا معمولاً سیکل زمانی حافظه طولانی‌تر از سیکل پالس ساعت است. در این حالت لازم است زمان

انتظاری[1] در پردازشگر برای فرصت دادن به حافظه ایجاد کرد.تا کلمه حافظه مهیا گردد. برای سهولت در ارائه مطلب ما فرض خواهیم کرد که درکامپیوتر پایه، زمان انتظار لازم نیست.

برای درک کامل عملکرد یک کامپیوتر، باید رابطه زمانی بین لبه‌های پالس ساعت و سیگنال‌های زمانبندی را دانست. بعنوان مثال عبارت انتقال ثبات

$$T_0 : AR \leftarrow PC$$

انتقال محتوای PC را به AR بهنگام فعال شدن T_0 مشخص می‌کند. T_0 در تمام مدت یک سیکل ساعت فعال است. در طول این مدت محتوای PC روی گذرگاه قرار می‌گیرد. (با $S_2S_1S_0=010$) و ورودی بارشدن LD ثبات AR فعال می‌شود. انتقال واقعی تا پایان ساعت و در واقع تا آغاز لبه مثبت پالس بعدی رخ نمی‌دهد. همین لبه SC را از 0000 به 0001 افزایش می‌دهد. پالس ساعت بعدی T_1 را فعال و T_0 را غیرفعال می‌سازد.

۵-۵ سیکل دستورالعمل

برنامه‌ای که در واحد حافظه کامپیوتر ذخیره شده از دنباله‌ای از دستورالعمل‌ها تشکیل شده است. این برنامه طی سیکل‌های اجرایی دستورالعمل اجرا می‌شود. هر سیکل دستورالعمل نیز خود به مجموعه‌ای ازسیکل‌های جزیی یا فازها تقسیم می‌شود. در کامپیوتر پایه هر سیکل دستور متشکل از فازهای زیر است .

۱- برداشت یک دستور از حافظه

۲- دیکد کردن دستور

۳- خواندن آدرس مؤثر

۴- اجرای دستورالعمل

پس از پایان مرحله ۴، کنترل به مرحله ۱ بازگشته و برداشت[2]، دیکد و اجرای[3] دستور بعدی را انجام می‌دهد. این فرآیند بطور نامحدود ادامه می‌یابد مگر اینکه کامپیوتر با دستور HALT مواجه شود.

برداشت و دیکد

در آغاز شمارنده برنامه، PC، با آدرس اولین دستورالعمل برنامه بار می‌شود. شمارنده SC پاک شده و در نتیجه سیگنال T_0 را تولید می‌نماید. پس از هر پالس ساعت، SC یک واحد افزایش می‌یابد تا رشته سیگنال‌های T_0، T_1، T_2 و غیره متوالیاً تولید گردند. ریزعمل‌های مربوط به فاز برداشت و دیکد را می‌توان توسط عبارات انتقال ثبات زیر مشخص نمود.

$T_0 : AR \leftarrow PC$

1- Wait 2- Fetch 3- Execute

T_1 : IR ← M [AR] , PC ← PC + 1

T_2 : D_0,... D_7 ← دیکد ، AR ← IR (0-11),I ← IR(15)

از آنجا که فقط AR به ورودی‌های آدرس حافظه متصل می‌باشد، لازم است که در حین سیگنال زمانبندی T_0، آدرس از PC به AR منتقل شود. سپس با خواندن دستورالعمل از حافظه، در سیگنال زمانبندی T_1، مقدار دستور در ثبات دستورالعمل قرار می‌گیرد. همزمان باعمل اخیر، PC یک واحد افزایش می‌یابد تا حاوی دستورالعمل بعدی در برنامه باشد. در زمان T_2، کد عمل موجود در IR ، دیکد می‌شود و بیت غیرمستقیم به فلیپ فلاپ I منتقل می‌گردد و نهایتاً بخش آدرس دستورالعمل به AR انتقال می‌یابد. توجه داشته باشید که SC پس از هر پالس ساعت افزایش می‌یابد تا سیگنال‌های T_0، T_1، T_2 متوالیاً تولید گردند.

شکل ۸-۵ چگونگی پیاده سازی دو عبارت انتقال ثبات را در سیستم گذرگاه نشان می‌دهد. برای فراهم کردن مسیر داده‌ها در انتقال PC به AR، باید سیگنال زمانبندی T_0 را به مدار اعمال نمائیم تا اتصالات زیر فراهم شود.

۱- قرار دادن محتوای PC روی گذرگاه با ورودی‌های انتخاب کننده $S_2S_1S_0$=010

۲- انتقال محتوای گذرگاه به AR با توانا کردن ورودی LD مربوط به AR

پالس ساعت بعدی انتقال از PC به AR را موجب می‌شود زیرا T_0=1 است. برای پیاده کردن عبارت دوم

T_1: IR ← M[AR], PC ← PC + 1

لازم است با استفاده از سیگنال زمانبندی T_1 اتصالات زیر را در سیستم گذرگاه برقرار کنیم.

۱- ورودی خواندن حافظه فعال شود

۲- محتوای حافظه به کمک $S_2S_1S_0$=111 روی گذرگاه قرار گیرد.

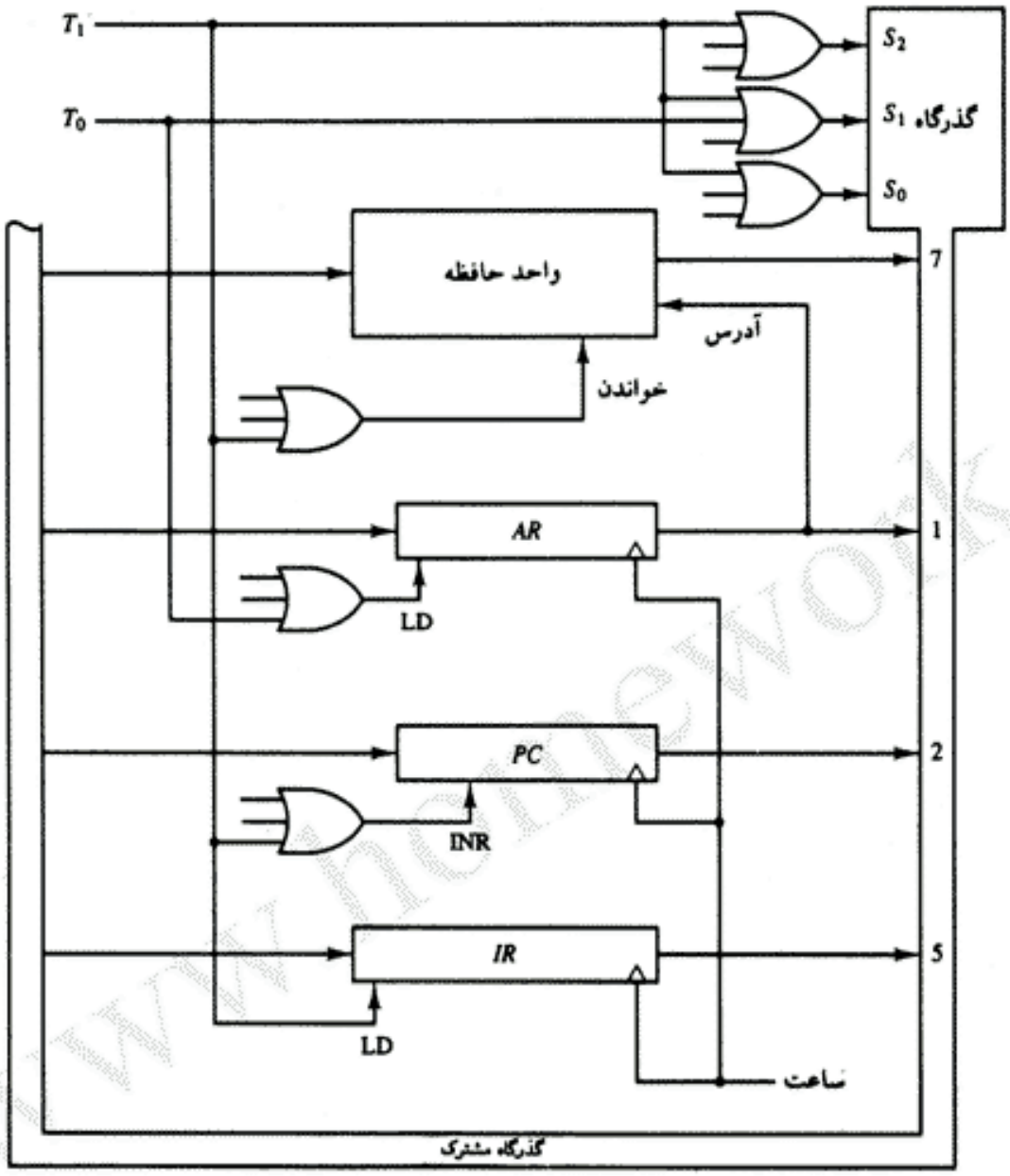
۳- محتوای گذرگاه با توانا شدن LD ثبات IR، به آن منتقل گردد.

۴- PC با توانا شدن ورودی INR، یک واحد افزایش می‌یابد.

شکل ۸-۵ بخشی از گذرگاه سیستم را همراه با اتصالات T_0 و T_1 به ورودی‌های کنترل ثبات‌ها، حافظه‌ها و ورودی‌های انتخاب کننده نشان می‌دهد. چون توابع کنترلی دیگری که اعمال مشابهی را انجام می‌دهند نیز وجود دارند لذا گیت‌های OR چند ورودی در دیاگرام آورده شده‌اند.

تعیین نوع دستورالعمل

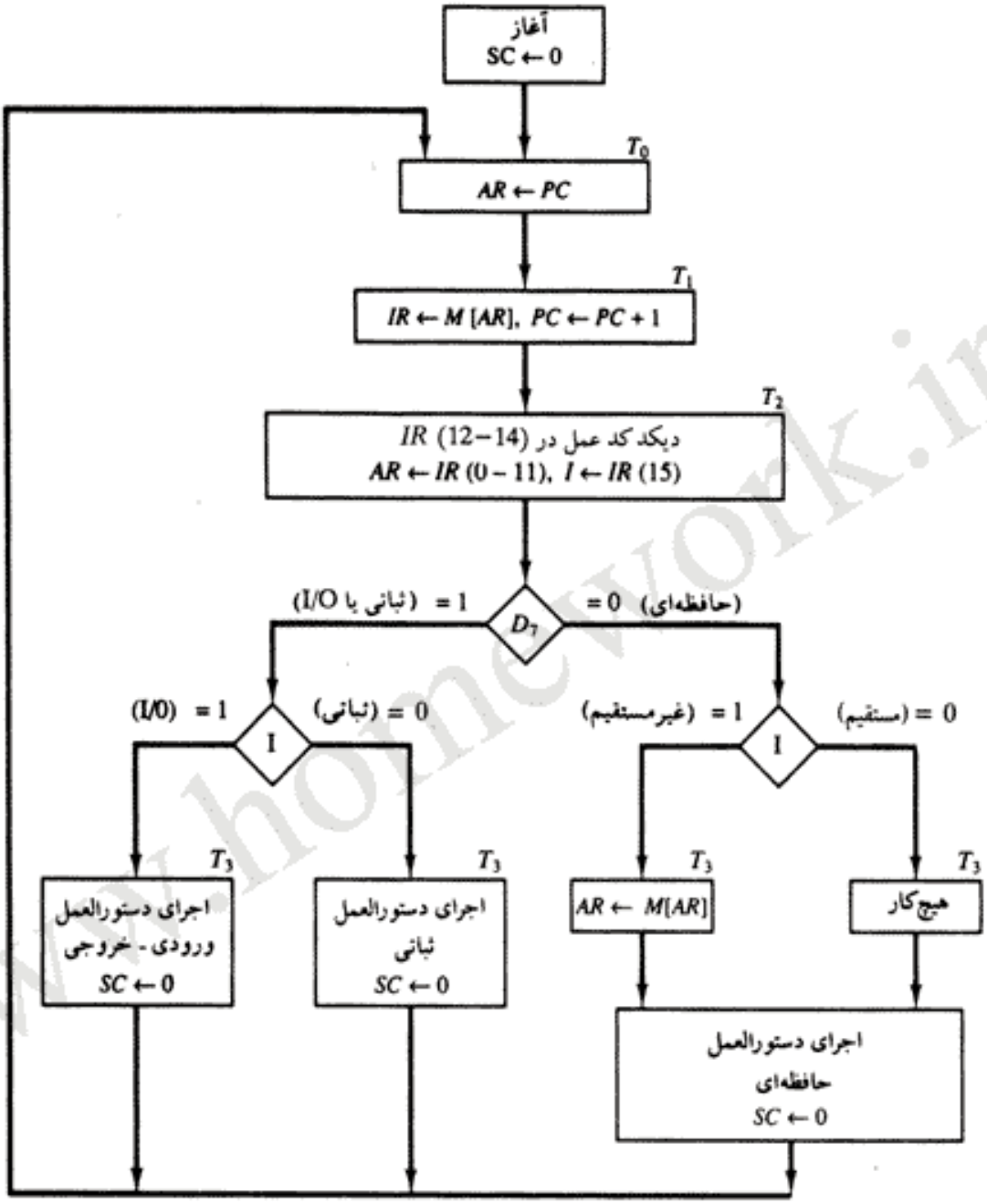
سیگنال زمانبندی بعدی که پس از دیکد شدن فعال است T_3 می‌باشد. در مدت زمان T_3، واحد



شکل ۸-۵ انتقال ثبات‌های مربوط به فاز برداشت

کنترل نوع دستوری را که درست چند لحظه قبل خوانده شده معین می‌کند. فلوچارت شکل ۹-۵ آرایش اولیه‌ای را برای سیکل دستورالعمل و چگونگی تعیین نوع دستورات را توسط کنترل پس از دیکد شدن نشان می‌دهد. سه نوع دستورالعمل موجود در کامپیوتر پایه در شکل ۵-۵ دیده می‌شود.

اگر کد عمل برابر با 111 باشد خروجی D_7 دیکدر 1 خواهد بود. با ملاحظه شکل ۵-۵ دیده می‌شود که اگر $D_7=1$ باشد دستور از نوع ارجاع به ثبات یا ورودی- خروجی است. برای $D_7=0$ کد عمل یکی از هفت نوع دیگر 000 تا 110 خواهد بود که همگی از نوع ارجاع به حافظه می‌باشند. سپس کنترل اولین



شکل ۹–۵ فلوچارت سیکل دستورالعمل (آرایش اولیه)

بیت دستورالعمل راکه در این لحظه در فلیپ فلاپ I است چک می‌کند. اگر $D_7=0$ و $I=1$ باشد دستور از نوع ارجاع به حافظه با آدرس غیرمستقیم است. در اینصورت لازم است که آدرس موثر از حافظه خوانده شود. ریزعمل مربوط به حالت آدرس غیرمستقیم توسط عبارت انتقال ثبات زیر صورت می‌گیرد.

$$AR \leftarrow M[AR]$$

در ابتدا، AR بخش آدرس دستورالعمل را در خود دارد. این آدرس در طول مدت خواندن حافظه بکار می‌رود. کلمه‌ای که توسط آدرس موجود در AR معین می‌شود از حافظه خوانده شده و روی گذرگاه مشترک قرار می‌گیرد. سپس ورودی LD مربوط به AR فعال می‌شود تا آدرس غیرمستقیم 12 بیتی با ارزش کمتر کلمه حافظه را دریافت نماید.

سه نوع دستورالعمل موجود به چهار مسیر فرعی جداگانه تقسیم می‌شوند. عمل انتخاب شده بوسیله سیگنال مرتبط با T_3 فعال می‌گردد که برای چهارمسیر توسط روابط زیر مشخص می‌شود.

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

$D'_7I'T_3$: هیچ کار

$D_7I'T_3$: اجرای دستور ارجاع به ثبات

D_7IT_3 : اجرای یک دستور ورودی - خروجی

وقتی که دستورالعمل ارجاع حافظه با $I=0$ پیش آید، لازم نیست کاری انجام شود زیرا در این حالت آدرس موثر از قبل در AR است. با این وجود شمارنده SC، وقتی که $D'_7T_3 = 1$ باشد، باید افزایش یابد تا اجرای دستورالعمل ارجاع به حافظه با متغیر زمانی T_4 ادامه یابد. یک دستورالعمل ارجاع به ثبات یا ورودی - خروجی با پالس ساعت مربوط به T_3 اجرا می‌شود. پس از اجرای دستور، SC پاک شده وکنترل با $T_0=1$ به فاز برداشت باز می‌گردد.

دقت کنید که شمارنده SC با هر لبه مثبت پالس افزایش یافته و یا پاک می‌شود. ما در این جا این قرارداد را می‌پذیریم که بازاء هر افزایش عبارت $SC \leftarrow SC + 1$ را ننویسیم ولی بطور ضمنی می‌دانیم که کنترل به مرحله بعدی منتقل می‌شود. چنانچه قرار باشد SC پاک شود عبارت $SC \leftarrow 0$ را خواهیم نوشت.

در این بخش انتقال‌های ثباتی لازم برای اجرای دستورات ارجاع به ثبات ارائه شده است. دستورالعمل‌های ارجاع به حافظه در بخش بعد شرح داده شده‌اند. دستورالعمل‌های ورودی - خروجی در بخش ۷-۵ توضیح داده شده‌اند.

دستورالعمل‌های ارجاع به ثبات

این نوع دستورات وقتی که $D_7=1$ و $I = 0$ است توسط کنترل قابل تشخیص‌اند. بیت‌های 0 تا 11 این دستورات برای تعیین یکی از 12 دستورالعمل بکار می‌روند. این 12 بیت در IR (0-11) قرار دارند و در طول T_2 نیز به AR منتقل شده‌اند. توابع کنترل و ریزعمل‌ها برای دستورات ارجاع به ثبات در جدول ۳-۵ لیست شده‌اند. این دستورالعمل‌ها با انتقال پالس ساعت در T_3 اجرا می‌شوند. هر یک از توابع کنترل به یک $D_7I'T_3$ نیاز دارند که ما آن را به r نشان می‌دهیم. توابع کنترل توسط یکی از بیت‌های IR (0-11) از هم متمایز می‌گردند. با تخصیص نماد یا سمبل B_i به بیت i از IR، همه توابع کنترلی را می‌توان با rB_i نشان داد. مثلاً دستورالعمل CLA دارای کد مبنای شانزده 7800 است (جدول ۲-۵

جدول ۳-۵ اجرای دستورالعمل‌های ثباتی

$D_7I'T_3 = r$ مشترک در همه دستورالعملهای ثباتی
$IR(i) = B_i$ بیتی در IR (0-11) که عمل را مشخص می‌کند

	r:	$SC \leftarrow 0$	پاک کردن SC
CLA	rB_{11}:	$AC \leftarrow 0$	پاک کردن AC
CLE	rB_{10}:	$E \leftarrow 0$	پاک کردن E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	متمم کردن AC
CME	rB_8:	$E \leftarrow \overline{E}$	متمم کردن E
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	شیفت چرخشی به راست
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$	شیفت چرخشی به چپ
INC	rB_5:	$AC \leftarrow AC + 1$	افزایش AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	گذر در صورت مثبت بودن
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	گذر در صورت منفی بودن
SZA	rB_2:	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$	گذر در صورت صفر بودن AC
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	گذر در صورت صفر بودن E
HLT	rB_0:	$S \leftarrow 0$ S یک فلیپ‌فلاپ آغاز و پایان است	توقف کامپیوتر

ملاحظه شود)، که معادل عدد دودویی 0000 0000 1000 0111 می‌باشد. اولین بیت 0 است که معادل با I' می‌باشد. سه بیت بعدی کد عمل را تشکیل می‌دهند و از خروجی D_7 دیکدر قابل تشخیص‌اند. بیت 11 در IR برابر 1 است که از B_{11} قابل تشخیص است. بنابراین تابع کنترلی که ریز عملیات را برای این دستور آغاز می‌کند $D_7I'T_3B_{11}$=rB11 خواهد بود. اجرای دستورالعمل ارجاع به ثبات در زمان T_3 پایان می‌پذیرد. شمارنده SC پاک شده و کنترل برای برداشت دستوربعدی با T_0 به عقب برمی‌گردد.

هفت دستور اول ثباتی، اعمال پاک کردن، مکمل سازی، چرخش و افزایش روی ثبات‌های AC و E را انجام می‌دهند. چهار دستوربعدی موجب گذر[1] (رد شدن) از دستور بعدی در صورت برقرار بودن یک شرط معین می‌شود. این گذر با افزایش PC صورت می‌گیرد (علاوه برافزایش آن در فاز برداشت دستورالعمل در فاز T_1). عبارت‌های کنترل شرطی را باید بخشی از شرایط کنترلی تلقی کرد. هرگاه بیت علامت AC، AC(15)=0 شود محتوای آن یک عدد مثبت است و اگر AC=1 باشد منفی است. اگر تمام فلیپ فلاپ های ثبات صفر باشند محتوای AC صفر است. دستور HLT فلیپ فلاپ استارت - استاپ[2] را پاک می‌کند که موجب توقف شمارنده SC می‌گردد. برای بکار انداختن مجدد کامپیوتر فلیپ - فلاپ استارت - استاپ باید بصورت دستی 1 شود.

۵-۶ دستورالعمل‌های ارجاع به حافظه

در تعیین ریزعملهای موردنیاز برای اجرای هر دستورالعمل، لازم است تابعی که باید آنرا اجرا کند دقیقاً تعریف شود. با نگرشی به جدول ۲-۵، که در آن دستورالعمل‌ها تعریف شدند، می‌بینیم که برخی

1- skip جدول ۳-۵ اجرای دستورات ارجاع به ثبات 2- Start – stop

از دستورات شرح مبهمی دارند. دلیل این است که این دستورات با جملات طولانی توصیف شده‌اند و فضای کافی برای چنین توضیح طولانی در جدول وجود ندارد. حال ما نشان خواهیم داد که دستور ارجاع به حافظه کاملاً با توجه به علائم و زبان انتقال ثبات قابل تعریف است.

جدول ۴-۵ هفت دستور ارجاع به حافظه را نشان‌می‌دهد. خروجی دیکد شده D_i برای $i=0,1,2,3,4,5,6$ متعلق به هر دستور نیز در جدول آورده شده است. آدرس موثر در ثبات AR و در مقطع T_2 بهنگام $I=0$ و یا در T_3 با $I=1$ جای می‌گیرد. اجرای دستورهای حافظه‌ای با سیگنال زمانبندی T_4 شروع می‌شود. توصیف سمبلیک هر دستور هم در جدول برحسب علائم انتقال ثبات مشخص شده است. اجرای واقعی هر دستور در سیستم نیاز به یک رشته ریزعمل دارد. دلیل این است که داده‌های ذخیره شده در حافظه نمی‌توانند مستقیماً پردازش شوند. این داده‌ها باید ابتدا از یک حافظه خوانده شده و در ثباتی قرار گیرند که مدارهای منطقی بتوانند روی آنها عمل کنند. حال ما عملکرد هر دستور را توضیح داده و توابع کنترل و ریزعمل‌های لازم رابرای اجرای آنها لیست می‌کنیم. فلوچارتی که تمام این ریز اعمال را در اختیار می‌گذارد در آخر این بخش آمده است.

AND با AC

این دستور عمل منطقی AND بر روی جفت بیت‌های AC و حافظه‌ای که توسط آدرس مؤثرش معین می‌شود را انجام می‌دهد. نتیجه عمل فوق در انتها به AC منتقل می‌شود. ریزعمل‌هایی که این دستورالعمل را اجرا می‌کنند عبارتند از

$$D_0\ T_4: DR \leftarrow M[AR]$$
$$D_0\ T_5: AC \leftarrow AC \wedge DR\ ,\ SC \leftarrow 0$$

تابع کنترل این دستور خروجی دیکدر D_0 را بکار می‌برد زیرا این خروجی وقتی که عمل AND با

جدول ۴-۵ دستورالعمل‌های حافظه‌ای

سمبل	دیکدر عمل	توضیح سمبلیک
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR],\quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC,\quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

کد 000 اجرا می‌شود فعال می‌گردد. دو سیگنال زمانبندی برای اجرای دستور فوق لازم است. لبه پالس‌مربوط به سیگنال T_4، عملوند را از حافظه به DR انتقال می‌دهد. لبه پالس بعدی یعنی T_5، نتیجه عمل AND بر روی AC و DR را به AC منتقل و SC را هم صفر می‌کند که این خود سبب واگذاری کنترل به سیگنال T_0 می‌گردد تا سیکل دستورالعمل جدیدی آغاز شود.

ADD با AC

این دستور محتوای حافظه مشخص شده توسط آدرس موثر را با مقدار AC جمع می‌کند. حاصل جمع به AC و رقم نقلی خروجی C_{out} به فلیپ‌فلاپ E (بیت گسترش انباره)[1] منتقل می‌شود. ریز اعمال لازم برای این دستور عبارتست از

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

$$D_1\ T_5: AC \leftarrow AC + DR,\quad E \leftarrow C_{out},\quad SC \leftarrow 0$$

مشابه با دستور قبل دو سینگال زمانی T_4 و T_5 بکار رفته‌اند ولی خروجی D_1 دیکدر بجای D_0که در عمل AND بکار رفت فعال است. پس از برداشت و دیکد دستور تنها یک خروجی دیکدر فعال خواهد بود و این خروجی رشته ریزعملهایی که واحد کنترل هنگام اجرای یک دستورالعمل حافظه‌ای دنبال می‌کند را تعیین می‌نماید.

LDA : بارکردن AC

این دستور محتوای حافظه‌ای که توسط آدرس موثرش مشخص شده را به AC منتقل می‌کند.

$$D_2\ T_4: DR \leftarrow M[AR]$$

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

با بازنگری مجدد شکل ۴-۵ ملاحظه می‌شود که مسیر مستقیمی از گذرگاه به AC وجود ندارد. مدار جمع کننده و منطقی، اطلاعات را از DR دریافت می‌کند که قابل انتقال به AC است. بنابراین لازم است تا ابتدا کلمه حافظه به DR فراخوانده شده و سپس از DR به AC منتقل شود. دلیل عدم ارتباط گذرگاه به AC، تاخیری است که در مدارهای جمع کننده و منطقی وجود دارد. در اینجا فرض بر این است که زمان لازم برای خواندن از حافظه و انتقال به گذرگاه و مدارهای جمع کننده و منطقی بیش از یک پالس ساعت است. با عدم اتصال گذرگاه به ورودی‌های AC می‌توانیم زمان یک سیکل ساعت را برای هر ریز عمل حفظ کنیم.

1- Extended AC

STA: ذخیره کردن AC در حافظه

این دستور محتوای AC را درحافظه‌ای که با آدرس موثرش مشخص شده ذخیره می‌نماید. چون خروجی AC به گذرگاه وصل است و ورودی حافظه هم به گذرگاه متصل شده، می‌توانیم این دستور را با یک ریز عمل انجام دهیم.

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

BUN: انشعاب بدون شرط

این دستور برنامه را به دستورالعملی که توسط آدرس موثرش مشخص شده منتقل می‌نماید. بخاطر داشته باشید که PC آدرس دستوری که در سیکل بعدی خوانده می‌شود را در خود نگمیدارد. PC در زمان T_1 افزایش می‌یابد تا برای دستور بعدی در برنامه آماده باشد. دستورالعمل BUN به برنامه نویس امکان می‌دهد تا دستوری را در خارج از رشته متوالی برنامه مشخص کند و در این حالت گوئیم برنامه بدون شرط انشعاب یافته است. این دستورالعمل با یک ریز عمل اجرا می‌شود.

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

آدرس موثر از AR و از طریق گذرگاه مشترک به PC منتقل می‌شود. با پاک شدن SC، کنترل به T_0 منتقل می‌گردد. سپس دستور بعدی توسط مقدار جدید در PC از حافظه برداشت و اجرا می‌گردد.

BSA: انشعاب با ذخیره آدرس برگشت

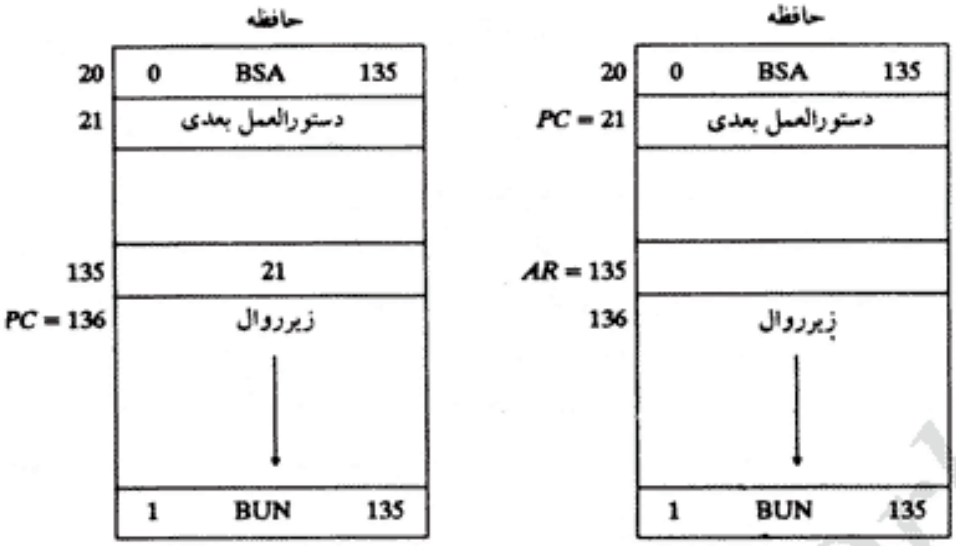
این دستور برای انشعاب به بخشی از برنامه که زیرروال[1] یا رویه خوانده می‌شود مفید است. هروقت این دستور اجرا شود BSA آدرس دستور بعدی راکه در PC موجود است در حافظه‌ای که توسط آدرس موثر معین می‌شود ذخیره می‌نماید. سپس آدرس موثر بعلاوه 1 به PC منتقل می‌شود تا بعنوان آدرس اولین دستور در زیرروال مورد استفاده قرارگیرد. این عمل در جدول ۴-۵ همراه با عبارت انتقال ثبات در زیر آمده است.

$$M[AR] \leftarrow PC\ ,\ PC \leftarrow AR+1$$

یک مثال عددی که نشان دهنده چگونگی بکار رفتن این دستور است در شکل ۱۰-۵ دیده می‌شود فرض بر این است که دستور BSA در حافظه 20 قرار گرفته باشد. بیت I برابر 0 و بخش آدرس دستور 135 تصور می‌شود. پس از فازهای برداشت و دیکد، PC حاوی 21 خواهد بود که آدرس دستور بعدی در برنامه است (این همان آدرس بازگشت است). AR حاوی آدرس موثر 135 است. این مطلب در قسمت (الف) شکل ملاحظه می‌گردد. دستورالعمل BSA عمل عددی زیر را اجرا می‌نماید.

$$M[135] \leftarrow 21\ ,\ PC \leftarrow 135+1 = 136$$

1- Subroutine



شکل ۱۰-۵

نتیجه این عمل در بخش (ب) از شکل مزبور دیده می‌شود. آدرس بازگشت 21 در حافظه 135 ذخیره شده و کنترل از 136 زیرروال بکار خود ادامه می‌دهد. بازگشت به برنامه اصلی (در آدرس 21) توسط دستور BUN که در انتهای زیرروال قرار دارد انجام می‌شود. وقتی این دستور اجرا شود کنترل آدرس موثر را که در مکان 135 قرار دارد و همان آدرس 21 است می‌خواند. پس از اجرای BUN، آدرس 21 به PC منتقل می‌شود و در نتیجه دستور واقع در آدرس بازگشت اجرا خواهد شد.

دستورالعمل BSA عملی را انجام می‌دهد که معمولاً به آن فراخونی[1] زیرروال می‌گوئید. دستور BUN در انتهای زیر روال عملی را که به آن بازگشت از زیرروال گویند انجام می‌دهد. در بسیاری از کامپیوترهای تجاری آدرس برگشت از زیرروال در یکی از ثباتهای پردازشگر یا در بخشی از حافظه بنام پشته[2] ذخیره می‌شود. این موضوع در بخش ۷-۸ مشروحاً توضیح داده خواهد شد.

امکان اجرای دستور BSA در یک پالس ساعت، در صورتی که از سیستم گذرگاه کامپیوتر پایه استفاده کنیم، میسر نیست. برای استفاده صحیح از حافظه و گذرگاه، دستور BSA باید با دو ریزعمل زیر اجرا شود

$$D_5\ T_4:\ M[AR] \leftarrow PC\ ,\ AR \leftarrow AR+1$$
$$D_5\ T_5:\ PC \leftarrow AR\ \ ,\ SC \leftarrow 0$$

سیگنال زمانبندی T_4 یک عمل نوشتن در حافظه را آغاز می‌کند و طی آن محتوای PC را روی گذرگاه قرار می‌دهد، و ورودی INR مربوط به AR را فعال می‌کند. نوشتن در حافظه و افزایش AR تالبه پالس

1- Call
2- Stack

بعدی تکمیل می‌گردد. در T_5 گذرگاه برای انتقال AR به PC مورد استفاده قرار می‌گیرد.

ISZ: افزایش و گذر اگر نتیجه صفر

این دستور کلمه‌ای که توسط آدرس موثر مشخص می‌شود را یک واحد افزایش می‌دهد و اگر مقدار افزایش یافته 0 شود، PC را یک واحد افزایش می‌دهد. برنامه نویس معمولاً یک عدد منفی را (بفرم متمم 2) در حافظه قرار می‌دهد. این عدد منفی مرتباً افزایش می‌یابد و نهایتاً به صفر می‌رسد. در این لحظه PC یک واحد اضافه می‌شود تا از دستور بعدی در برنامه گذر[1] کند.

چون امکان افزایش کلمه در داخل حافظه ممکن نیست، لازم است کلمه به DR منتقل و در DR یک واحد افزایش و سپس به حافظه بازگردانده شود. این کار با ریزعمل‌های زیر انجام می‌شود:

$D_6\ T_4$: DR ← M[AR]

$D_6\ T_5$: DR ← DR+1

$D_6\ T_6$: M[AR] ← DR if (DR=0) then (PC←PC+1),SC←0

فلوچارت کنترل

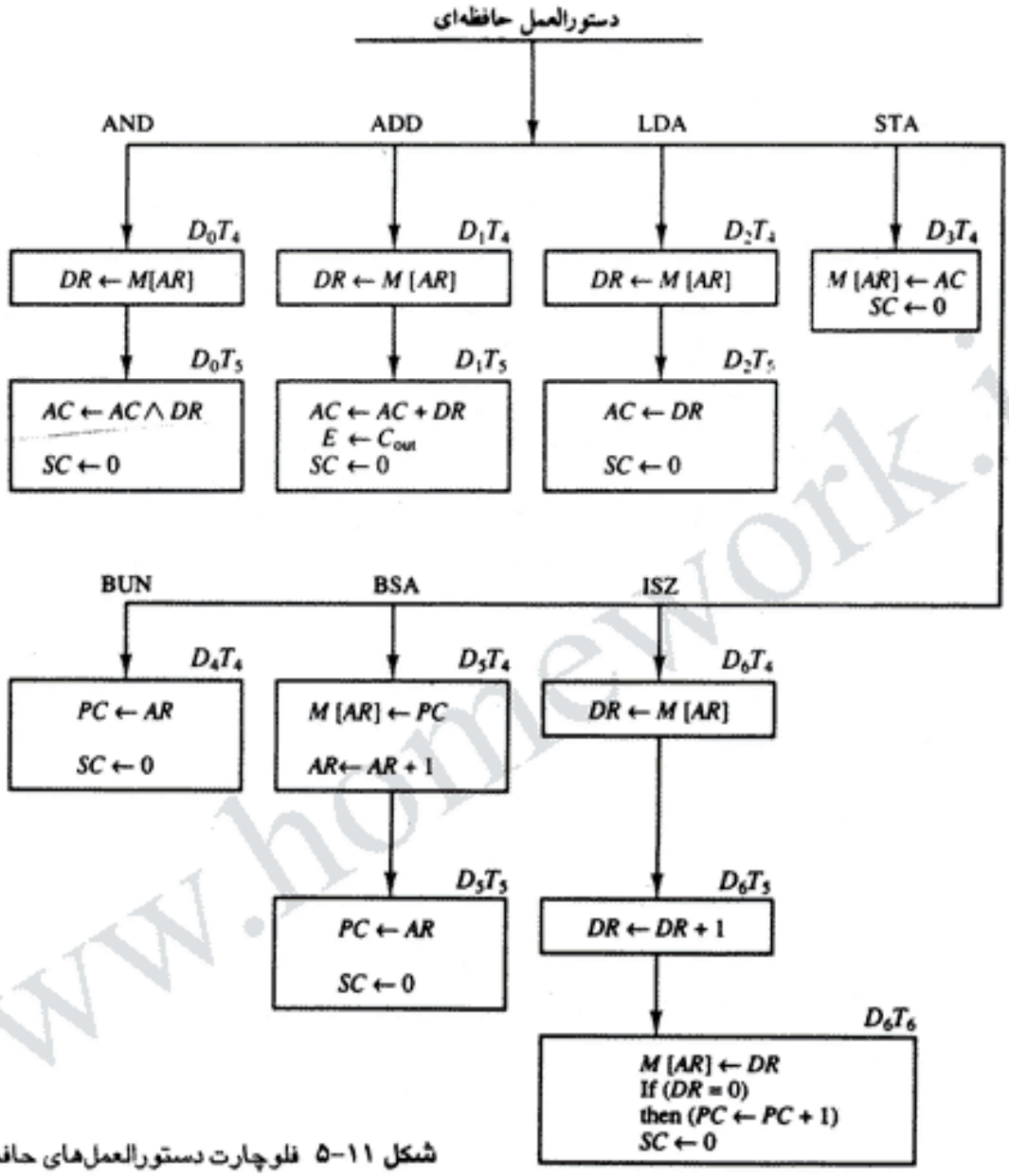
فلوچارت نمایش اجرای ریزعمل‌های مربوط به هفت دستورالعمل ارجاع به حافظه در شکل ۱۱-۵ نشان داده شده است. توابع کنترل در بالای هرچهارگوشه آمده است. ریزعمل‌های اجرا شده در T_4، T_5 و T_6 به مقدار کد عمل بستگی دارد. این مطلب در فلوچارت توسط شش مسیر مختلف مشخص شده که هر یک از آنها پس از دیکد شدن دستورالعمل توسط کنترل انتخاب می‌شوند. شمارنده برنامه SC در هر حالت پس از آخرین سیگنال زمانبندی ، 0 می‌شود. این امر باعث انتقال کنترل به سیگنال زمانبندی T_0 برای آغاز سیکل دستورالعمل بعدی می‌گردد.

توجه کنید که برای اجرای طولانی ترین دستورالعمل (ISZ) به هفت سیگنال زمانبندی نیاز داریم. لذا کامپیوتر را می‌توان با یک شمارنده سه بیتی SC طراحی کرد. دلیل بکارگیری چهاربیت در SC این است که سیگنال‌های زمانی لازم برای سایر دستورات در بخش تمرینات میسر باشد.

۵-۷ ورودی - خروجی و وقفه[2]

یک کامپیوتر مادامی که با محیط خارج خود ارتباط برقرار نکرده است نمی‌تواند مفید واقع شود. دستورات و داده‌های ذخیره شده در حافظه باید از برخی وسایل خارجی متصل به ورودی گرفته شوند. نتایج محاسبات هم باید از طریق نوعی وسیله خروجی به کاربر انتقال یابد. کامپیوترهای تجاری مجهز به انواع وسایل ورودی و خروجی هستند. برای نمایشی از اساسی ترین ملزومات ارتباطی ورودی و

1- Skip 2- Interrupt



شکل ۱۱-۵ فلوچارت دستورالعمل‌های حافظه‌ای

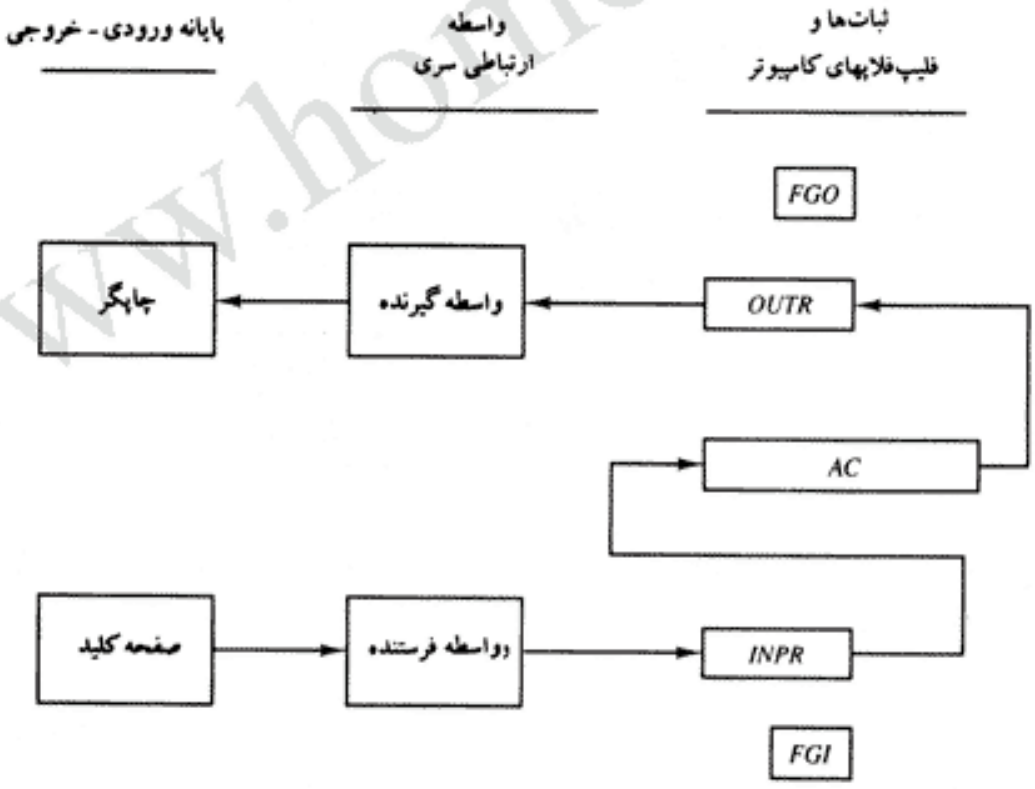
خروجی، ما واحد پایانه‌ای را با صفحه کلید و چاپگر نشان خواهیم داد. سازمان ورودی - خروجی در فصل ۱۱ بحث خواهد شد.

آرایش ورودی - خروجی

پایانه، اطلاعات سری را ارسال و دریافت می‌کند. هر یک از کمیت‌های اطلاعاتی حاوی یک کد الفباء عددی هشت بیتی است. اطلاعات سری وارده از صفحه کلید بداخل INPR شیفت داده می‌شود. اطلاعات سری چاپگر در OUTR ذخیره می‌گردد. این دو ثبات بایک مدار واسط وسیله جانبی بطور سری وبا

AC ارتباط موازی دارند. آرایش ورودی - خروجی در شکل ۱۲-۵ دیده می‌شود. مدار واسط وسیله ارسال‌کننده[1] (فرستنده) اطلاعات سری را از صفحه کلید دریافت و به INPR می‌فرستد. مدار واسط دریافت کننده،[2] اطلاعات را از OUTR دریافت وبه چاپگر ارسال می‌دارد. عملکرد مدار واسط در بخش ۳-۱۱ تشریح شده است.

ثبات INPR از هشت بیت تشکیل و اطلاعات الفبا عددی را در خود نگه می‌دارد. پرچم یک بیتی FGI یک فلیپ فلاپ کنترل است و این پرچم هر وقت اطلاعات جدیدی دروسیله ورودی موجود باشد برابر 1 می‌شود و وقتی کامپیوتر این اطلاعات را پذیرفت برابر 0 می‌گردد. این پرچم برای همزمان کردن سرعت جریان اطلاعات بین وسیله ورودی و کامپیوتر لازم است. فرآیند انتقال اطلاعات بقرار زیر است. در ابتدا به پرچم ورودی FGI مقدار 0 داده می‌شود. وقتی کلیدی روی صفحه کلید فشار داده می‌شود، یک کد الفبا عددی هشت بیتی بطور سری به INPR شیفت پیدا کرده و پرچم FGI برابر 1 می‌گردد. مادامی که این پرچم 1 است محتوای INPR قابل تغییر بوسیله کلید دیگر نیست. کامپیوتر بیت پرچم را چک می‌نماید، اگر پرچم 1 باشد، اطلاعات از INPR بطور موازی به AC منتقل و FGI پاک



شکل ۱۲-۵ آرایش ورودی- خروجی

1- Transmitter Interface 2- Reciever Interface

می‌شود. به محض پاک شدن پرچم، اطلاعات جدید حاصل از کلید جدید می‌تواند وارد INPR گردد. ثبات خروجی OUTR بطور مشابه کار می‌کند ولی جهت انتقال اطلاعات معکوس است. در آغاز پرچم خروجی FGO برابر 1 است. کامپیوتر این پرچم را چک می‌کند؛ اگر برابر 1 باشد، اطلاعات از AC بطور موازی به OUTR منتقل و FGO پاک می‌شود. وسیله جانبی خروجی اطلاعات کد شده را پذیرفته و کاراکتر مربوطه را چاپ و درپایان FGO را 1 می‌کند. کامپیوتر مادامی که FGO برابر O است اطلاعاتی را در OUTR ذخیره نمی‌کند زیرا صفربودن FGO نشانه این است که خروجی در حال چاپ کاراکتر است.

دستورات ورودی - خروجی

دستورات ورودی و خروجی برای تبادل اطلاعات از ثبات AC، و چک بیت‌های پرچم برای کنترل وقفه مورد نیازند. دستورات ورودی و خروجی دارای کد عملیاتی 1111 بوده و هنگامی که $D_7=1$ و $I=1$ باشد توسط کنترل قابل تشخیص اند. بیت‌های دیگر معین‌کننده دستورالعمل خاص موردنظر هستند. توابع کنترل و ریزعمل‌های ورودی - خروجی در جدول ۵-۵ لیست شده‌اند. اجرای این دستورات با لبه پالس ساعت T_3 آغاز می‌شود. همه توابع کنترل نیاز به رابطه بولی D_7IT_3 دارند که برای راحتی، آنرا با P نشان می‌دهیم. هر تابع کنترل خاص مورد نظر توسط یکی از بیت‌های IR(6-11) از دیگری تفکیک می‌شود. با فرض نمایش بیت نام IR با B_i تمام توابع کنترل را می‌توان با PB_i نشان داد که در آن i از 6 تا 11 تغییر می‌کند. شمارنده SC وقتی که $P=D_7IT_3=1$ باشد پاک می‌شود.

دستور INP اطلاعات ورودی را از INPR به هشت بیت پائین رتبه AC انتقال داده و پرچم ورودی را پاک می‌کند. دستور OUT هشت بیت پائین رتبه AC را به ثبات OUTR منتقل و پرچم خروجی را 0 می‌نماید. دو دستور بعدی در جدول (۵-۵) وضعیت پرچم‌ها را تست نموده و در صورت 1 بودن موجب گذر از دستور بعدی می‌شوند. دستورالعملی که از آن گذر می‌شود معمولاً یک دستور انشعاب برای بازگشت و کنترل مجدّد پرچم است. اگر پرچم 0 باشد از دستور انشعاب گذر نمی‌شود. ولی اگر پرچم 1 باشد دستور انشعاب گذر شده و دستوری از نوع ورودی یا خروجی اجرا گردد. (مثال‌هایی از برنامه‌های ورودی و خروجی در بخش ۸-۶ آمده است). دو دستور آخر یک فلیپ فلاپ IEN وقفه را

جدول ۵-۵ دستورالعمل‌های ورودی- خروجی

$D_7IT_3 = p$ (مشترک در همه دستورالعمل‌های ورودی - خروجی)
$IR(i) = B_i$ [بیتی در IR(0-11) که دستورالعمل را مشخص می‌کند]

	p:	$SC \leftarrow 0$	پاک کردن SC
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$, $FGI \leftarrow 0$	دریافت کاراکتر ورودی
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7)$, $FGO \leftarrow 0$	ارسال کاراکتر خروجی
SKI	pB_9:	If ($FGI = 1$) then ($PC \leftarrow PC + 1$)	گذر بر اساس پرچم ورودی
SKO	pB_8:	If ($FGO = 1$) then ($PC \leftarrow PC + 1$)	گذر بر اساس پرچم خروجی
ION	pB_7:	$IEN \leftarrow 1$	روشن کردن فعال‌ساز وقفه
IOF	pB_6:	$IEN \leftarrow 0$	خاموش کردن فعال‌ساز وقفه

پاک و یا برقرار[1] می‌نمایند (می‌نشاند). هدف از IEN به همراه بحث مربوط به وقفه بیان شده است.

وقفه برنامه

فرآیند ارتباطی بحث فوق بنام انتقال تحت کنترل برنامه خوانده می‌شود. کامپیوتر بطور مداوم بیت پرچم را چک کرده ووقتی آن را 1 بیابد انتقال را آغاز می‌کند. بدلیل تفاوت سرعت جریان اطلاعات بین کامپیوتر و وسیله ورودی - خروجی کارآیی این نوع انتقال پائین است. برای درک این مطلب، کامپیوتری را در نظر بگیرید که می‌تواند سیکل دستور را در $1\mu s$ انجام دهد. فرض کنید که وسیله ورودی - خروجی می‌تواند اطلاعات را حداکثر با میزان 10 کاراکتر در ثانیه منتقل نماید. این سرعت معادل با 100,000 میکروثانیه برای هر کاراکتر می‌باشد. وقتی که کامپیوتر بیت پرچم را چک می‌کند و تصمیم می‌گیرد اطلاعات را انتقال ندهد دو دستورالعمل اجرا می‌شود. این بدان معنی است که در سرعت ماکزیمم، کامپیوتر بین هر دو انتقال 50000 بار پرچم را چک خواهد کرد. کامپیوتر مادامی که در حال چک کردن پرچم‌هاست بعوض انجام کار مفیدی وقت را تلف می‌کند.

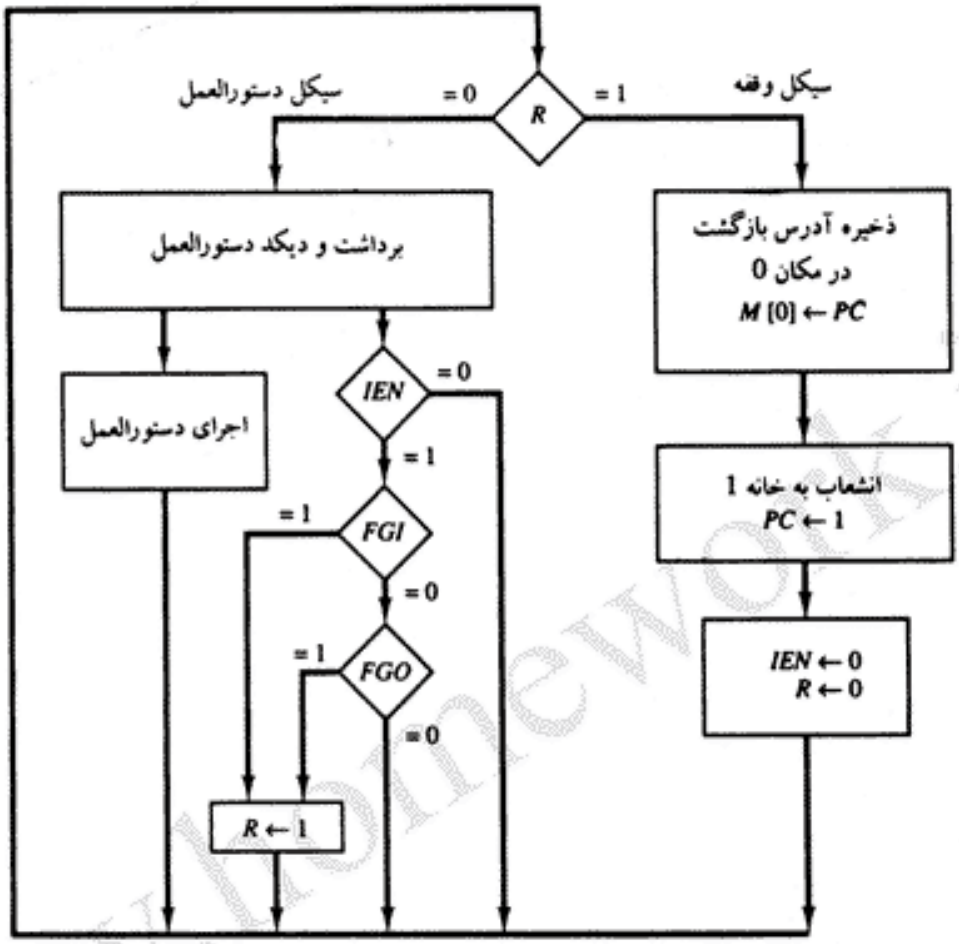
روشی دیگر این است که وسیله خارجی بهنگام آماده بودن انتقال، کامپیوتر را آگاه سازد. در این هنگام کامپیوتر مشغول به کارهای دیگری است. این نوع انتقال از امکان وقفه استفاده می‌کند. کامپیوتر در حالت اجرای یک برنامه، دیگر نیاز به چک کردن پرچم‌ها ندارد. با این وجود هر وقت پرچمی 1 شد کامپیوتر از پیشروی در برنامه جاری بازداشته شده و از 1 شدن پرچم آگاه می‌شود. سپس کامپیوتر انتقال اطلاعات را از ورودی یا خروجی انجام می‌دهد. در پایان به برنامه جاری بازگشته و به آنچه قبل از وقفه انجام می‌داد ادامه می‌دهد.

فلیپ‌فلاپ تواناساز (فعال‌ساز) وقفه IEN را با دو دستور می‌توان 0 یا 1کرد. وقتی IEN پاک‌شود (با دستور IOF) پرچم‌ها قادر به ایجاد وقفه در کامپیوتر نیستند. وقتی IEN برقرار شود (۱ شود) (با دستور ION) کامپیوتر قابل توقف است. این دو دستور برنامه‌نویس را قادرمی‌سازند تا از امکانات وقفه استفاده نماید.

نحوه واکنش کامپیوتر به وقفه با توجه به شکل ۱۳-۵ قابل تشریح است. یک فلیپ فلاپ R در کامپیوتر منظور شده است. وقتی R=0 باشد کامپیوتر واردسیکل دستورالعمل می‌گردد. به موازات سیکل اجرای دستورالعمل، IEN توسط کنترل چک می‌شود. 0 بودن این فلیپ فلاپ بمعنی این است که برنامه نویس مایل به استفاده از وقفه نیست. لذاکنترل با دستور بعدی بکار ادامه می‌دهد. اگر IEN=1 باشد کنترل بیت‌های پرچم را چک می‌نماید. اگر هر دو پرچم 0 باشند نه ورودی و نه خروجی آماده انتقال اطلاعات نیستند. در این حالت کنترل با دستور بعدی به کارادامه می‌دهد. اگر ضمن IEN=1 هر یک از دو پرچم ورودی یا خروجی 1 شوند R=1 می‌گردد. در انتهای فاز اجرای دستور، کنترل R را چک می‌نماید و چنانچه مقدار آن برابر 1 باشد کنترل در عوض اجرای دستور بعدی وارد سیکل وقفه می‌گردد. سیکل وقفه، پیاده سازی سخت افزاری یک انشعاب و ذخیره آدرس برگشت است. آدرس موجود در

---

1 – Set

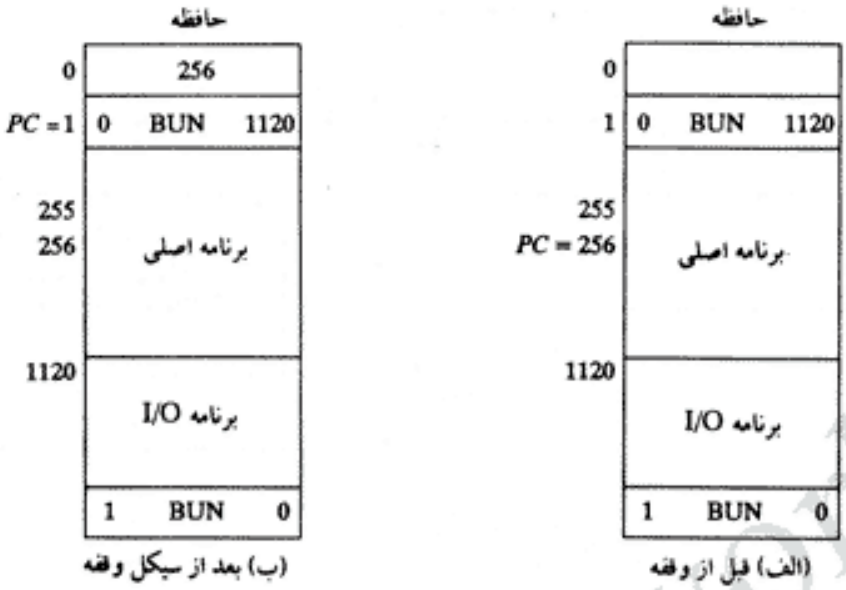


شکل ۱۳-۵ فلوچارت سیکل وقفه

PC در یک مکان خاص ذخیره می‌شود تا بهنگام بازگشت برنامه به دستوری که آنرا ترک کرده است از آن استفاده نماید. این مکان ممکن است یک ثبات پردازشگر، یک حافظه پشته یا یک مکان خاصی از حافظه باشد. در اینجا ما حافظه با آدرس 0 را برای نگهداری آدرس برگشت انتخاب می‌کنیم. کنترل سپس آدرس 1 را در PC وارد کرده و IEN و R را پاک می‌کند در نتیجه هیچ وقفه‌ای، مادامی که تقاضای وقفه قبلی سرویس نشده باشد اتفاق نمی‌افتد.

مثالی که نشان دهنده رخدادها در سیکل وقفه است در شکل ۱۴-۵ دیده می‌شود. فرض کنید که وقفه‌ای رخ داده و R=1 است و بموازات آن هم کنترل مشغول اجرای دستوری در آدرس 255 می‌باشد. در این زمان، آدرس برگشت در PC برابر با 256 است. برنامه نویس قبلاً برنامه‌ای را برای سرویس دادن به ورودی و خروجی در حافظه و در آدرس 1120 قرار داده و یک BUN 1120 را در آدرس 1 قرار داده است. این مطلب در شکل ۱۴-۵ (الف) نشان داده شده است.

وقتی کنترل به T_0 رسیده و R را برابر 1 بیابد، وارد سیکل وقفه می‌گردد. محتویات PC(256) در



شکل ۱۴-۵ مثالی از سیکل وقفه

مکان 0 حافظه ذخیره می‌شود. PC مساوی 1 می‌شود و R با 0 پاک می‌شود. در آغاز سیکل دستورالعمل بعدی، دستوری که ازحافظه خوانده شده در آدرس 1 قرار دارد زیرا این عدد همان محتوای PC است. دستور انشعاب واقع در آدرس 1 سبب می‌شود تا اجرا به برنامه سرویس ورودی -خروجی واقع در آدرس 1120 منتقل گردد. این برنامه پرچم ها را چک و تعیین می‌کند که کدام پرچم 1 شده است و سپس اطلاعات ورودی یا خروجی را منتقل می‌نماید. پس از انجام این عمل، دستورالعمل ION برای 1کردن IEN اجرا می‌شود (تا وقفه را ادامه دهد) و برنامه به مکانی که وقفه در آن رخ داده بود باز می‌گردد.

دستورالعملی که کامپیوتر را به مکان اولیه‌اش در برنامه اصلی باز می‌گرداند یک انشعاب غیرمستقیم است که بخش آدرس آن 0 است. این دستور در انتهای برنامه سرویس I/O[1] قرار دارد. پس از اینکه این دستور از حافظه دریافت شده کنترل به فاز غیرمستقیم (زیرا I=1) رفته و آدرس مؤثر را می‌خواند. آدرس مؤثر در مکان 0است و همان آدرس برگشتی است که در لحظه بروز وقفه ذخیره شده بود. اجرای دستور غیرمستقیم BUN نشاندن آدرس برگشت در PC است.

سیکل وقفه

ما حالا آماده‌ایم تا عبارات انتقال ثبات را برای سیکل وقفه بنویسیم. سیکل وقفه پس از آخرین فاز اجرا، اگر فلیپ فلاپ R=1 باشد آغاز می‌شود. این فلیپ فلاپ به شرطی 1 می‌شود. که IEN=1 و یکی از دو فلیپ فلاپ FGI یا FGO برابر 1 باشد. این وضعیت در هر انتقال پالس ساعتی امکان پذیر است

1- Input/Output

بجز وقتی که سیگنال‌های T_0، T_1 و یا T_2 فعال باشند. شرایط 1 شدن فلیپ فلاپ R با عبارت انتقال ثبات زیر می‌تواند بیان شود:

$$T'_0T'_1T'_2\ (IEN)\ (FGI+FGO):\ R_1 \leftarrow 1$$

نماد + بین FGI و FGO در تابع کنترل یک عمل منطقی OR را بیان می‌دارد. این عمل با IEN و $T'_0T'_1T'_2$، AND می‌شود.

حال ما فازهای برداشت و دیکد سیکل دستور را تصحیح می‌کنیم. بجای استفاده از T_0، T_1 و T_2 (طبق شکل ۹-۵) سه سیگنال مذکور را با R'، AND می‌نمائیم در نتیجه فازهای برداشت و دیکد با سه تابع کنترل $R'T_0$، $R'T_1$ و $R'T_2$ تعریف می‌شوند. دلیل این مورد این است که پس از فاز اجرای دستورالعمل و پاک شدن SC، کنترل فقط با شرط R=0 به فاز برداشت خواهد رفت. در غیراینصورت اگر R=1 باشد کنترل به سیکل وقفه خواهد رفت. سیکل وقفه آدرس برگشت (موجود در PC) را در آدرس 0 ذخیره ساخته و سپس به حافظه 1 انشعاب کرده و IEN و R و SC را هم 0 می‌کند. این اعمال با رشته ریزعمل های زیر امکان پذیر است.

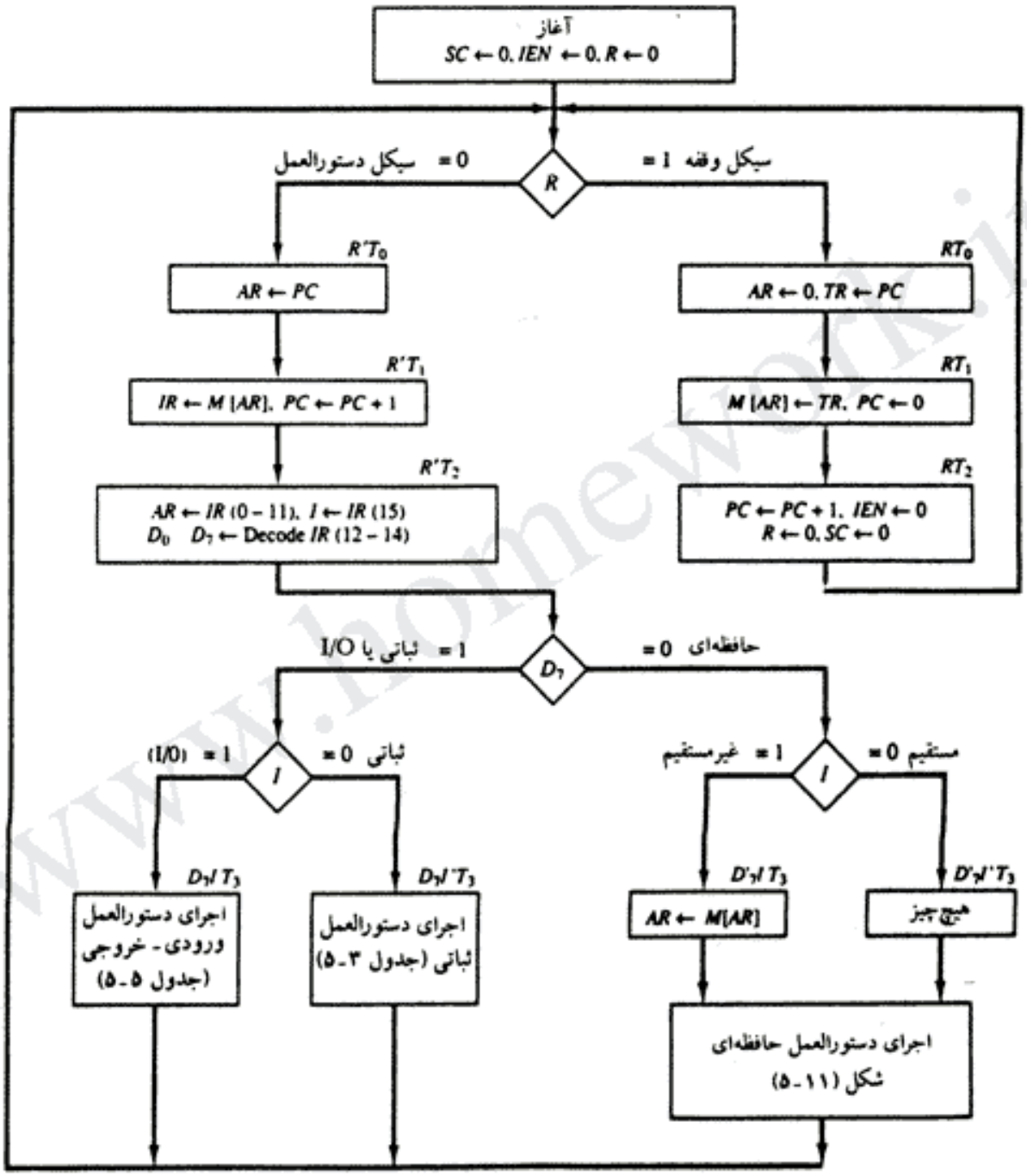
RT_0: AR ← 0 , TR ← PC
RT_1: M[A R] ← TR , PC ← 0
RT_2: PC ← PC+1 , IEN ← 0 , R ← 0 , SC ← 0

در طول اولین سیگنال زمانبندی، AR پاک می‌شود و PC به ثبات موقتی بنام TR انتقال می‌یابد. با دومین سیگنال آدرس برگشت در مکان 0 حافظه ذخیره شده و PC پاک می‌شود. سومین سیگنال زمانبندی PC را یک واحد افزایش می‌دهد، IEN و R را پاک می‌کند و کنترل نیز با پاک شدن SC به T_0 باز می‌گردد. شروع دستورالعمل بعدی شرط $R'T_0$ را داراست و محتویات PC نیز برابر 1 است. سپس کنترل وارد سیکل دستور شده و دستور BUN را از حافظه 1 برداشت و اجرا می‌کند.

۵-۸ تشریح کامل کامپیوتر

فلوچارت نهایی سیکل دستورالعمل به همراه سیکل وقفه برای کامپیوتر پایه در شکل ۵-۱۵ نشان داده شده است. فلیپ فلاپ R هر لحظه ممکن است در طول فازهای غیرمستقیم[1] یا اجرا[2] برابر 1 شود. کنترل پس از SC=0 به T_0 برمی‌گردد. اگر R=1 باشد کنترل وارد سیکل وقفه می‌گردد. اگر R=0 باشد کامپیوتر وارد سیکل دستورالعمل می‌شود. اگر دستور ازنوع ارجاع به حافظه باشد، کامپیوتر در صورت وجود آدرس غیرمستقیم اجرای دستور دیکد شده را طبق شکل ۵-۱۱ ادامه می‌دهد. اگر دستورالعمل از نوع ارجاع به ثبات باشد، اجرای آن طبق جدول ۵-۳ خواهد بود و بالاخره اگر دستور از نوع ورودی -

1- Indirect 2- Execute



شكل ۱۵-۵ فلوچارت براي اعمال كامپيوتر

خروجی باشد، اجرای آن با یکی از ریزعمل‌های جدول ۵-۵ می‌باشد.

طرز کار کامپیوتر را می‌توان با لیستی از عبارات انتقال ثبات بجای فلوچارت نشان داد. این عمل با سرجمع کردن تمام ریز اعمال در یک جدول انجام می‌شود. جدول مذکور با استفاده از شکل‌های ۱۱-۵ و ۱۵-۵ و جداول ۳-۵ و ۵-۵ تکمیل گردیده است.

توابع کنترل و ریزاعمال برای کل کامپیوتر درجدول ۶-۵ آورده شده است. در این جدول عبارات انتقال ثبات، سازمان داخلی کامپیوتر را بفرم فشرده توصیف می‌کنند. این عبارات اطلاعات لازم را برای طراحی مدارهای منطقی نیز فراهم می‌آورند. توابع کنترل و عبارات کنترل شرطی که در جدول لیست شده‌اند توابع بول گیت‌ها را در واحد کنترل فرموله می‌کنند. لیست ریز اعمال، نوع ورودی‌های کنترل لازم برای ثبات‌ها و حافظه‌ها را معین می‌سازد. یک عبارت انتقال ثبات نه فقط برای توصیف سازمان داخلی یک سیستم دیجیتال مفید است بلکه برای مشخص کردن مدارهای منطقی لازم در طراحی نیز می‌باشد.

۹-۵ طراحی کامپیوتر پایه

کامپیوتر پایه متشکل از بخش‌های سخت افزاری زیر است.

۱- یک واحد حافظه با 4096 کلمه 16 بیتی

۲- 9 ثبات: AR، PC، DR، AC، IR، TR، OUTR، INPR و SC

۳- 7 فلیپ فلاپ: I، S، E، R، IEN، FGI و FGO

۴- دو دیکدر: یک دیکدر عملیاتی 3×8 و یک دیکدر زمانی 4×16

۵- یک گذرگاه مشترک 16 بیتی

۶- کنترل منطقی گیتی

۷- جمع کننده و مدار منطقی متصل به ورودی AC

واحد حافظه قطعه استانداردی است و بصورت آماده قابل تهیه از منابع تجاری است. ثبات‌ها از نوع شکل ۱۱-۲ و مشابه آی سی 74163 می‌باشند. فلیپ فلاپها می‌توانند از نوع D یا JK مطابق بخش ۶-۱ باشند. دو دیکدر نیز استاندارد و مشابه آنچه در بخش (۲-۲) گفته شد می‌باشند. سیستم گذرگاه مشترک می‌تواند با شانزده مولتی پلکسر 8×1 با آرایشی مشابه با شکل ۳-۴ ساخته شود. حالا تصمیم داریم نشان دهیم که چگونه مدار منطقی کنترل را می‌توان طراحی کرد. بخش بعد طراحی جمع کننده و مدارهای منطقی مربوط به AC را بحث می‌کند.

گیت‌های مدار کنترل

دیاگرام مدار کنترل درشکل ۶-۵ دیده می‌شود. ورودی‌های این مدار شامل دو دیکدر، فلیپ فلاپ I و بیت های 0 تا 11 ثبات IR می‌باشند. سایر ورودی‌ها به کنترل عبارتند از بیت‌های 0 تا 15 ثبات AC

جدول ۶-۵ توابع کنترل و اعمال جزیی کامپیوتر پایه

برداشت	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
دیکد	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{-}14)$, $AR \leftarrow IR(0\text{-}11)$, $I \leftarrow IR(15)$
غیرمستقیم	D_7IT_3:	$AR \leftarrow M[AR]$
وقفه:		
	$T_4T_1T_2(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
حافظه‌ای:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
ثباتی:		
	$D_7I'T_3 = r$	(مشترک در همه دستورالعمل‌های ثباتی)
	$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$	
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
ورودی - خروجی:		
	$D_7IT_3 = p$	(مشترک در همه دستورالعمل‌های ورودی خروجی)
	$IR(i) = B_i$ $(i = 6, 7, 8, 9, 10, 11)$	
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7)$, $FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

www.homework.ir

برای کنترل AC=0 و کنترل بیت علامت AC(15)؛ بیت‌های 0 تا 15 ثبات DR برای کنترل DR=0 و مقادیر هفت فلیپ فلاپ.

خروجی‌های مدار کنترل منطقی عبارتند از:

۱- سیگنال‌های کنترل ورودی 9 عدد ثبات

۲- سیگنال‌های کنترل ورودی‌های خواندن و نوشتن

۳- سیگنال‌های نشاندن[1]، پاک کردن[2] و مکمل‌سازی فلیپ فلاپها

۴- سیگنال S_2،S_1 و S_0 برای انتخاب ثبات برای گذرگاه

۵- سیگنال‌های کنترل مدار منطقی جمع کننده AC

مشخصات انواع سیگنال‌های کنترل مستقیماً از لیست عبارات انتقال ثبات در جدول ۶-۵ قابل‌دسترسی‌است.

کنترل ثبات‌ها و حافظه

ثبات‌های کامپیوتر متصل به سیستم گذرگاه مشترک در شکل ۴-۵ دیده می‌شوند. ورودی‌های کنترل ثبات‌ها LD (بارشدن) ، INR (افزایش) و CLR (پاک کردن) می‌باشند. فرض کنید که می‌خواهیم ساختار گیتی ورودی‌های کنترل AR را بدست آوریم. برای این منظور جدول ۶-۵ را برای یافتن تمام عباراتی که محتوای AR را تغییر می‌دهد جستجو می‌کنیم.

$$R'T_0 : AR \leftarrow PC$$
$$R'T_2 : AR \leftarrow IR\ (0\text{-}11)$$
$$D'_7IT_3 : AR \leftarrow M[AR]$$
$$RT_0 : AR \leftarrow 0$$
$$D_5T_4 : AR \leftarrow AR+1$$

سه عبارت اول بیانگر انتقال اطلاعات از یک ثبات یا حافظه به AR است. محتوای ثبات مبدأ یا حافظه روی گذرگاه قرار گرفته و محتویات گذرگاه هم با تواناسازی ورودی کنترل LD منتقل می‌شود. عبارت چهارم AR را 0 می‌کند. عبارت آخر هم AR را یک واحد افزایش می‌دهد. توابع کنترل می‌توانند با هم ترکیب و با سه عبارت بولی نشان داده شوند.

$$LD\ (AR) = R'T_0 + R'T_2 + D'_7IT_3$$
$$CLR\ (AR) = RT_0$$
$$INR\ (AR) = D_5\ T_4$$

1- Set 2- Clear

که در آن‌ها LD(AR) ورودی بارکردن AR، CLR (AR) ورودی پاک کردن AR و INR(AR) ورودی افزایش AR است. مدار منطقی کنترل گیتی AR در شکل ۱۶-۵ دیده می‌شود.
بطریق مشابه می‌توان مدار کنترل سایر ثبات‌ها و نیز ورودی‌های کنترل حافظه را نیز بدست آورد. گیت‌های منطقی مربوط به ورودی خواندن با جستجو در جدول ۶-۵ برای یافتن عبارات خواندن نیز امکان پذیر است. نشانه عمل خواندن با نماد M[AR]← قابل تشخیص است.

$$\text{Read: } R'T_1 + D'_7IT_3 + (D_0 + D_1 + D_2 + D_6)\,T_4$$

خروجی گیت‌های منطقی‌که عبارت فوق را پیاده‌سازی می‌کنند باید به ورودی خواندن حافظه متصل‌گردد.

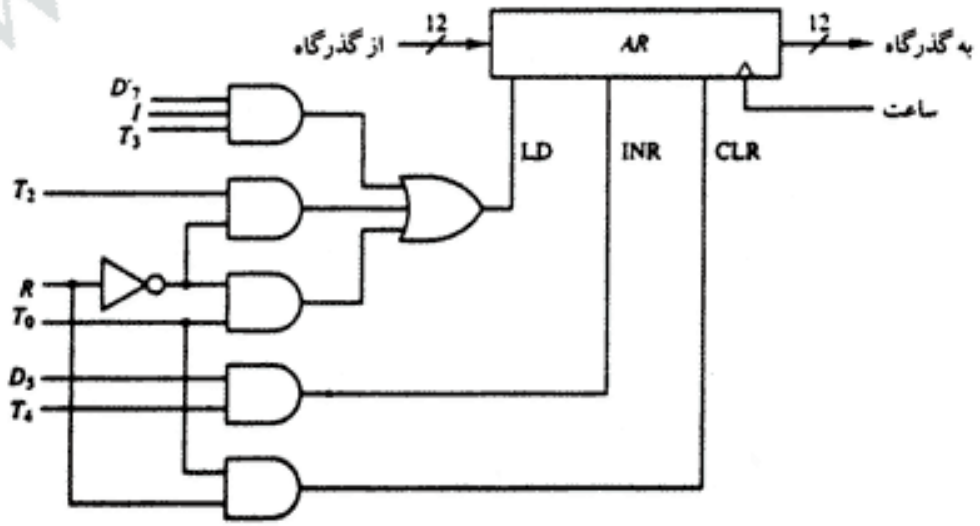
کنترل فلیپ فلاپ های منفرد

گیت‌های کنترل هفت فلیپ فلاپ بروش مشابهی تعیین می‌شود. بعنوان مثال جدول ۶-۵ نشان می‌دهد که IEN توسط دو دستورالعمل ممکن است تغییر یابد.

$$pB_7: \text{IEN} \leftarrow 1$$
$$pB_6: \text{IEN} \leftarrow 0$$

که در آن $p=D_7IT_3$ و B_7 و B_6 بیت‌های 7 و 6 از ثبات IR هستند. بعلاوه در انتهای سیکل وقفه هم IEN، پاک می‌شود.

$$RT_2: \text{IEN} \leftarrow 0$$



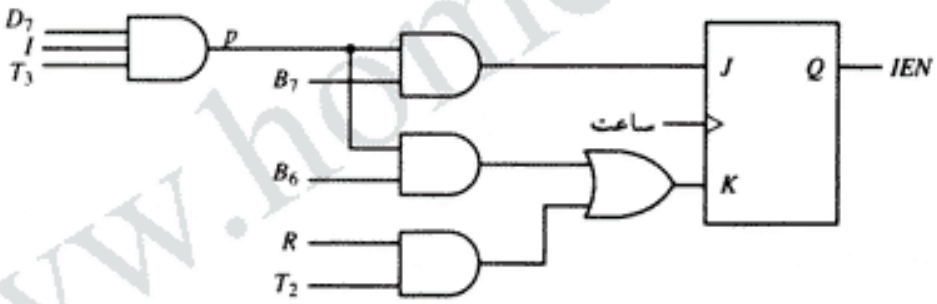
شکل ۱۶-۵ گیت‌های کنترل AR

چنانچه یک فلیپ فلاپ JK برای IEN بکار رود، مدار کنترل منطقی مطابق شکل ۱۷-۵ خواهد بود.

کنترل گذرگاه مشترک

گذرگاه 16 بیتی شکل ۴-۵ توسط ورودی‌ها انتخاب S_2، S_1 و S_0 کنترل می‌شود. عدد اعشاری مقابل هر ورودی گذرگاه عدد معادل دودویی اعمال شده به ورودی های کنترل است تا ثبات مربوط به آن عدد انتخاب شود. جدول ۷-۵ اعداد اعمال شده به $S_2S_1S_0$ برای انتخاب هر ثبات را نشان می‌دهد. هر عدد دودویی مرتبط با یک متغیر بولی x_1 تا x_7 است که با ساختار گیتی که باید برای انتخاب ثبات یا حافظه فعال باشد مرتبط است. بعنوان مثال وقتی $x_1=1$ است مقدار $S_2S_1S_0$ باید 001 باشد و خروجی AR برای گذرگاه انتخاب می‌شود. جدول ۷-۵ می‌تواند بعنوان جدول درستی انکدر دودویی در نظر گرفته شود. مکان قرار گرفتن انکدر در ورودی‌های انتخاب گر در شکل ۱۸-۵ دیده می‌شود. توابع بول برای انکدر برابر است با:

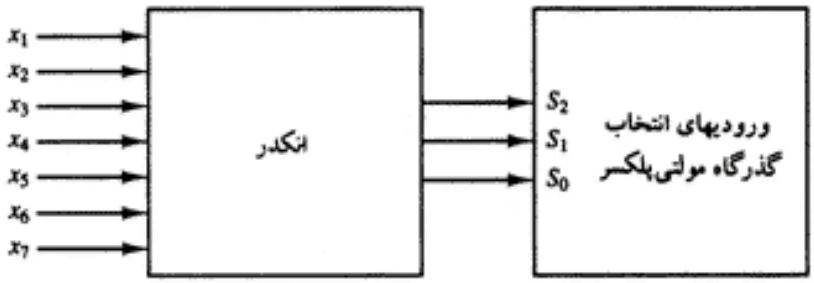
$$S_0 = x_1 + x_3 + x_5 + x_7$$
$$S_1 = x_2 + x_3 + x_6 + x_7$$
$$S_2 = x_4 + x_5 + x_6 + x_7$$



شکل ۱۷-۵ ورودی‌های کنترل برای IEN

جدول ۷-۵ انکدر برای مدار انتخاب گذرگاه

ورودیها							خروجیها			ثبات انتخاب شده برای گذرگاه
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	هیچکدام
1	0	0	0	0	0	0	0	0	1	*AR*
0	1	0	0	0	0	0	0	1	0	*PC*
0	0	1	0	0	0	0	0	1	1	*DR*
0	0	0	1	0	0	0	1	0	0	*AC*
0	0	0	0	1	0	0	1	0	1	*IR*
0	0	0	0	0	1	0	1	1	0	*TR*
0	0	0	0	0	0	1	1	1	1	حافظه



شکل ۱۸-۵ انکدر برای ورودی‌های انتخاب گذرگاه

برای تعیین مدار منطقی ورودی انکدر، لازم است توابع کنترلی که ثبات مربوطه را روی گذرگاه قرار دهند تعیین گردد. برای مثال، برای یافتن مدار منطقی مربوط به $x_1=1$ ما تمام عبارات انتقال ثبات در جدول ۶-۵ را جستجو کرده و عباراتی که در آنها AR منبع است را استخراج می‌کنیم.

$$D_4\ T_4: PC \leftarrow AR$$
$$D_5\ T_5: PC \leftarrow AR$$

بنابراین، تابع بول x_1 برابرست با

$$x_1 = D_4T_4 + D_5T_5$$

خروجی داده حافظه وقتی انتخاب می‌شود که $x_7=1$ و $S_2S_1S_0=111$ باشد. مدار گیتی که x_7 را تولید می‌کند نیز باید به ورودی خواندن حافظه وصل شود. بنابراین تابع بول برای x_7 مشابه آنچه قبلاً برای "خواندن" بدست آمد خواهد بود.

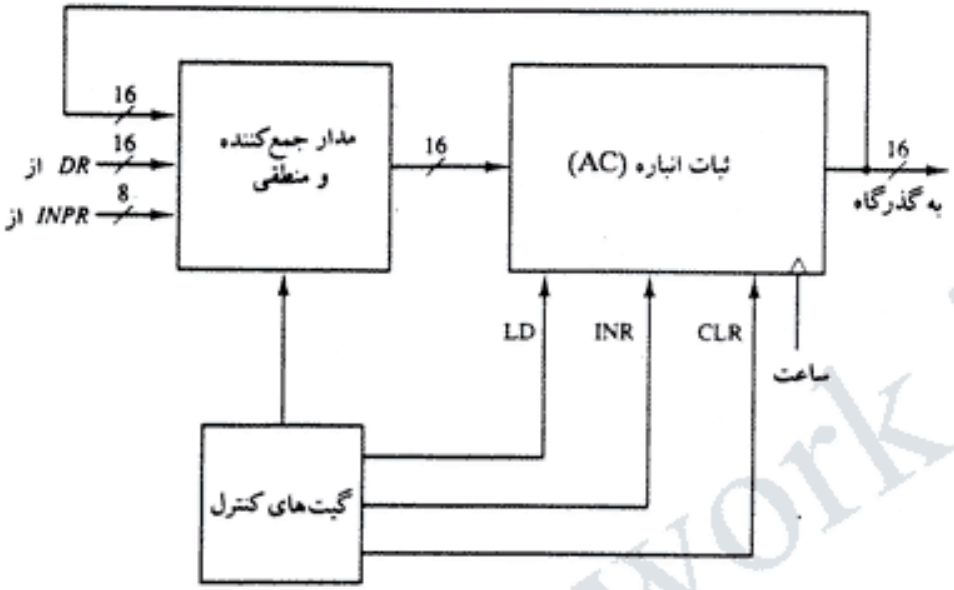
$$x_7 = R'T_1 + D'_7IT_3 + (D_0 + D_1 + D_2 + D_6)\ T_4$$

به‌روشی مشابه می‌توانیم مدارگیتی را برای سایر ثبات ها بدست آوریم

۱۰-۵ طراحی مدار منطقی انباره[1]

مدارهای مربوط به ثبات AC در شکل ۱۹-۵ دیده می‌شود. جمع کننده و مدار منطقی دارای سه مجموعه ورودی است. یک مجموعه 16 ورودی از خروجی AC می‌آید. مجموعه 16 ورودی دیگر از DR می‌رسد. سومین مجموعه هشت ورودی از ثبات ورودی INPR گرفته شده است. خروجی جمع کننده و مدار منطقی ورودی‌های داده را برای ثبات AC فراهم می‌آورند. بعلاوه مدارهای منطقی لازم

1- Accumlator



شکل ۱۹-۵ مدار مربوط به AC

برای کنترل LD، INR و CLR در ثبات و کنترل جمع کننده و مدار منطقی مربوطه نیز باید مد نظر باشند.

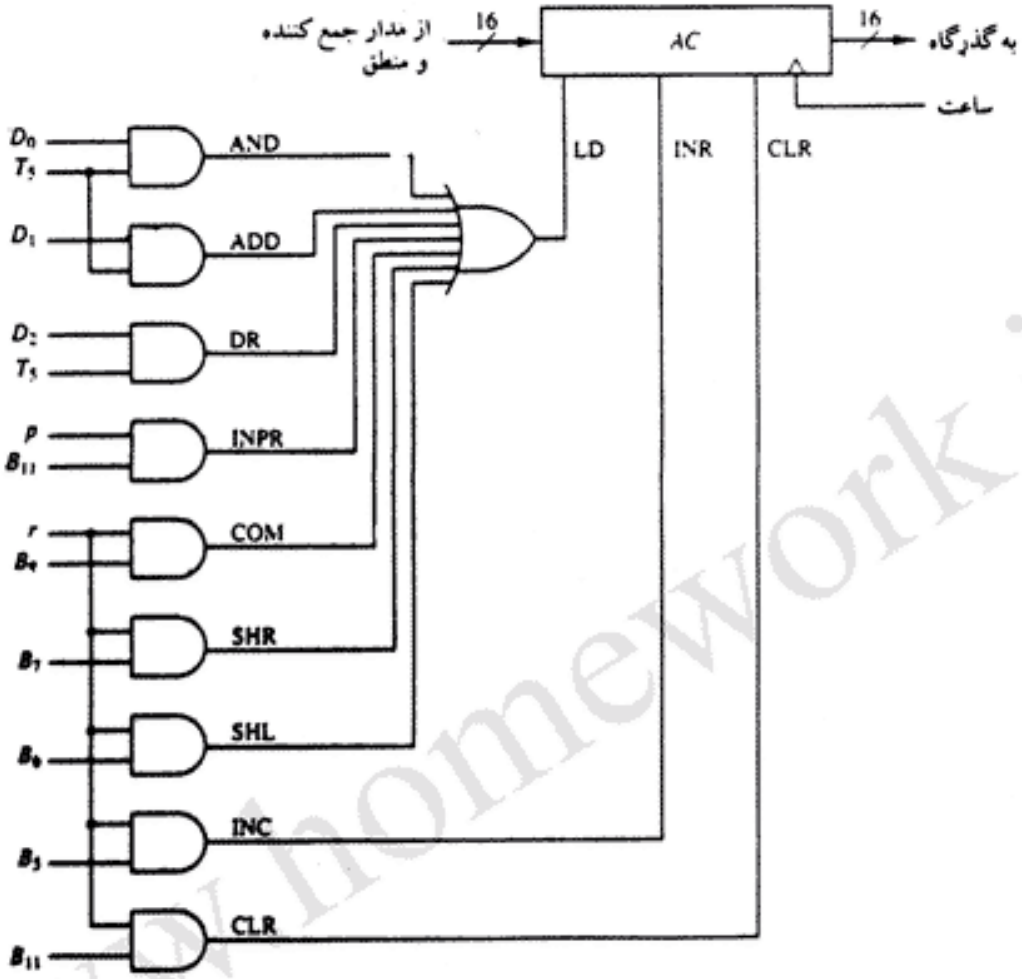
برای طراحی مدار منطقی مربوط به AC، باید به عبارات انتقال ثبات در جدول ۶-۵ مراجعه و تمام عباراتی که محتویات AC را تغییر می‌دهند استخراج شوند.

D_0T_5:	$AC \leftarrow AC \wedge DR$	AND با DR
D_1T_5:	$AC \leftarrow AC + DR$	Add با DR
D_2T_5:	$AC \leftarrow DR$	انتقال از DR
pB_{11}:	$AC(0\text{-}7) \leftarrow INPR$	انتقال از INPR
rB_9:	$AC \leftarrow \overline{AC}$	متمم
rB_7:	$AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E$	شیفت به راست
rB_6:	$AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E$	شیفت به چپ
rB_{11}:	$AC \leftarrow 0$	پاک
rB_5:	$AC \leftarrow AC + 1$	افزایش

از این لیست می‌توانیم مدار منطقی جمع کننده و گیت های منطقی آنرا بدست آوریم

کنترل ثبات AC

ساختار گیتی کنترل‌کننده ورودی‌های AC یعنی LD، INR و CLR در شکل ۲۰-۵ نشان داده شده‌اند. پیکربندی گیت‌ها از توابع کنترل در لیست فوق بدست آمده‌اند. تابع کنترل برای ریزعمل‌های پاک



شکل ۲۰-۵ ساختار گیت‌های مربوط به کنترل در ورودی‌های LD، INR و CLR از AC

کردن[1]، rB_{11} است که در آن $r = D_7I'T_3$ و $B_{11} = IR(11)$ می‌باشد. خروجی گیت AND که این تابع کنترل را تولید می‌کند به ورودی CLR ثبات متصل است. بطور مشابه خروجی گیتی که ریزعمل افزایش[2] را پیاده سازی می‌نماید به ورودی INR ثبات متصل است. هفت ریز عمل دیگر در جمع کننده و مدار منطقی تولید و در زمان معینی به AC اعمال می‌شوند. خروجی‌های گیت‌ها برای هر تابع کنترل با یک نام سمبلیک علامت گذاری شده است. این خروجی‌ها در طراحی جمع کننده و مدار منطقی بکار می‌روند.

جمع‌کننده و مدار منطقی

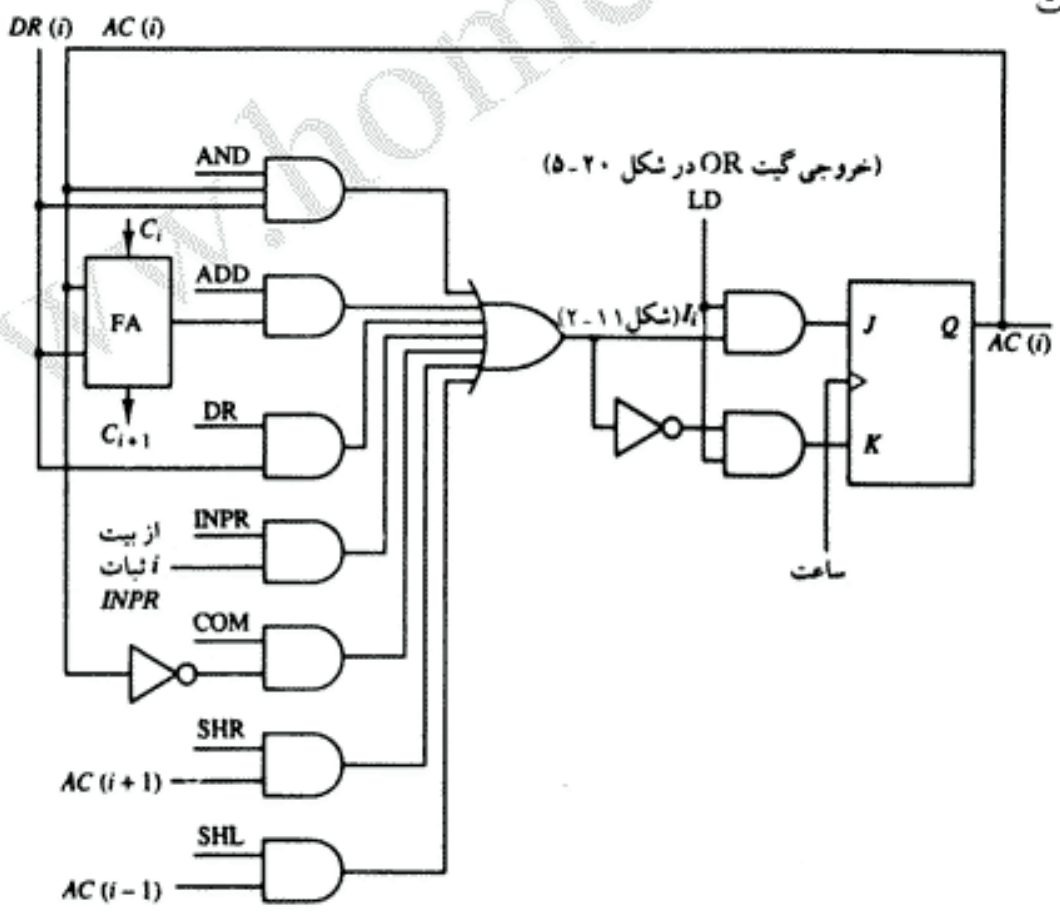
جمع کننده و مدار منطقی مربوطه می‌تواند به 16 قسمت شود که هر قسمت مربوط به یک بیت از

1- Clear
2- Increment

AC است. ساختمان درونی ثبات مطابق شکل ۱۱-۲ است. با بازگشت به این شکل می‌بینیم که هر گیت دارای یک فلیپ فلاپ JK، دو گیت OR و دو گیت AND است. ورودی بار (LD) به ورودی گیت‌های AND وصل است. شکل ۲۱-۵ این بخش از ثبات AC را نشان می‌دهد (یک گیت OR حذف شده است). ورودی با I_i و خروجی با AC(i) نام گذاری شده است. وقتی که ورودی LD فعال است، 16 ورودی I_i که در آن i=0,1,2,...15 است به AC(0-15) منتقل می‌شوند.

یک طبقه از مدار جمع کننده و منطقی از هفت گیت AND ، یک گیت OR و یک تمام جمع کننده (FA) مطابق شکل ۲۱-۵ ساخته شده است. ورودی این گیت‌ها با نام‌هایی سمبلیکی از خروجی گیت‌ها با همان نام در شکل ۲۰-۵ آمده‌اند. بعنوان مثال، ورودی علامت زده شده با ADD در شکل ۲۱-۵ به خروجی مشابه با همان نام در شکل ۲۰-۵ وصل است.

عمل AND با AND کردن AC(i) با بیت متناظرش درثبات داده DR(i) صورت می‌گیرد. عمل ADD هم با جمع‌کننده‌ای مشابه با شکل ۶-۴ انجام می‌شود. در طبقه جمع کننده از یک تمام جمع کننده با ورودی و خروجی نقلی استفاده شده است. انتقال از INPR به AC فقط از بیت 0 تا 7 میسر است. ریزعمل مکمل سازی با معکوس سازی مقادیر بیت‌های AC بدست می‌آید. عمل شیفت به راست



شکل ۲۱-۵ یک طبقه از مدار جمع کننده و منطق

انتقال را از بیت AC(i+1) به AC (i) و شیفت به چپ انتقال را از بیت AC(i−1) به AC (i) انجام می‌دهد. مدار کامل جمع‌کننده و مدار منطقی از 16 بخش که بهم متصلند تشکیل شده‌اند.

مسائل

۵-۱ یک کامپیوتر از حافظه‌ای با 256k کلمه 32 بیتی استفاده می‌کند. یک دستورالعمل دودویی در یک کلمه از حافظه ذخیره شده است. دستورالعمل چهاربخش دارد: بیت غیرمستقیم، یک کد عملیاتی، یک کد ثبات برای تعیین یکی از 64 ثبات و بخش آدرس.
الف) چند بیت در کد عملیاتی، کد ثبات و آدرس وجود دارد؟
ب) قالب کلمه دستورالعمل را ترسیم و تعداد بیت در هر قسمت را معین کنید.
پ) در ورودی‌های داده و آدرس حافظه چند بیت وجود دارد.

۵-۲ اختلاف بین دستور با آدرس مستقیم و غیرمستقیم چیست؟ چند ارجاع به حافظه برای هر نوع دستورالعمل لازم است تا عملوند را به ثبات پردازشگر منتقل کند؟

۵-۳ ورودی‌های کنترل زیر در سیستم گذرگاه شکل ۴-۵ فعالند. برای هر حالت انتقال ثباتی که در پالس ساعت بعدی اجرا شود را معین کنید.

	S_2	S_1	S_0	ثبات LD	حافظه	جمع‌کننده
(الف	1	1	1	*IR*	خواندن	—
(ب	1	1	0	*PC*	—	—
(ج	1	0	0	*DR*	نوشتن	—
(د	0	0	0	*AC*	—	جمع

۵-۴ انتقال ثبات‌های زیر قرار است در سیستم شکل ۴-۵ اجرا شوند. برای هر انتقال: (۱) مقداردودویی که باید به ورودی‌های انتخاب گذرگاه S_2، S_1 و S_0 اعمال شوند را معین کنید؛ (۲) ثباتی که کنترل LD آن باید فعال شود (اگر وجود دارد)؛ (۳) عمل نوشتن یا خواندن حافظه (اگر لازم است)؛ و (۴) عمل در جمع‌کننده و مدار منطقی (اگر وجود دارد).

الف) AR ← PC
ب) IR ← M[AR]
ج) M[AR] ← TR
د) AC ← DR, DR ← AC

۵-۵ توضیح دهید چرا هیچک از ریزعمل‌های زیر در طول یک پالس ساعت در سیستم شکل ۴-۵ اجرا نمی‌شوند. رشته ریزاعمال لازم برای انجام عملیات را معین کنید

الف) IR ← M[PC]
ب) AC ← AC + TR

ج) $DR \leftarrow DR + AC$ (AC تغییر نمی‌کند)

۵-۶ قالب دستورات کامپیوتر پایه شکل ۵-۵ و لیست دستورات جدول ۲-۵ را ملاحظه کنید. برای هریک از دستورات 16 بیتی، کد معادل چهاربیتی مبنای شانزده را نوشته و بزبان ساده بگوئید این دستور چه کاری انجام می دهد.

الف) 0001 0000 0010 0100

ب) 1011 0001 0010 0100

ج) 0111 0000 0010 0000

۵-۷ کدام دودستور برای 1کردن فلیپ فلاپ E در کامپیوتر پایه بکار می‌روند؟

۵-۸ یک دیاگرام زمانبندی مشابه شکل ۵-۷ ترسیم و فرض کنید SC در T_3 برابر 0 شده باشد بشرط اینکه سیگنال کنترل C_7 فعال باشد.

$$C_7T_3: SC \leftarrow 0$$

C_7 با لبه پالس مربوط به T_1 فعال می‌شود.

۵-۹ محتویات AC در کامپیوتر پایه عدد مبنای شانزده A937 است و مقدار اولیه E برابر 1 است. محتویات AC، E، PC، AR و IR در مبنای 16 پس ازاجرای دستور CLA چیست. عمل قبل را 11 بار با هر یک از دستورالعمل‌ها تکرار کنید. مقدار اولیه PC را عدد مبنای شانزده 021 فرض کنید.

۵-۱۰ دستورالعملی در آدرس 021 کامپیوتر پایه دارای I=0، کد عملیاتی AND و آدرس 083 است (تمام ارقام در مبنای شانزده است). کلمه حافظه واقع در آدرس 083 دارای عملوند B8F2 و محتویات AC هم A937 است. درطول سیکل دستور محتویات ثبات‌های زیر را در پایان فاز اجرا معین کنید: PC، AR، DR، AC و IR. مسئله را شش بار دیگر برای دستورالعمل ارجاع به حافظه دیگری تکرار کنید.

۵-۱۱ محتویات ثبات‌های PC، AR، DR، IR و SC در مبنای شانزده وقتی که در کامپیوتر پایه دستور غیرمستقیم ISZ دریافت و اجرا شود چیست. مقدار اولیه PC را 7FF در نظر بگیرید. محتویات حافظه در آدرس 7FF برابر EA9F است. محتویات حافظه در آدرس A9F هم 0C35 می‌باشد. محتویات حافظه C35 برابر با FFFF می‌باشد. پاسخ خود را بصورت جدولی با پنج ستون با هر ستون برای یک ثبات، و هر سطر برای یک سیگنال زمان بندی تهیه کنید. محتویات هر ثبات را پس از لبه مثبت هر پالس ساعت نشان دهید.

۵-۱۲ محتویات PC در کامپیوتر پایه 3AF است (تمام اعداد در مبنای شانزده). محتویات AC هم 7EC3 است. محتویات حافظه آدرس 3AF برابر با 932E می‌باشد. محتویات حافظه در آدرس 32E برابر 09AC و در آدرس 9AC هم 8B9F است

الف) دستورالعملی که بعداً دریافت و اجرا شود چیست؟
ب) عمل دودویی که در AC پس از اجرای دستورالعمل رخ می‌دهد چیست
ج) محتویات ثبات‌های PC، AR، DR، AC و IR در مبنای شانزده چیست. همچنین مقادیر E و I و SC درانتهای سیکل دستورالعمل را معین کنید.

۵-۱۳ فرض کنید که شش دستور ارجاع به حافظه در کامپیوتر پایه در جدول ۴-۵ با جدول زیر تعویض شوند. EA آدرس موثر واقع در AR در T_4 است. فرض کنید که جمع کننده و مدار منطقی شکل ۴-۵ می‌تواند عمل XOR را انجام دهد $AC \leftarrow AC \oplus DR$. بعلاوه فرض کنید که جمع‌کننده و مدار منطقی نمی‌توانند مستقیماً تفریق را انجام دهند. تفریق باید بکمک مکمل 2 انجام‌شود. رشته عبارات انتقال ثبات لازم برای اجرای هر دستور لیست شده را از T_4 به بعد مشخص کنید. دقت کنید که هیچ تغییری در AC رخ نمی‌دهد مگر اینکه دستورالعمل تغییری را در آن معین کند. شما می‌توانید با استفاده از TR محتویات AC راموقتاً ذخیره و یا محتویات AC و DR را باهم عوض کنید.

سمبل	کد عمل	نمایش سمبلیک	توضیح
XOR	000	$AC \leftarrow AC \oplus M[EA]$	OR انحصاری با AC
ADM	001	$M[EA] \leftarrow M[EA] + AC$	جمع AC با حافظه
SUB	010	$AC \leftarrow AC - M[EA]$	تفریق حافظه از AC
XCH	011	$AC \leftarrow M[EA],\ M[EA] \leftarrow AC$	تبادل AC با حافظه
SEQ	100	If $(M[EA] = AC)$ then $(PC \leftarrow PC + 1)$	گذر در صورت برابری
BPA	101	If $(AC > 0)$ then $(PC \leftarrow EA)$	انشعاب اگر AC مثبت و غیر صفر باشد

۵-۱۴ تغییرات زیر را در کامپیوتر پایه بعمل آورید.
۱- یک ثبات CTR (ثبات شمارنده) را به سیستم گذرگاه اضافه کنید و آنرا با $S_2S_1S_0 = 000$ انتخاب نمایید.
۲- ISZ را با دستوری که یک عدد را در CTR بار کند عوض کنید.

LDC Address CTR ← M[address]

۳- یک دستور ارجاع به ثبات ICSZ به مجموعه اضافه کنید: CTR را یک واحد اضافه کرده و از اجرای دستور بعدی اگر حاصل افزایش صفر است صرفنظر نمائید. مزیت این تغییر را بیان کنید.

۵-۱۵ واحد حافظه کامپیوتر پایه در شکل ۵-۱۳ را با یک حافظه 16 × 65536 عوض کنید . این

حافظه‌آدرس 16 بیتی نیاز دارد. قالب دستورالعمل ارجاع به حافظه در شکل (۵-۵ (الف) برای I=1 بلاتغییرو بخش آدرس در مکان‌های 0 تا 11 قرار دارد. اما وقتی I=0 است (آدرس مستقیم) آدرس‌دستورالعمل توسط 16 بیت در کلمه دیگری که بدنبال دستور آمده است داده شده است. ریز اعمال‌را در T_2 ، T_3 (و T_4 اگر لازم باشد) تصحیح کنید تا با این پیکربندی هماهنگ باشد.

۵-۱۶ کامپیوتری از یک حافظه هشت بیتی 65536 کلمه‌ای استفاده می‌کند. این کامپیوتر دارای ثبات‌های PC ، AR، TR (هریک شانزده بیت) و AC و DR و IR (هریک هشت بیت) است. یک دستور ارجاع به حافظه متشکل از سه کلمه است: یک کد عملیات 8 بیتی (یک کلمه) و یک آدرس 16 بیتی (در دو کلمه بعدی). تمام عملوندها هشت بیت هستند. بیت‌غیرمستقیم هم وجود ندارد.

الف) بلاک دیاگرامی از کامپیوتر ترسیم کنید و ثبات‌هاو حافظه‌ها را مطابق شکل ۳-۵ نشان دهید. (از یک گذرگاه مشترک استفاده نکنید).

ب) طرزقرارگرفتن یک نمونه دستور سه کلمه‌ای را به همراه عملوند 8 بیتی در حافظه نشان دهید.

رشته ریزاعمال برای دریافت یک دستور ارجاع به حافظه را لیست کنید و سپس عملوند را در DR قرار دهید. از سیگنال زمانی T_0 شروع کنید.

۵-۱۷ یک کامپیوتر دیجیتال دارای 16384 حافظه 40 بیتی در هر کلمه است. قالب کد دستور از شش بیت برای عملوند و 14 بیت برای آدرس تشکیل شده است. (بیت غیرمستقیم ندارد). دو دستورالعمل در یک کلمه جای داده شده‌اند و یک ثبات دستورالعمل 40 بیتی IR هم در واحد کنترل وجود دارد. برنامه‌ای را برای فازهای برداشت و اجرا در این کامپیوتر بنویسید.

۵-۱۸ یک برنامه خروجی از آدرس 2300 نوشته شده است. این برنامه وقتی کامپیوتر یک وقفه را در FGO=1 تشخیص دهد اجرا می‌گردد (در حالیکه IEN=1 است)

الف - چه دستوری باید در آدرس 1 قرار گیرد؟

ب - دو دستور آخر برنامه خروجی چیست؟

۵-۱۹ عبارات انتقال ثبات برای ثبات R و حافظه در یک کامپیوتر مطابق زیر است (Xها توابع کنترل هستند و بطور تصادفی رخ می‌دهند).

$X'_3\ X_1$: R← M [AR]	کلمه حافظه را در R بنویس
$X'_1\ X_2$: R← AC	انتقال AC به R
$X'_1\ X_3$: M [AR] ← R	R را در حافظه بنویس

حافظه دارای ورودی‌های داده، خروجی‌های داده، ورودی‌های آدرس و ورودی‌های کنترل برای

خواندن و نوشتن مطابق شکل ۱۲-۲ است. سخت افزار R و حافظه را بشکل بلاک دیاگرام بکشید نشان دهید که چگونه توابع کنترل X_1 تا X_3 ورودی R، ورودی مولتی پلکسرهائی که شما در دیاگرام وارد کرده‌اید، و ورودی‌های خواندن و نوشتن حافظه را انتخاب می‌کنند.

۵-۲۰ رشته اعمالی که باید توسط فلیپ فلاپ F انجام شوند با عبارت انتقال ثبات داده شده‌اند

1 در F نشانده شود	XT_3: F ← 1
F با 0 پاک شود	yT_1: F← 0
مکمل F	ZT_2: F←$\overline{F}$
مقدار G را به F انتقال بده	WT_5: F ← G

در غیراینصورت F نباید تغییر یابد. دیاگرام منطقی مربوط به اتصالات گیت‌ها که توابع کنترل و ورودی‌های فلیپ فلاپ F تشکیل می‌دهند را ترسیم نمائید. از یک فلیپ فلاپ JK استفاده کرده و تعداد گیت‌هارا حداقل کنید.

۵-۲۱ مدار کنترل گیتی مربوط به شمارنده برنامه PC را در کامپیوتر پایه بدست آورید.

۵-۲۲ مدار کنترل گیتی برای ورودی‌خواندن یک حافظه را در کامپیوتر پایه بدست آورید.

۵-۲۳ مدار کامل منطقی وقفه را در کامپیوتر پایه نشان دهید. از فلیپ فلاپ JK استفاده کرده و گیت‌ها را به حداقل برسانید.

۵-۲۴ عبارت بولی را برای x_2 (جدول ۷-۵ را ملاحظه کنید) بدست آورید. نشان دهید که x_2 می‌تواند با یک گیت AND و یک OR تولید شود.

۵-۲۵ عبارت بولی برای یک مدار گیتی که شمارنده SC را پاک کند بدست آورید. دیاگرام منطقی آنرا رسم نموده و نشان دهید که چگونه خروجی به ورودی‌های INR و CLR از شمارنده SC وصل می‌شود شکل ۶-۵. تعداد گیت‌ها را حداقل نمائید.

٦ برنامه نویسی کامپیوتر پایه

۶-۱ مقدمه
۶-۲ زبان ماشین
۶-۳ زبان اسمبلی
۶-۴ اسمبلر
۶-۵ حلقه در برنامه‌نویسی
۶-۶ برنامه‌نویسی اعمال حسابی و منطقی
۶-۷ زیرروال‌ها
۶-۸ برنامه‌نویسی ورودی - خروجی

۶-۱ مقدمه

یک سیستم کامپیوتری کامل متشکل از سخت‌افزار و نرم‌افزار است. سخت‌افزار از قطعات فیزیکی و همه تجهیزات مرتبط با آنها تشکیل می‌شود. منظور از نرم‌افزار برنامه‌هایی است که برای کامپیوتر نوشته می‌شود. می‌توان از جهات مختلف با نرم‌افزار کامپیوتر، بدون توجه به جزئیات چگونگی عملکرد آن، آشنا شد. همچنین می‌توان بخش‌هایی از سخت‌افزار را بدون اطلاع از توانایی‌های نرم‌افزار طراحی نمود. در هر صورت، افرادی که با معماری کامپیوتر سر و کار دارند باید هم از سخت‌افزار و هم از نرم‌افزار اطلاعاتی داشته باشند زیرا این دو شاخه بر یکدیگر تأثیر می‌گذارند.

نوشتن یک برنامه برای کامپیوتر به معنی مشخص کردن رشته‌ای از دستورالعمل‌های ماشین، به طور مستقیم یا غیر مستقیم است. دستورالعمل‌های ماشین در داخل کامپیوتر الگویی دودویی را تشکیل می‌دهند که کار با آن و یا درک آن، اگر غیر ممکن نباشد، مشکل هست. لذا نوشتن برنامه‌ها با مجموعه کاراکترهای آشناتر ترجیح داده می‌شود. در نتیجه، ترجمه برنامه‌هایی که بر اساس سلیقه کاربر نوشته شده به برنامه‌های قابل تشخیص بوسیله سخت‌افزار، لازم به نظر می‌رسد.

برنامه‌ای که بوسیله یک کاربر نوشته می‌شود ممکن است وابسته و یا مستقل از کامپیوتر فیزیکی باشد که برنامه را اجرا می‌کند. مثلاً برنامه‌ای که بزبان فرترن استاندارد نوشته می‌شود مستقل از ماشین

است، زیرا اکثر کامپیوترها برنامه مترجمی دارند که فرترن استاندارد را به کد دودویی آن کامپیوتر تبدیل می‌کنند. اما خود برنامه مترجم وابسته به ماشین است زیرا باید برنامه فرترن را به کد دودویی قابل تشخیص بوسیله سخت افزار کامپیوتر خاص مورد استفاده ترجمه کند.

این فصل برخی از مفاهیم ابتدایی برنامه نویسی را معرفی می‌کند و رابطه آنها را با نمایش سخت‌افزاری دستورالعمل‌ها نشان می‌دهد. اولین بخش، عملکرد و ساختار اساسی برنامه‌ای که بزبان سمبلیک نوشته شده و بوسیله کاربر به یک برنامه معادل دودویی ترجمه می‌شود را ارائه می‌نماید. تأکید در بحث بیشتر بر روی مفاهیم مهم برنامه مترجم است و نه در چگونگی نوشتن خود برنامه. پس از آن فواید انواع دستورات ماشین به وسیله چند مثال برنامه‌نویسی ساده نشان داده شده است.

مجموعه دستورات پایه، که سازمان سخت‌افزار آن در فصل ۵ بررسی شد، در این فصل به منظور تشریح بسیاری از تکنیک‌های مورد استفاده در برنامه‌نویسی یک کامپیوتر، بکار رفته است. به این ترتیب *امکان خواهد داشت تا رابطه بین یک برنامه و عملیات سخت‌افزاری که دستورات را اجرا می‌کنند مورد تحقیق قرار گیرد.*

۲۵ دستور پایه کامپیوتر در جدول ۱-۶ تکرار شده‌اند تا مراجعه به آنها بهنگام مثال‌های برنامه‌نویسی تسهیل گردد. به هر دستورالعمل یک سمبل سه حرفی اختصاص یافته است تا نوشتن برنامه سمبلیک ساده شود. هفت دستورالعمل اول، دستورات ارجاع به حافظه (حافظه‌ای) است و هجده دستورالعمل بعدی دستورات ارجاع به ثبات (ثباتی) و ورود و خروج هستند. یک دستورالعمل ارجاع به حافظه دارای سه بخش است: بیت روش[1] (شیوه) سه بیت کد عمل، و دوازده بیت آدرس. اولین رقم شانزده شانزدهی یک دستورالعمل حافظه‌ای شامل بیت روش و کد عمل آنست. سه رقم دیگر آدرس را مشخص می‌کنند. در یک دستورالعمل با آدرس غیرمستقیم بیت روش برابر 1 و اولین رقم شانزده شانزدهی بین 8 تا E می‌باشد. در حالت مستقیم محدوده از 0 تا 6 است. هجده دستور دیگر دارای کد عمل شانزده بیتی هستند. کد هر دستورالعمل بصورت یک عدد چهار رقمی شانزده شانزدهی لیست شده است. اولین رقم یک دستورالعمل ثباتی همواره 7 است. اولین رقم یک دستور ورودی- خروجی همیشه F است. سمبل m در ستون توضیح بمعنی آدرس موثر است. حرف M کلمه‌ای از حافظه را که در آدرس موثر قرار دارد نشان می‌دهد.

۲-۶ زبان ماشین

برنامه مجموعه‌ای از دستورالعمل‌ها برای هدایت کامپیوتر در اجرای یک کار داده پردازی مورد نظر است. زبان‌های مختلفی برای نوشتن یک برنامه کامپیوتری وجود دارد. ولی کامپیوتر می‌تواند تنها برنامه‌هایی را اجرا[2] کند که در داخل آن به شکل دودویی ارائه شده باشند. برنامه‌های نوشته شده به هر زبان دیگر، قبل از اجرا بوسیله کامپیوتر باید به دستورالعمل دودویی ترجمه شوند. برنامه‌های نوشته

1- Mode bit 2- Execute

جدول ۱-۶ دستورالعمل‌های کامپیوتر

سمبل	کد شانزده شانزدهی	شرح
AND	0 یا 8	AND کردن M با AC
ADD	1 یا 9	جمع M با AC، نقلی به E
LDA	2 یا A	بارگردن AC از M
STA	3 یا B	ذخیره AC در M
BUN	4 یا C	انشعاب غیرشرطی به m
BSA	5 یا D	ذخیره آدرس برگشت در m و انشعاب به m+1
ISZ	6 یا E	افزایش M و گذر در صورت صفر بودن نتیجه
CLA	7800	صفر کردن AC
CLE	7400	صفر کردن E
CMA	7200	متمم کردن AC
CME	7100	متمم کردن E
CIR	7080	چرخش به راست E و AC
CIL	7040	چرخش به چپ E و AC
INC	7020	افزایش AC
SPA	7010	گذر در صورت مثبت بودن AC
SNA	7008	گذر در صورت منفی بودن AC
SZA	7004	گذر در صورت صفر بودن AC
SZE	7002	گذر در صورت صفر بودن E
HLT	7001	توقف کامپیوتر
INP	F800	دریافت اطلاعات ورودی و 0 کردن پرچم
OUT	F400	ارسال اطلاعات خروجی و 0 کردن پرچم
SKI	F200	گذر در صورت 1 بودن پرچم ورودی
SKO	F100	گذر در صورت 1 بودن پرچم خروجی
ION	F080	فعال کردن وقفه
IOF	F040	غیرفعال کردن وقفه

شده برای یک کامپیوتر می‌توانند یکی از رسته‌های زیر باشند.

۱- **کد دودویی**. این کد رشته‌ای از دستورالعمل‌ها و عملوندها به شکل دودویی است که شکل واقعی آنها را آنطور که در حافظه کامپیوتر ظاهر می‌شوند، نشان می‌دهد.

۲- **کد هشت هشتی یا شانزده شانزدهی**. این کد معادل ترجمه شده کد دودویی به هشت هشتی یا شانزده شانزدهی است.

۳- **کد سمبلیک**. در این کد، کاربر از سمبل‌ها (حروف، اعداد یا کاراکترهای خاص) برای بخش عملیاتی، بخش آدرس، و سایر قسمت‌های کد دستورالعمل استفاده می‌کند. هر دستورالعمل سمبلیک را می‌توان به یک دستورالعمل کد شده با دودویی ترجمه کرد. این ترجمه توسط برنامه خاصی بنام اسمبلر[1] انجام می‌شود. چون اسمبلر سمبل‌ها را ترجمه می‌کند، این نوع برنامه سمبلیک، برنامه بزبان اسمبلی خوانده می‌شود.

1- Assembler

۴- **زبان‌های برنامه‌نویسی سطح بالا.** این برنامه‌ها که بزبان‌های خاصی نوشته می‌شوند بخاطر دریافت تأثیر رویّه‌هایی است که به منظور حل مسئله خاصی بکار می‌روند و نه صرفاً بخاطر تأثیر بر رفتار سخت‌افزار کامپیوتر. مثالی از یک برنامه بزبان سطح بالا فرترن[1] است. برنامه بصورت رشته‌ای از عبارات بر اساس نحوه تفکر فرد بهنگام حل یک مسئله نوشته می‌شود. با این وجود، هر عبارت باید قبل از اجرا در کامپیوتر به رشته‌ای از دستورات دودویی تبدیل شود. برنامه‌ای که یک برنامه دیگر بزبان سطح بالا را به دودویی ترجمه می‌کند کامپایلر نامیده می‌شود.

دقیقتر بگوئیم، برنامه زبان ماشین یک برنامه دودویی از رسته ۱ است. بعلت شباهت بین نمایش‌های دودویی، هشت هشتی و شانزده شانزدهی، رسم بر این است که رسته ۲ بعنوان زبان ماشین شناخته شود. بدلیل رابطه یک به یک بین دستور سمبلیک و معادل دودویی‌اش، زبان اسمبلی بعنوان زبان در سطح ماشین شناخته می‌شود.

اکنون با بکارگیری کامپیوتر پایه رابطه بین زبان‌های دودویی و اسمبلی را شرح می‌دهیم. برنامه لیست شده در جدول ۲-۶ را ملاحظه کنید. اولین ستون، مکان حافظه (به دودویی) را برای هر دستور یا عملوند می‌دهد. ستون دوم محتوای دودویی این مکان‌هاست. (مکان، آدرس کلمه حافظه‌ای است که دستورالعمل در آن ذخیره شده است. شناخت تفاوت این آدرس، از بخش آدرس دستورالعمل اهمیت دارد) برنامه در بخش مشخصی از حافظه ذخیره شده و سپس بوسیله کامپیوتر و با شروع از آدرس 0اجرا می‌شود. سخت‌افزار کامپیوتر این دستورالعمل‌ها را اجرا کرده و کار مورد نظر را انجام می‌دهد. معهذا، فردی که به این برنامه نگاه می‌کند به سختی درخواهد یافت که پس از اجرای برنامه چه رخ خواهد داد. با این حال، سخت‌افزار کامپیوتر تنها این نوع کدهای دستورالعمل را تشخیص می‌دهد.

نوشتن 16 بیت برای هر دستورالعمل خسته کننده است زیرا رقم‌ها زیادند. ما می‌خواهیم تعداد ارقام برای هر دستورالعمل را با بکاربردن کد هشت هشتی معادل کد دودویی کاهش دهیم. این کار نیاز به شش

جدول ۲-۶ برنامه دودویی برای جمع کردن دو عدد

خانه حافظه	کد دستورالعمل
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

1- Fortran

رقم برای هر دستورالعمل دارد. از طرف دیگر، ما می‌توانیم هر دستورالعمل را به چهار رقم تقلیل دهیم بشرطی که از کد معادل شانزده شانزدهی جدول ۳-۶ استفاده کنیم. نمایش شانزده شانزدهی در کاربرد ساده است؛ اما باید دانست که هر رقم شانزده شانزدهی وقتی که برنامه وارد کامپیوتر می‌شود باید تبدیل به معادل چهاربیتی‌اش شود. مزیت نوشتن برنامه‌های دودویی به معادل هشت هشتی یا شانزده شانزدهی از این مثال باید آشکار شده باشد.

برنامه جدول ۴-۶ از نام‌های سمبلیک دستورالعمل‌ها (این دستورالعمل‌ها در جدول ۱-۶ معرفی شدند) بجای معادل دودویی یا شانزده شانزدهی آنها استفاده کرده است. بخش آدرس دستورات حافظه‌ای، همانند عملوندها، به همان شکل شانزده شانزدهی‌شان باقی می‌مانند. توجه کنید که در مکان 005 مقدار عملوند منفی است زیرا بیت علامت در سمت چپ‌ترین مکان 1 است. در نظر گرفتن ستونی برای توضیحات راهی را برای شرح عملکرد دستورات در اختیار می‌گذارد. کار با برنامه‌های سمبلیک آسانتر است، و در نتیجه، بهتر است برنامه‌ها با سمبل‌ها نوشته شوند. این برنامه‌ها قابل تبدیل به کد دودویی معادل برای تولید برنامه دودویی می‌باشند.

حال یک قدم جلوتر می‌رویم و هر آدرس شانزده شانزدهی را با یک آدرسی سمبلیک و نیز هر عملوند شانزده شانزدهی را با عملوند دهدهی جایگزین می‌کنیم. این اقدام مناسبی است زیرا معمولاً نویسنده برنامه بهنگام برنامه‌نویسی عدد واقعی مکان حافظه را نمی‌داند. اگر عملوند در حافظه بعد از

جدول ۳-۶ برنامه شانزده شانزدهی برای جمع کردن دو عدد

خانه حافظه	دستورالعمل
000	2004
001	1005
002	3006
003	7001
004	0053
005	FFE9
006	0000

جدول ۴-۶ برنامه با کدهای عمل سمبلیک

خانه حافظه	دستورالعمل	توضیحات
000	LDA 004	بار کردن عملوند اول در AC
001	ADD 005	جمع کردن عملوند دوم در AC
002	STA 006	ذخیره مجموع در خانه 006
003	HLT	توقف کامپیوتر
004	0053	عملوند اول
005	FFE9	عملوند دوم
006	0000	محل ذخیره مجموع

دستورالعمل قرار داده شود و اگر طول برنامه از قبل مشخص نباشد، محل عددی عملوندها تا پایان برنامه معلوم نخواهد بود. بعلاوه اعداد دهدهی آشناتر از معادل شانزده شانزدهی‌شان است.

برنامه جدول ۵-۶ یک برنامه به زبان اسمبلی برای جمع دو عدد است. سمبل ORG که بدنبال آن اعدادی آمده است یک دستورالعمل ماشین نیست. هدف از آن تعیین مبدأ، یعنی، تعیین مکانی از حافظه است که دستورالعمل بعدی در زیر آن قرار می‌گیرد. سه خط بعدی دارای آدرس‌های سمبلیک (نمادین) هستند. مقدار هریک از سمبل‌ها از سطری که آن سمبل را بصورت عنوان در ستون اول خود دارد مشخص می‌شود. عملوندهای دهدهی با سمبل DEC مشخص می‌گردند. عددها می‌توانند مثبت یا منفی باشند، اما اگر منفی باشند باید به فرم متمم 2 علامتدار دودویی درآیند. آخرین خط سمبل END را دارد که انتهای برنامه را نشان می‌دهد. سمبل‌های ORG، DEC و END، شبه دستورالعمل[1]ها خوانده می‌شوند و در بخش بعد تعریف شده‌اند. توجه کنید که قبل از تمام توضیحات علامت ممیز (/) آمده است.

برنامه معادل بزبان فرترن برای جمع دو عدد صحیح در جدول ۶-۶ لیست شده است. دو مقدار برای A و B را می‌توان با دستور INPUT یا با دستور DATA مشخص کرد. عمل حسابی روی دو عدد با یک دستور ساده انجام می‌شود. ترجمه این برنامه فرترن به یک برنامه دودویی متشکل از تخصیص سه مکان حافظه به مضاف، مضاف‌الیه و حاصل‌جمع، و سپس بدست آوردن رشته دستورالعمل‌های دودویی که مجموع را تشکیل می‌دهند می‌باشد. بنابراین یک برنامه مترجم سمبل‌های برنامه فرترن را به لیستی از برنامه جدول ۲-۶ که بشکل دودویی است ترجمه می‌کند.

جدول ۵-۶ برنامه به زبان اسمبلی برای جمع دو عدد

```
        ORG 0       / مبداء برنامه خانه 0 است
        LDA A       / بارکردن عملوند از A
        ADD B       / جمع کردن عملوند در B
        STA C       / ذخیره مجموع در C
        HLT         / توقف کامپیوتر
A,      DEC 83      / عملوند دهدهی
B,      DEC -23     / عملوند دهدهی
C,      DEC 0       / مجموع در خانه C ذخیره می‌شود
        END         / پایان برنامه نمادین
```

جدول ۶-۶ برنامه فرترن برای جمع دو عدد

```
INTEGER A, B, C
DATA A,83  B,-23
C = A + B
END
```

1- Preudomstruction

۳-۶ زبان اسمبلی

هر زبان برنامه نویسی با مجموعه‌ای از قواعد تعریف می‌شود. اگر کاربران بخواهند برنامه‌هایشان بطور صحیح ترجمه شود باید این قواعد را رعایت کنند. تقریباً هر کامپیوتر تجاری دارای زبان اسمبلی خاص خود است. قواعد نوشتن برنامه‌ها به زبان اسمبلی بصورت راهنماها مستندسازی و چاپ می‌شود و معمولاً توسط سازندگان کامپیوتر در دسترس قرار می‌گیرد.

واحد پایه یک برنامه اسمبلی یک سطر کد است. یک زبان خاص بر اساس مجموعه قواعد خاصی تعریف می‌شود و این قواعد بیان می‌دارند که سمبل‌ها چگونه می‌توانند استفاده شوند یا با یکدیگر ترکیب شوند تا یک خط از کد را بسازند. حال ما قواعد یک زبان اسمبلی را برای نوشتن یک برنامه سمبلیک برای کامپیوتر پایه فرموله می‌کنیم.

قواعد زبان

هر خط از برنامه زبان اسمبلی به صورت سه ستون مرتب شده‌اند که هر یک میدان[1] نامیده می‌شوند. میدان‌ها اطلاعات زیر را مشخص می‌کنند.

۱- میدان عنوان (برچسب) که می‌تواند خالی باشد یا یک آدرس سمبلیک را مشخص کند.

۲- میدان دستورالعمل که یک دستورالعمل ماشین یا یک شبه دستورالعمل را مشخص می‌نماید.

۳- میدان توضیحات که می‌تواند خالی یا حاوی یک توضیح باشد.

هر آدرس سمبلیک حاوی یک، دو، سه ولی نه بیش از سه کاراکتر الفبا عددی است. اولین کاراکتر حتماً باید یک حرف، دو کاراکتر بعدی حرف یا عدد باشند. سمبل می‌تواند بدلخواه توسط برنامه‌نویس انتخاب شود. یک آدرس سمبلیک در میدان عنوان به یک کاما ختم می‌شود تا اسمبلر تشخیص دهد که عنوان است.

میدان دستورالعمل در یک برنامه زبان اسمبلی یکی از اقلام زیر را مشخص می‌نماید.

۱- دستورالعمل‌های حافظه‌ای[2] MRI

۲- دستورالعمل‌های ثباتی[3] یا ورودی خروجی از نوع غیر ارجاع به حافظه non – MRI .

۳- شبه دستورالعمل‌ها با ، و بدون عملوند

یک دستورالعمل حافظه‌ای دو یا سه سمبل را که با فواصلی از هم جدا شده‌اند استفاده می‌کند. سمبل اول باید سه حرفی باشد که یکی از کدهای عمل MRI جدول ۱-۶ را تعریف می‌نماید. دومین سمبل، آدرس سمبلیک است. سومین سمبل، که ممکن است وجود داشته یا نداشته باشد، حرف I است. اگر I وجود نداشته باشد، سطر، بیانگر آدرس مستقیم است. حضور I بمعنی غیرمستقیم بودن

1- Field 2- Memory– Reference Instructions

3- Register – Reference Instruction

آدرس دستورالعمل است.

دستورالعمل غیرارجاع به حافظه دستوری است که بخش آدرس ندارد. نشانه غیرارجاع به حافظه در میدان دستورالعمل برنامه بوسیله هر یک از سمبل‌های سه حرفی در لیست جدول ۱-۶ برای دستورات ثباتی و ورودی تشخیص داده می‌شود.

مثالهای زیر نمونه‌ایی از سمبل‌هایی است که می‌توان در میدان دستورالعمل یک برنامه قرار داد.

CLA	non−MRI
ADD OPR	MRI با آدرس مستقیم
ADD PRT I	MRI با آدرس غیرمستقیم

سمبل سه حرفی هر سطر باید یکی از دستورالعمل‌های کامپیوتر در جدول ۱-۶ باشد. بدنبال یک دستور، مانند ADD، باید یک آدرس سمبلیک وجود داشته باشد. وجود حرف I اختیاری است.

آدرس سمبلیک در میدان آدرس، مکان حافظه مربوط به عملوند را معین می‌کند. این مکان مجدداً باید در جایی از برنامه بصورت عنوان در ستون اول در جلو دستورالعمل آورده شود. برای اینکه بتوان یک برنامه بزبان اسمبلی را به یک برنامه دودویی ترجمه کرد، ضروری است، هر آدرس سمبلیک که در میدان آدرس ذکر شده است مجدداً در میدان عنوان (برچسب) ظاهر گردد.

شبه دستور، یک دستورالعمل ماشین نیست ولی دستورالعمل برای اسمبلر است تا اطلاعاتی از بعضی فازهای ترجمه را در اختیار آن قرار دهد. چهارشبه دستوری که اسمبلر آنها را می‌شناسد در جدول ۶-۷ ارائه شده‌اند. (برنامه‌های اسمبلی دیگر شبه دستورات متعدد دیگری دارند.) شبه دستور ORG به اسمبلر اطلاع می‌دهد که دستورالعمل یا عملوند سطر بعدی قرار است در مکانی از حافظه که بوسیله عدد بعد از ORG مشخص می‌شود قرار گیرد. امکان استفاده بیش از یکبار ORG در برنامه برای مشخص کردن بیش از یک قطعه از حافظه وجود دارد. سمبل END در انتهای برنامه قرارگرفته است تا اسمبلر را از ختم برنامه مطلع سازد. دو شبه دستورالعمل دیگر پایه[1] عملوند را مشخص کرده و به

جدول ۶-۷ تعریف شبه دستورالعمل‌ها

سمبل	اطلاعاتی که به اسمبلر داده می‌شود
ORG N	عدد شانزده شانزدهی N مکانی از حافظه است که دستورالعمل یا عملوند. واقع در سطر بعدی باید در آنجا قرار داده شود
END	انتهای برنامه سمبلیک
DEC N	عدد دهدهی علامتدار N که باید به دودویی تبدیل گردد
HEX N	عدد شانزده شانزدهی N که باید به دودویی تبدیل گردد

1- Radix

اسمبلر می‌گویند که چگونه اعداد لیست شده را به اعداد دودویی تبدیل کند.

میدان سوم در یک برنامه برای ارائه توضیحات در نظر گرفته شده است. هر خط از کد ممکن است توضیح داشته و یا نداشته باشد، ولی اگر داشت، باید برای اسمبلر بوسیله یک ممیز آغاز شود. توضیحات برای تشریح برنامه مفید بوده و به درک قدم به قدم روش اتخاذ شده بوسیله برنامه کمک می‌کند. چون توضیحات فقط نقش تشریحی را دارند لذا در مرحله ترجمه از آنها چشم‌پوشی می‌شود.

مثال

برنامه جدول ۸-۶ مثالی از یک برنامه زبان اسمبلی است. در اولین سطر، شبه دستور ORG برای تعریف مبدأ برنامه در مکان حافظه $(100)_{16}$ قرار دارد. شش خط بعد دستورالعمل‌های ماشین را تعریف می‌کنند و چهار سطر آخر هم شبه دستورالعمل می‌باشند. سه آدرس سمبلیک بکاررفته و هر یک از آنها در ستون ۱ به صورت عنوان و در ستون ۲ به صورت آدرس یک دستور حافظه‌ای آورده شده‌اند. سه شبه دستور عملوند را مشخص می‌نمایند، و آخرین سطر نیز پایان برنامه را مشخص می‌کند.

وقتی که برنامه به کد دودویی ترجمه شده و توسط کامپیوتر اجرا شود، تفریق بین دو عدد را انجام می‌دهد. عمل تفریق بوسیله جمع مفروق‌منه با متمم 2 مفروق انجام می‌شود. مفروق یک عدد منفی است. این عدد به یک عدد دودویی در نمایش متمم 2 علامت‌دار بدل می‌شود زیرا خواسته‌ایم تا تمام اعداد دودویی بشکل متمم 2 باشند. وقتی که متمم 2 مفروق بدست آمد (با متمم کردن و افزایش دادن AC)، 23- به 23+ تبدیل و اختلاف 106 = 23 + 83 = (متمم 2 عدد 23-) + 83 بدست می‌آید.

ترجمه به دودویی

ترجمه برنامه سمبلیک به دودویی توسط برنامه خاصی بنام اسمبلر صورت می‌گیرد. اگر ابتدا ترجمه را روی کاغذ انجام دهیم کارهای انجام شده بوسیله اسمبلر بهتر درک خواهد شد. ترجمه برنامه

جدول ۸-۶ برنامه زبان اسمبلی برای تفریق دو عدد

	ORG 100	/ مبدأ برنامه مکان 100 است
	LDA SUB	/ بار کردن مفروق در AC
	CMA	/ متمم کردن AC
	INC	/ افزایش AC
	ADD MIN	/ جمع کردن مفروق‌منه با AC
	STA DIF	/ ذخیره تفاضل
	HLT	/ توقف کامپیوتر
MIN,	DEC 83	/ مفروق منه
SUB,	DEC -23	/ مفروق
DIF,	HEX 0	/ محل ذخیره تفاضل
	END	/ پایان برنامه سمبلیک

جدول ۸-۶، به کد دودویی معادل آن را می‌توان با مرور برنامه و جایگزین کردن سمبل‌ها با معادل دودویی کد ماشین آنها انجام داد. با شروع از سطر اول، ابتدا با شبه دستورالعمل ORG مواجه می‌شویم. این دستور می‌گوید که برنامه دودویی را از آدرس 100 شانزده شانزدهی شروع کنیم. در سطر دوم دو سمبل بکار رفته است. این سطر باید یک دستورالعمل حافظه‌ای باشد که هر خانه 100 حافظه قرار دارد. چون حرف I وجود ندارد بیت اول کد دستورالعمل باید 0 باشد. نام سمبلیک دستورالعمل LDA است. با ملاحظه جدول ۱-۶ می‌بینیم که اولین رقم شانزده شانزدهی دستور باید 2 باشد، مقدار دودویی بخش آدرس باید از آدرس SUB بدست آید. ستون عنوان را مرور می‌کنیم و می‌بینیم که این سمبل در سطر ۹ واقع شده است. برای تعیین مقدار شانزده شانزدهی می‌بینیم که سطر ۲ حاوی دستورالعملی برای مکان 100 است و هر یک از سطرهای بعدی یک دستور ماشین یا یک عملوند را برای خانه‌های متوالی حافظه مشخص می‌کنند. با شمارش سطرها، می‌بینیم که عنوان SUB در سطر ۹ مربوط به مکان 107 است. بنابراین آدرس شانزده شانزدهی دستور LDA باید 107 باشد. وقتی که دو بخش دستور مونتاژ شوند، کد شانزده شانزدهی 2107 حاصل می‌گردد. سطرهای دیگری که نمایش دهنده دستورات ماشین هستند به روش مشابهی ترجمه شده و کد شانزده شانزدهی آنها در جدول ۹-۶ لیست شده است.

دو سطر در برنامه سمبلیک که با شبه دستور DEC مشخص شده‌اند عملوندهای دهدهی را مشخص می‌نمایند. سومین سطر صفری را با شبه دستور HEX معین می‌کند (DEC نیز می‌تواند بکار رود). عدد دهدهی 83 به دودویی تبدیل شده و در مکان 106 بفرم شانزده شانزدهی قرار داده شده است. عدد دهدهی 23- یک عدد منفی است و باید بفرم متمم 2 علامت دار دودویی تبدیل شود. معادل شانزده شانزدهی عدد دودویی در مکان 107 قرار دارد. سمبل END انتهای برنامه را خبر می‌دهد و می‌گوید که دیگر سطری برای ترجمه وجود ندارد.

جدول ۹-۶ لیست برنامه ترجمه شده جدول ۸-۶

کد شانزده شانزدهی		برنامه سمبلیک	
عمل	محتوا		
			ORG 100
100	2107		LDA SUB
101	7200		CMA
102	7020		INC
103	1106		ADD MIN
104	3108		STA DIF
105	7001		HLT
106	0053	MIN,	DEC 83
107	FFE9	SUB,	DEC -23
108	0000	DIF,	HEX 0
			END

اگر ما دو مرور کامل بر روی برنامه سمبلیک داشته باشیم عمل ترجمه ساده می‌شود. در مرور اول ترجمه‌ای صورت نمی‌گیرد بلکه فقط مکانی برای هر دستورالعمل ماشین و عملوند در حافظه تعلق می‌گیرد. تخصیص مکان مقدار آدرس را برای عناوین تعریف کرده و برای فرآیند ترجمه که در مرور دوم صورت می‌گیرد تسهیلات لازم را فراهم می‌آورد. بنابراین ما در جدول ۹-۶ مکان 100 را به اولین دستور بعد ORG اختصاص می‌دهیم. سپس مکان‌های متوالی بعدی را تا انتهای برنامه برای هریک از سطرهای کد که حاوی دستور یا عملوند هستند، اختصاص می‌دهیم. (به ORG و END عددی اختصاص نمی‌یابد زیرا دستور و یا عملوند نیستند). وقتی که اولین مرور برنامه تکمیل شد، برای هر عنوان شماره مکان آن را مشخص می‌کنیم و جدولی را تشکیل می‌دهیم که مقدار شانزده شانزدهی هر آدرس سمبلیک را نشان می‌دهد. برای این برنامه جدول سمبل آدرس برابر زیر است:

سمبل آدرس	آدرس شانزده شانزدهی
MIN	106
SUB	107
DIF	108

در دومین مرور به برنامه سمبلیک، با مراجعه به جدول سمبل آدرس‌ها مقدار آدرس هر دستورالعمل حافظه‌ای را تعیین می‌کنیم. مثلاً سطر کد LDA SUB در مرور دوم با برداشتن مقادیر شانزده شانزدهی LDA از جدول ۱-۶ و SUB از لیست جدول سمبل آدرس ترجمه می‌شود. سپس دو بخش را بصورت یک دستورالعمل در مبنای شانزده کنار هم[1] می‌گذاریم. اگر بخواهیم نحوه قرار گرفتن این برنامه را در کامپیوتر بدانیم می‌توانیم به آسانی کد شانزده شانزدهی حاصل را به دودویی تبدیل کنیم.

وقتی که ترجمه از سمبل‌ها به دودویی بوسیله یک برنامه اسمبلر انجام شد، اولین مرور، اولین گذر خوانده شده و دومین مرور نیز دومین گذر نامیده می‌شود.

۶-۴ اسمبلر

اسمبلر برنامه‌ای است که یک برنامه بزبان اسمبلی را دریافت و معادل زبان ماشین دودویی آن را تولید می‌کند. برنامه سمبلیک ورودی برنامه منبع[2] و برنامه دودویی حاصل برنامه مقصد[3] خوانده می‌شود. در واقع برنامه اسمبلر روی رشته‌های حرفی عمل کرده و معادل دودویی آنها را تولید می‌نماید.

1- Assemble 2- Source Program 3- Object Program

نمایش برنامه سمبلیک در حافظه

قبل از آغاز اسمبل کردن (مونتاژ کردن)، برنامه سمبلیک باید در حافظه ذخیره شود. کاربر برنامه سمبلیک را از یک پایانه تایپ می‌کند. با استفاده از یک برنامه بارکننده، کاراکترهای برنامه سمبلیک را در حافظه وارد می‌نماید. چون برنامه از مجموعه‌ای از سمبل‌ها ساخته شده است، نمایش آن در حافظه با استفاده از کد کاراکترهای الفباء عددی است. در کامپیوتر پایه، هر کاراکتر با یک کد هشت بیتی نمایش داده می‌شود. همواره با ارزشترین بیت 0 است و هفت بیت دیگر بوسیله ASCII مشخص می‌شود. معادل شانزده شانزدهی مجموعه کاراکترها در جدول ۱۰-۶ آورده شده است. به هر کاراکتر دو رقم مبنای شانزده تخصیص یافته است که بسادگی قابل تبدیل به کد معادل هشت بیتی‌اش می‌باشد. آخرین وارده در جدول، کاراکتری را تولید نمی‌کند بلکه مربوط به حرکت فیزیکی مکان‌نما[1] بر روی صفحه پایانه است. کد مربوطه به CR هنگامی تولید می‌شود که کلید Return فشار داده شود. این باعث بازگشت "نورد"[2] به مکان اولیه‌اش برای شروع سطر جدید می‌گردد. اسمبلر کد CR را بعنوان کد انتهای سطر می‌شناسد.

هر سطر کد در خانه‌های متوالی حافظه بصورت دو کاراکتر در هر مکان ذخیره می‌شود. در هر کلمه دو کاراکتر را می‌توان ذخیره کرد زیرا هر کلمه حافظه دارای 16 بیت ظرفیت است. هر سمبل عنوان به یک کاما ختم می‌شود. سمبل‌های عملیات و آدرس با یک فاصله[3] و پایان هر سطر را کد CR مشخص می‌کند. مثلاً سطرکد

```
PL3,    LDA  SUB  I
```

جدول ۱۰-۶ کد شانزده شانزدهی کاراکترها

کاراکتر	کد	کاراکتر	کد	کاراکتر	کد	
A	41	Q	51	6	36	
B	42	R	52	7	37	
C	43	S	53	8	38	
D	44	T	54	9	39	
E	45	U	55	space	20	
F	46	V	56	(	28	
G	47	W	57	)	29	
H	48	X	58	*	2A	
I	49	Y	59	+	2B	
J	4A	Z	5A	,	2C	
K	4B	0	30	-	2D	
L	4C	1	31	.	2E	
M	4D	2	32	/	2F	
N	4E	3	33	=	3D	
O	4F	4	34	CR	0D	برگشت نورد
P	50	5	35			

1- Cursor 2- Carriage 3- Space

در هفت کلمه حافظه متوالی، مطابق جدول ۱۱-۶، ذخیره شده است. عنوان PL3 دو کلمه را اشغال می‌کند و به یک کاما (2C) منتهی می‌شود. میدان دستورالعمل در سطر کد ممکن است دارای یک یا چند سمبل باشد. هر سمبل به کد مربوط به فاصله (20) ختم می‌شود، بجز آخرین سمبل که به کد برگشت نورد (0D) ختم می‌گردد. اگر سطر کد دارای توضیحات باشد، اسمبلر آن را بوسیله ممیز یعنی 2F شناسایی می‌کند. اسمبلر کلیه کاراکترهای میدان توضیحات را چشم‌پوشی کرده و بدنبال کد CR می‌گردد. پس از اینکه با این کد مواجه شد، به جای آن کد فاصله را پس از آخرین سمبل سطر قرار می‌دهد. ورودی برنامه اسمبلر برنامه بزبان سمبلیک در ASCII است. این ورودی بوسیله اسمبلر دوبار مرور می‌شود تا برنامه معادل دودویی آن تولید شود. برنامه دودویی، خروجی تولید شده توسط اسمبلر می‌باشد. اکنون ما بطور مختصر عمده کارهای اسمبلر را حین ترجمه تشریح می‌کنیم.

مرور اول

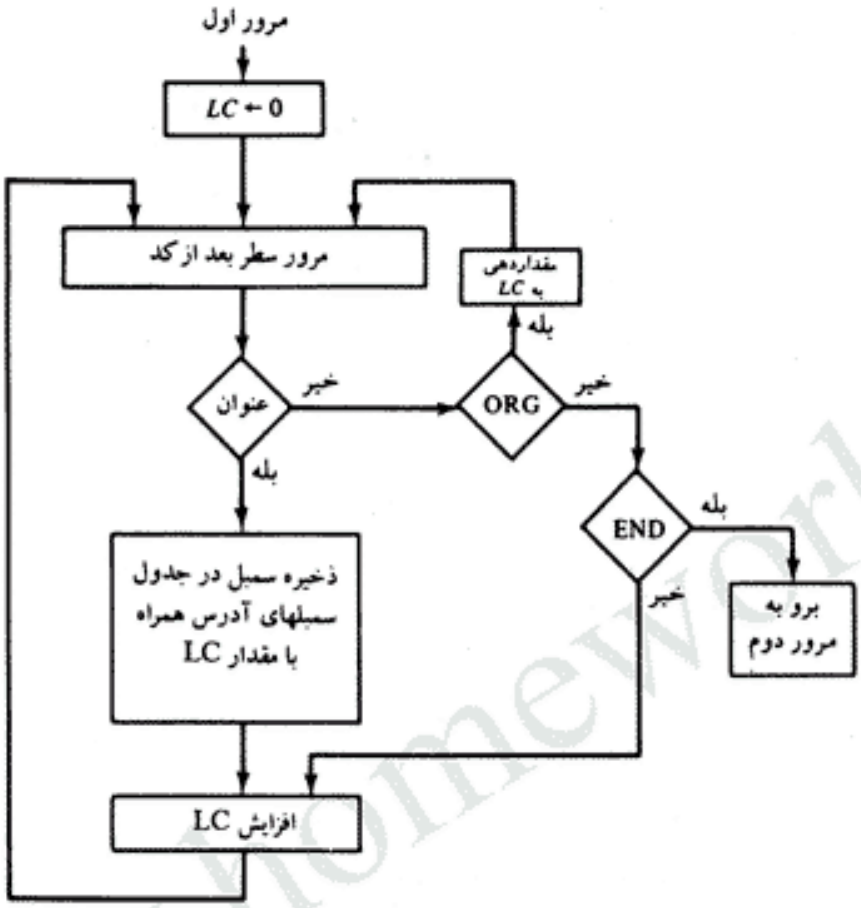
یک اسمبلر دو مروری سراسر برنامه سمبلیک را دوبار مرور می‌کند. در مرور اول، جدولی را تولید می‌کند که مقدار معادل دودویی همه سمبل‌های آدرس تعریف شده توسط کاربر را نشان می‌دهد. ترجمه به دودویی در مرور دوم صورت می‌گیرد. برای دنبال کردن محل دستورالعمل‌ها، اسمبلر از کلمه‌ای در حافظه به نام مکان‌شمار[1] (بطور خلاصه LC) استفاده می‌کند. محتوای LC ، آدرس محلی از حافظه را که به دستورالعمل یا عملوند در دست پردازش اختصاص داده می‌شود، مشخص می‌کند. شبه دستور ORG آدرس اولین مکان حافظه در نظر گرفته شده برای برنامه را به عنوان مقدار اولیه به مکان شمار می‌دهد. چون دستورالعمل‌ها متوالیاً ذخیره می‌شوند، پس از پردازش هر سطر کد، LC افزایش می‌یابد. برای جلوگیری از هرگونه ابهام در غیاب ORG ، اسمبلر در آغاز مقدار 0 را به LC می‌دهد.

کارهایی که توسط اسمبلر در اولین مرور صورت می‌گیرد در فلوچارت شکل ۱-۶ توصیف شده

جدول ۱۱-۶ نمایش سطر کد PL3, LDA SUB I در کامپیوتر

کلمه حافظه	سمبل	کد شانزده شانزدهی	نمایش دودویی
1	P L	50 4C	0101 0000 0100 1100
2	3 ,	33 2C	0011 0011 0010 1100
3	L D	4C 44	0100 1100 0100 0100
4	A	41 20	0100 0001 0010 0000
5	S U	53 55	0101 0011 0101 0101
6	B	42 20	0100 0010 0010 0000
7	I CR	49 0D	0100 1001 0000 1101

1- Loation Counter



شکل ۱-۶ فلوچارت مرور اول اسمبلر

است. LC ابتدا 0 می‌شود. خط اول کد سمبلیک تحلیل می‌شود تا معین شود آیا دارای عنوان است یا خیر (این کار با وجود کاما مشخص می‌گردد). اگر سطر فاقد عنوان باشد، اسمبلر سمبل را در میدان دستورالعمل چک می‌کند. اگر این میدان حاوی شبه دستور ORG باشد، اسمبلر LC را با مقداری که در جلوی ORG آمده است بار می‌کند و برای پردازش سطر بعدی باز می‌گردد. اگر سطر دارای شبه دستور END باشد، اسمبلر مرور اول را خاتمه می‌دهد و به سراغ مرور دوم می‌رود (دقت کنید سطر ORG و END دارای عنوان نیست). اگر سطر موردنظر دارای عنوان باشد، این عنوان همراه با معادل دودویی‌اش که در LC قرار دارد در جدول سمبل آدرس قرار داده می‌شود. اگر عنوان وجود نداشته باشد چیزی در جدول ذخیره نمی‌شود. سپس LC یک واحد افزایش یافته و سطر جدیدی پردازش می‌گردد.

برای برنامه جدول ۸-۶، اسمبلر جدول سمبل آدرس در جدول ۱۲-۶ را تولید می‌کند. هر سمبل عنوان در دو مکان حافظه ذخیره می‌شود و بوسیله یک کاما خاتمه می‌یابد. اگر عنوان کمتر از سه کاراکتر داشته باشد، مکان‌های حافظه اضافی با کد فاصله پر می‌شود. مقداری که در LC در حین پردازش سطر وجود دارد در حافظه بعدی ذخیره می‌شود. این برنامه سه آدرس سمبلیک دارد: MIN ، SUB و DIF .

جدول ۱۲-۶ جدول سمبل‌های آدرس برای برنامه جدول ۸-۶

کلمه حافظه	سمبل یا (LC)*	کد شانزده شانزدهی	نمایش دودویی
1	M I	4D 49	0100 1101 0100 1001
2	N ,	4E 2C	0100 1110 0010 1100
3	(LC)	01 06	0000 0001 0000 0110
4	S U	53 55	0101 0011 0101 0101
5	B ,	42 2C	0100 0010 0010 1100
6	(LC)	01 07	0000 0001 0000 0111
7	D I	44 49	0100 0100 0100 1001
8	F ,	46 2C	0100 0110 0010 1100
9	(LC)	01 08	0000 0001 0000 1000

*(LC) محتوای شمارنده مکان را مشخص می‌کند

این سمبل‌ها بترتیب آدرس‌های 12 بیتی را که معادل 106 ، 107 و 108 می‌باشند نمایش می‌دهند. جدول سمبل آدرس سه کلمه حافظه فوق را برای هر سمبل عنوان که با آن مواجه شود اشغال می‌کند و داده‌های خروجی را که اسمبلر در حین مرور اول ایجاد می‌کند تشکیل می‌دهد.

مرور دوم

دستورات ماشین در حین مرور دوم با استفاده از روش‌های نظاره به جدول[1] ترجمه می‌شوند. روش نظاره به جدول جستجویی است بر واردهای جدول برای اینکه تعیین کنیم آیا یک نمونه مورد نظر با نمونه‌های ذخیره شده در جدول مطابقت دارد یا خیر. اسمبلر چهار جدول را بکار می‌برد. هر سمبلی که در برنامه با آن مواجه شود باید بعنوان یکی از واردها در این جدول موجود باشد؛ در غیر اینصورت سمبل قابل تفسیر نیست. ما نام‌های زیر را به چهار جدول اختصاص می‌دهیم:

۱- جدول شبه دستورالعمل‌ها

۲- جدول MRI

۳- جدول non-MRI

۴- جدول سمبل آدرس

واردهای جدول شبه دستورالعمل‌ها چهار سمبل دارد: ORG ، END ، DEC و HEX . هر وارده هنگام برخورد با آن در برنامه، اسمبلر را به زیرروال شبه دستور مربوطه، هدایت می‌کند. جدول MRI حاوی هفت سمبل دستورالعمل‌های حافظه‌ای و کد عمل معادل سه بیتی آنهاست. جدول non-MRI حاوی هجده دستورالعمل ثباتی و ورودی- خروجی همراه با شانزده کد دودویی معادل آنها می‌باشد.

1- Table look up

جدول سمبل آدرس در مرور اول پردازش اسمبلی تولید می‌گردد. اسمبلر این جداول را برای یافتن سمبلی که در حال پردازش است جستجو می‌کند تا مقدار دودویی آن را معین کند.

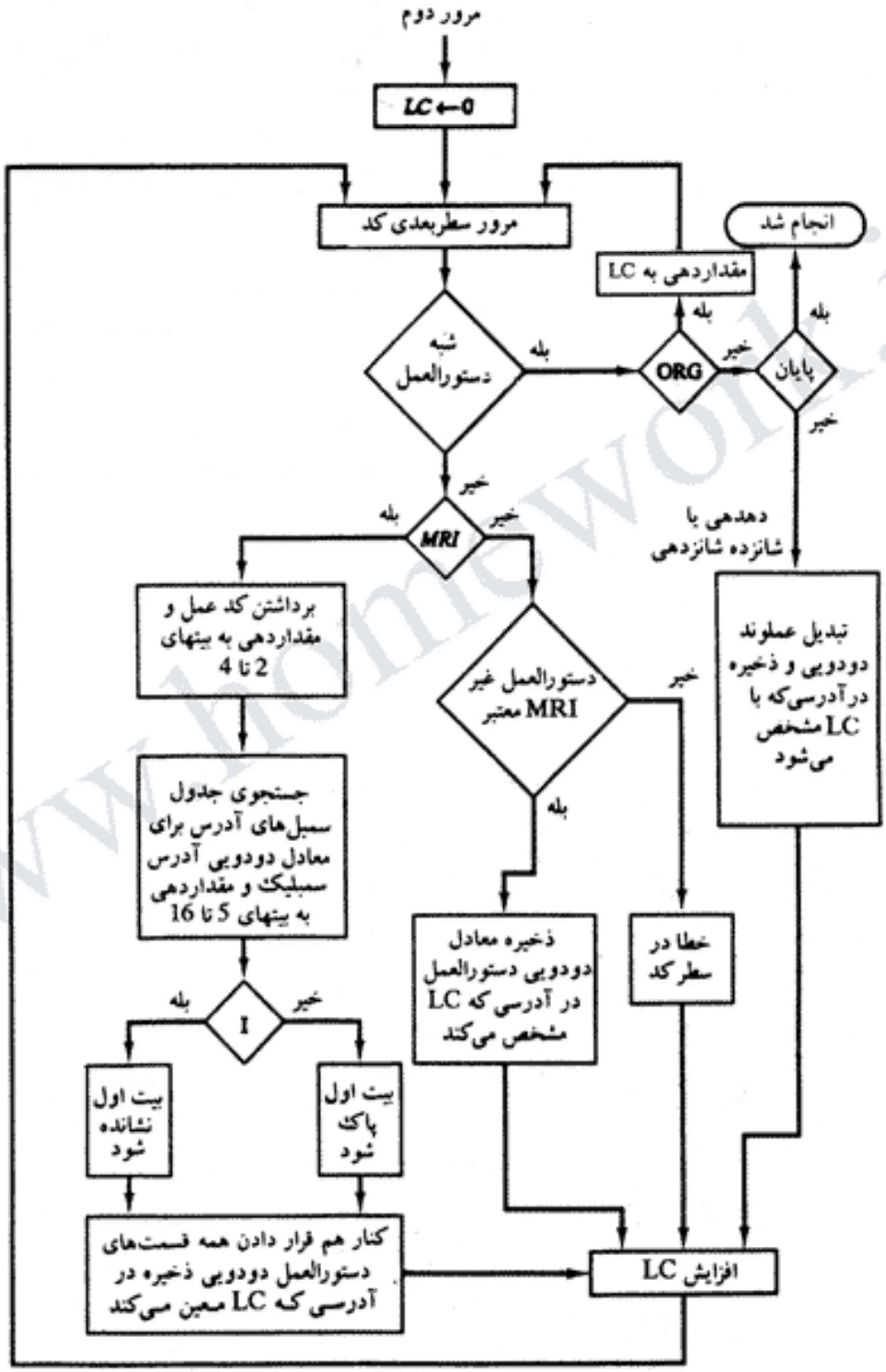
کاری که در حین دومین مرور توسط اسمبلر انجام می‌شود در شکل ۲-۶ آمده است. LC در آغاز با 0 بار می‌شود. سپس سطرهای کد یک به یک تحلیل می‌شوند. عناوین در مرور دوم چشم‌پوشی می‌شوند، لذا اسمبلر بلافاصله به میدان آدرس رفته و اولین سمبلی راکه با آن مواجه می‌شود وارسی می‌کند. ابتدا جدول شبه دستور را چک می‌نماید. تطابق با ORG ، اسمبلر را به زیرروالی می‌فرستد که در آن LC با یک مقدار اولیه بار می‌شود. هر تطابق با END ، فرآیند ترجمه را خاتمه می‌دهد. شبه دستورات عملوندی موجب تبدیل عملوند به دودویی‌شان می‌گردند. این عملوند در خانه‌ای از حافظه که بوسیله LC مشخص می‌شود قرار می‌گیرد. سپس شمارنده مکان، LC ، یک واحد افزایش می‌یابد و اسمبلر به تحلیل کد سطر بعدی ادامه می‌دهد.

اگر سمبلی که با آن برخورد شده است شبه دستور نباشد، اسمبلر به جدول MRI مراجعه می‌کند. اگر سمبل در این جدول پیدا نشد، اسمبلر به جدول non-MRI می‌رود. سمبلی که در این جدول پیدا شود متعلق به دستورالعمل‌های ثباتی یا ورودی- خروجی است. اسمبلر کد شانزده بیتی دستورالعمل را در کلمه حافظه‌ای که بوسیله LC مشخص می‌گردد ذخیره می‌کند. مکان‌شمار، LC ، افزایش یافته و تحلیل ادامه می‌یابد.

هنگامی که سمبلی در جدول MRI پیدا شود، اسمبلر کد سه بیتی آن را استخراج کرده و آن را در بیت‌های 2 الی 4 از یک کلمه قرار می‌دهد. یک دستورالعمل حافظه‌ای با دو یا سه سمبل مشخص می‌شود. سمبل دوم آدرس سمبلیک است و سومی، که ممکن است وجود داشته باشد و یا نداشته باشد حرف I است. آدرس سمبلیک با جستجوی جدول سمبل آدرس به دودویی تبدیل می‌گردد. بیت اول دستور، بسته به اینکه حرف I وجود داشته باشد یا خیر، 1 یا 0 است. سه جزء کد دستورالعمل در کنار هم گذاشته شده[1] (یا به بیان دیگر مونتاژ شده) و سپس در مکانی از حافظه که توسط محتوای LC مشخص می‌گردد، ذخیره می‌شود. مکان‌شمار افزایش یافته و اسمبلر به پردازش سطر بعدی ادامه می‌دهد.

عمل مهمی که یک اسمبلر می‌باید انجام دهد یافتن خطاهای ممکن در برنامه سمبلیک است. این کار تشخیص خطا[2] خوانده می‌شود. چنین خطایی ممکن است سمبل نامعتبر کد ماشین باشد که نشانه آن عدم وجودش در جدول‌های MRI و non-MRI است. اسمبلر قادر به ترجمه چنین سمبلی نیست زیرا مقدار معادل دودویی آن را نمی‌داند. در چنین حالتی، اسمبلر پیام خطایی را چاپ می‌کند تا برنامه‌نویس را از وجود خطا در سطر خاصی از کد برنامه سمبلیک مطلع سازد. خطای ممکن دیگر این است که برنامه دارای آدرس سمبلیکی باشد که در جدول وجود ندارد اسمبلر قادر نخواهد بود این سطر کد را بطور صحیحی ترجمه کند زیرا معادل دودویی آن سمبل را در جدول سمبل آدرس که در مرور اول تولید شده نمی‌یابد. خطاهای دیگری هم ممکن است رخ دهند و یک اسمبلر قابل استفاده در عمل باید

1- Assembled 2- Diagnostic



شکل ۲-۶ فلوچارت مرور دوم اسمبلر

همه این گونه خطاها را کشف و برای هر یک پیغام خطایی را چاپ کند.

باید تأکید کرد که یک اسمبلر عملی بسیار پیچیده‌تر از آنچه در اینجا شرح داده‌ایم می‌باشد. اکثر کامپیوترها انعطاف زیادی را در نوشتن برنامه‌ها بزبان اسمبلی در اختیار برنامه‌نویس می‌گذارند. مثلاً ممکن است به کاربر اجازه داد تا برای مشخص‌کردن آدرس از عدد یا یک سمبل استفاده کند. بسیاری از اسمبلرها این امکان را به کاربر می‌دهند که آدرس را با یک عبارت ریاضی مشخص کند. شبه‌دستورالعمل‌های بسیار زیادی را نیز ممکن است برای تسهیل در کار برنامه‌نویس مشخص کرد. با کاراتر شدن و تکمیل زبان اسمبلی، اسمبلر پیچیده‌تر می‌شود.

۶-۵ حلقه در برنامه‌نویسی

حلقه برنامه رشته‌ای از دستورات است که چندین بار، و هر بار با مجموعه متفاوتی از داده‌ها، اجرا می‌شود. حلقه برنامه در فرترن با عبارت DO مشخص می‌شود. مثال زیر برنامه‌ای بزبان فرترن است که مجموع 100 عدد صحیح را تهیه می‌کند.

```
      DIMENSION A(100)
      INTEGER SUM, A
      SUM = 0
      DO 3   J = 1 , 100
    3 SUM = SUM + A (J)
```

عبارت شماره 3 ، بتعداد 100 بار تکرار می‌شود و هر بار عملوند متفاوت بوده و مقادیر A(J) که در آن J = 0, 1, ..., 100 می‌باشد بکار می‌رود.

برنامه سیستمی که یک برنامه بزبان سطح بالا را، مانند آنچه در بالا دیدیم به زبان ماشین ترجمه کند، کامپایلر[1] نام دارد. کامپایلر در مقایسه با اسمبلر، برنامه پیچیده‌تری است و درک کامل عملکرد آن مستلزم دانش برنامه‌نویسی سیستم است. معهذا، می‌توانیم کارهای اصلی یک کامپایلر را با دنبال کردن فرآیند ترجمه یک برنامه ورای آنچه که در برنامه زبان اسمبلی بود نشان دهیم. یک کامپایلر می‌تواند از یک برنامه اسمبلی بعنوان مرحله میانی برای ترجمه استفاده کند و یا مستقیماً آنرا به دودویی تبدیل نماید.

اولین عبارت در برنامه فرترن یک دستور DIMENSION است. این عبارت به کامپایلر فرمان می‌دهد تا 100 کلمه حافظه را برای 100 عملوند رزرو نماید. مقدار عملوندها از یک عبارت ورودی[2] بدست می‌آیند (دستور مزبور در برنامه بالا آورده نشده است). عبارت دوم به کامپایلر خبر می‌دهد که

1- Compiler 2- INPUT

اعداد صحیح هستند. اگر اعداد از نوع غیرصحیح[1] باشند، کامپایلر باید مکان‌هایی را برای اعداد ممیز شناور رزرو نموده و دستوراتی را تولید کند که عملیات محاسباتی بعدی را با داده‌های ممیز شناور انجام دهند. این دو دستور همانند شبه دستورالعمل‌ها در اسمبلی، غیرقابل اجرا هستند. فرض کنید کامپایلر مکان‌های $(150)_{16}$ تا $(1B3)_{16}$ را برای 100 عملوند رزرو نماید. این کلمات حافظه رزرو شده در سطرهای 19 الی 118 برنامه ترجمه شده جدول ۱۳-۶ لیست شده‌اند. این کار توسط شبه دستور ORG در سطر 18 انجام می‌شود که ابتدای عملوندها را مشخص می‌کند. اولین و آخرین عملوندها توسط عدد دهدهی خاصی نشان داده شده‌اند، اگرچه این مقادیر ضمن ترجمه مشخص نیستند. کامپایلر فقط فضای داده‌ها را در حافظه رزرو کرده و مقادیر بعداً پس از اجرای عبارت ورودی INPUT در آنها جای می‌گیرد. شماره سطرها در برنامه سمبلیک فقط به منظور ارجاع می‌باشند و نه بخشی از برنامه ترجمه شده آن.

اندیس موجود در عبارت DO بصورت دستورالعمل‌های سطرهای 2 الی 5 و ثابت‌های سطرهای

جدول ۱۳-۶ برنامه سمبلیک برای جمع 100 عدد

سطر			
1		ORG 100	مبداء برنامه 100 شانزده شانزدهی /
2		LDA ADS	بار کردن آدرس ابتدای عملوندها /
3		STA PTR	ذخیره در اشاره گر /
4		LDA NBR	بارکردن 100 منفی /
5		STA CTR	ذخیره در شمارنده /
6		CLA	پاک کردن انباشتگر /
7	LOP,	ADD PTR I	جمع یک عملوند با AC /
8		ISZ PTR	افزایش اشاره گر /
9		ISZ CTR	افزایش شمارنده /
10		BUN LOP	تکرار مجدد حلقه /
11		STA SUM	ذخیره مجموع /
12		HLT	توقف /
13	ADS,	HEX 150	آدرس ابتدای عملوندها /
14	PTR,	HEX 0	محل نگهداری شده برای اشاره گر /
15	NBR,	DEC −100	ثابتی که بعنوان مقدار اولیه به اشاره گر داده می‌شود /
16	CTR,	HEX 0	محل ذخیره شده برای شمارنده /
17	SUM,	HEX 0	محل ذخیره مجموع /
18		ORG 150	مبداء عملوندها ۱۵۰ در مبنای شانزده شانزدهی است /
19		DEC 75	نخستین عملوند /
•			
•			
•			
118		DEC 23	آخرین عملوند /
119		END	انتهای برنامه نمادین /

1- Real

13 تا 16 ترجمه می‌شود. آدرس اولین عملوند (150) در مکان ADS در سطر 13 ذخیره می‌شود. تعداد دفعاتی که دستور شماره 3 برنامه فرترن اجرا می‌شود 100 است. بنابراین عدد 100- در مکان NBR ذخیره می‌گردد. سپس کامپایلر دستورات سطرهای 2 الی 5 را برای ایجاد مقادیر اولیه حلقه برنامه تولید می‌نماید. آدرس اولین عملوند به مکان PTR منتقل می‌گردد. این کار با توجه به A(1) حاصل از A(J)صورت می‌گیرد. سپس عدد 100- به مکان CTR منتقل می‌گردد. این مکان بعنوان شمارنده عمل کرده و هر بار که حلقه برنامه اجرا شود یک واحد افزایش می‌یابد. وقتی که شمارنده به صفر برسد، عمل مورد نظر یکصد بار انجام شده و برنامه از حلقه خارج می‌گردد.

برخی از کامپایلرها دستور SUM = 0 را به یک دستورالعمل ماشین که مکان SUM را بعنوان مقدار اولیه صفر مینماید ترجمه می‌کنند. سپس ارجاع به این مکان هر بار که سطر 3 برنامه فرترن اجرا شود صورت می‌گیرد. کامپایلر هوشمندتر می‌تواند چنین تشخیص دهد که جمع در انباره[1] صورت گرفته و فقط نتیجه نهایی در SUM قرار گیرد. کامپایلر دستورالعملی در سطر 6 تولید خواهد کرد تا انباره AC را پاک کند. همچنین مکانی از حافظه که با SUM مشخص شده است (در خط 17) برای ذخیره مقدار این متغیر در انتهای حلقه نیز بوسیله کامپایلر رزرو می‌شود.

حلقه مشخص شده با عبارت DO به رشته دستورالعمل‌های واقع در سطرهای 7 تا 10 ترجمه می‌شود. سطر 7 دستورالعمل ADD غیر مستقیم است زیرا دارای سمبل I می‌باشد. آدرس عملوند جاری در خانه PTR ذخیره می‌شود. وقتی این مکان بطور غیرمستقیم آدرس‌دهی شود کامپیوتر محتوای PTR را بعنوان آدرس عملوند تلقی می‌نماید. در نتیجه، عملوند در مکان 150 با انباره جمع می‌شود. سپس PTR با دستورالعمل ISZ در سطر 8 یک واحد افزایش می‌یابد، و مقدار آن برابر آدرس عملوند بعدی می‌گردد. مکان CTR در سطر 9 افزایش می‌یابد و اگر صفر نباشد، کامپیوتر از دستورالعمل بعدی گذر نمی‌کند. دستور بعدی یک دستورالعمل انشعاب (BUN) به ابتدای حلقه است، لذا کامپیوتر برای تکرار مجدد حلقه باز می‌گردد. وقتی مکان CTR صفر شود (پس از 100 بار تکرار حلقه) دستور بعدی گذر[2] (صرفنظر) می‌شود و کامپیوتر دستورالعمل سطر 11 و 12 را اجرا می‌نماید. جمع حاصل در انباره در SUM ذخیره شده و کامپیوتر متوقف می‌شود. دستورالعمل توقف[3] برای وضوح برنامه در اینجا قرار داده شده است؛ در واقع برنامه به محلی دیگر برای ادامه اجرای برنامه منتقل می‌گردد و یا به آغاز برنامه دیگری انشعاب می‌نماید. توجه کنید که ISZ در سطر 8 صرفاً برای افزایش اشاره‌گر آدرس PTR بکار رفته است. چون آدرس عددی مثبت است هرگز گذری رخ نمی‌دهد.

برنامه جدول ۱۳-۶ برای نخستین بار مفاهیم اشاره‌گر، شمارنده همراه با عمل آدرس‌دهی غیرمستقیم را بکار برده است تا یک حلقه را در برنامه بوجود آورد. اشاره‌گر به آدرس عملوند جاری اشاره می‌نماید و شمارنده تعداد دفعات اجرای حلقه برنامه را می‌شمارد. در این مثال ما دو حافظه را برای این مقاصد استفاده کردیم. در کامپیوترهایی که بیش از یک ثبات پردازنده دارند می‌توان از یک ثبات بعنوان

1- Accumulator 2- Skip 3- HALT

اشاره‌گر، از دیگری بعنوان شمارنده و از سومی بجای انباره استفاده کرد. وقتی که از ثبات‌های پردازنده بعنوان اشاره‌گر و شمارنده استفاده شود به آنها ثبات‌های شاخص[1] گفته می‌شود. ثبات‌های شاخص در بخش ۵-۸ مورد بحث قرار گرفته‌اند..

۶-۶ برنامه‌نویسی اعمال حسابی و منطقی

تعداد دستورالعمل‌های موجود در یک کامپیوتر از چند صد دستور در سیستم‌های بزرگ تا چند دوجین در کامپیوترهای کوچک متغیر است. برخی کامپیوترها عمل مفروضی را با یک دستورالعمل انجام می‌دهند و برخی دیگر برای انجام همان عمل ممکن است تعداد زیادی دستورات ماشین نیاز داشته باشند. مثلاً چهار عمل اصلی حساب را در نظر بگیرید. بعضی کامپیوترها دارای دستورات ماشین برای جمع، تفریق، ضرب و تقسیم هستند. برخی دیگر، مانند کامپیوتر پایه، فقط دارای یک دستور حسابی، ADD، است. اعمالی که در مجموعه دستورالعمل‌های ماشین گنجانده نشده باشد باید بوسیله یک برنامه پیاده سازی شود. ما در جدول ۸-۶ برنامه‌ای را برای تفریق دو عدد دیدیم. برنامه‌های مربوط به سایر عملیات حسابی به روشی مشابه بدست می‌آید.

اعمالی که در یک کامپیوتر توسط یک دستورالعمل ماشین پیاده سازی شده باشند گوئیم بطور سخت‌افزاری پیاده‌سازی شده‌اند. به اعمالی که بوسیله مجموعه‌ای از دستورالعمل‌ها پیاده سازی می‌شوند گوییم که بطور نرم‌افزاری پیاده‌سازی شده‌اند. برخی از کامپیوترها مجموعه گسترده‌ای از دستورات سخت‌افزاری را دارا هستند که برای بالا بردن سرعت انجام کارهای متداول طراحی شده‌اند. دسته‌ای دیگر دارای دستورالعمل‌های سخت‌افزاری کمتری هستند و بر پیاده‌سازی نرم‌افزاری اعمال متکی‌اند. پیاده سازی سخت‌افزاری بسیار پرهزینه است زیرا مدارهای اضافه‌تری برای پیاده‌سازی اعمال نیاز دارد. پیاده‌سازی نرم‌افزاری موجب ایجاد برنامه‌های طولانی هم از نظر تعداد دستورات و هم از نظر زمان اجرا می‌شود.

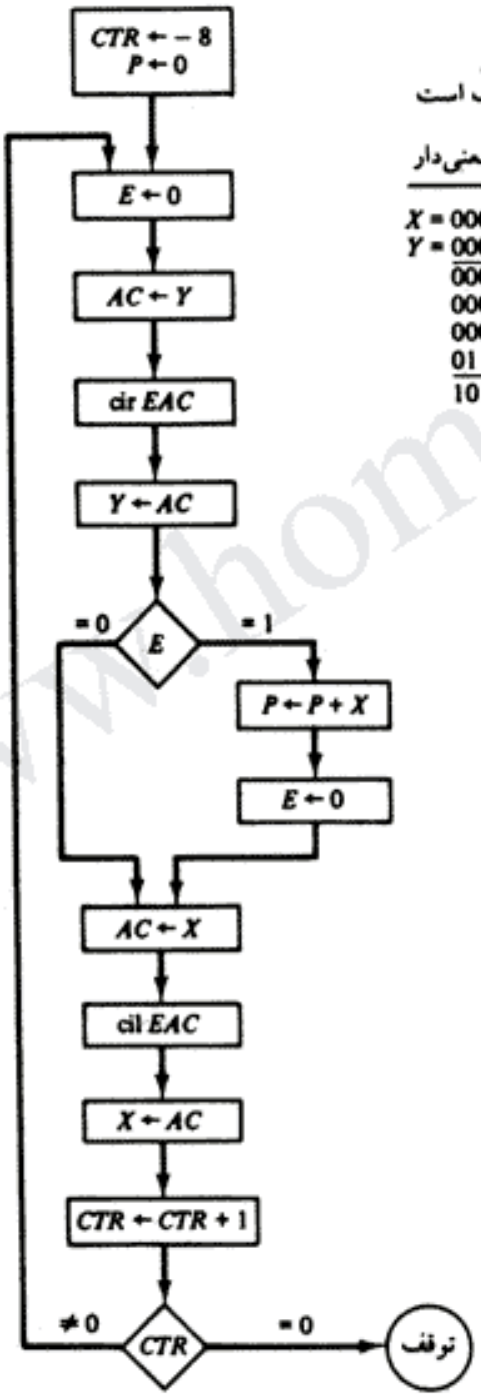
این بخش پیاده سازی نرم‌افزاری چند عمل حسابی و منطقی را نشان می‌دهد. برنامه‌ها برای هر نوع عمل حسابی نه فقط برای داده‌های ممیز ثابت بلکه برای داده‌های دهدهی و ممیز شناور نوشته شده‌اند. پیاده‌سازی سخت‌افزاری عملیات حسابی در فصل ۱۰ آورده شده است.

برنامه ضرب

اینک برنامه‌ای را برای ضرب دو عدد می‌نویسیم. برای ساده کردن برنامه از بیت علامت صرفنظر کرده و فرض می‌کنیم اعداد مثبت باشند. همچنین فرض می‌کنیم که دو عدد دودویی بیش از هشت بیت ندارند، لذا حاصلضرب آنها از 16 بیت تجاوز نمی‌کند. می‌توان برنامه را اصلاح کرد تا علامت اعداد نیز در نظر گرفته شود و یا از اعداد 16 بیتی استفاده گردد. در این صورت حاصلضرب ممکن است تا 31 بیت طول داشته و دو کلمه از حافظه را اشغال نماید.

1- Index Register

برنامه ضرب دو عدد مبتنی بر روش ضرب اعداد با قلم و کاغذ است. همانطور که در مثال عددی شکل ۳-۶ نشان داده شده است، فرآیند ضرب از وارسی بیت‌های مضروب‌فیه Y و جمع مضروب X بتعداد 1 های موجود در Y تشکیل می‌شود، بشرطی که مقدار X از هر سطر به سطر بعدی یک واحد بسمت چپ جابجا شود. چون کامپیوتر در هر لحظه قادر است دو عدد را با هم جمع کند، ما خانه‌ای از



X حاوی مضروب است
Y حاوی مضروب‌فیه است
P محل تشکیل حاصلضرب است

مثال با چهار رقم معنی‌دار

	P
X = 0000 1111	0000 0000
Y = 0000 1011	0000 1111
0000 1111	0010 1101
0001 1110	0010 1101
0000 0000	1010 0101
0111 1000	
1010 0101	

شکل ۳-۶ فلوچارت برنامه ضرب

حافظه را که با P نشان داده شده برای حفظ مجموع‌های میانی[1] در نظر می‌گیریم. مجموع‌های میانی، حاصلضرب‌های جزیی خوانده می‌شوند زیرا حاوی بخشی از حاصلضرب هستند که تا آن زمان بدست آمده است. همانطور که در مثال عددی در شکل نشان داده شده است، حاصلضرب جزیی با صفر شروع می‌شود، مضروب، X، به محتوای P، بازاء هر بیت از مضروب‌فیه Y که 1 است، اضافه می‌شود. پس از وارسی هر یک از بیت‌های مضروب‌فیه، مقدار X بسمت چپ شیفت داده می‌شود. مقدار نهایی حاصل در P حاصلضرب را تشکیل می‌دهد. وقتی که اعداد در هم ضرب شدند، حاصلضرب دارای چهار بیت معنی‌دار است. پس از عمل ضرب، حاصلضرب هشت بیت معنی‌دار خواهد داشت. کامپیوتر می‌تواند از اعداد با هشت بیت معنی‌دار استفاده کند و لذا حاصلضرب تا 16 بیت طول خواهد داشت.

فلوچارت شکل ۳-۶ رویه برنامه‌نویسی عمل ضرب را قدم به قدم نشان می‌دهد. برنامه دارای حلقه‌ای است که هشت بار طی می‌شود، یعنی هر بار بازاء یک بیت معنی‌دار مضروب‌فیه. در ابتدا، مکان X حاوی مضروب و مکان Y حاوی مضروب‌فیه است. شمارنده CTR با 8- بار می‌شود و مکان P به 0 پاک می‌گردد.

هر بیت مضروب‌فیه پس از انتقال به E وارسی می‌شود. این عمل با پاک کردن E، بارکردن Y در AC، چرخش E و AC به راست و ذخیره عدد شیفت یافته به Y انجام می‌گردد. بیت ذخیره شده در E کم‌ارزشترین بیت مضروب‌فیه است. اکنون مقدار E را وارسی می‌کنیم. اگر E برابر 1 باشد، مضروب، X، با حاصلضرب جزیی P جمع می‌شود. ولی اگر E برابر 0 باشد حاصلضرب جزیی تغییری نمی‌کند. سپس مقدار X را با بارکردن آن در AC و چرخش E و AC، یک واحد بچپ شیفت می‌دهیم. حلقه هشت بار، با افزایش CTR و رسیدن آن به صفر تکرار می‌گردد. وقتی که شمارنده به صفر برسد، برنامه از حلقه خارج می‌شود در حالی که نتیجه در مکان P قرار دارد.

برنامه جدول ۱۴-۶ دستورات ضرب دو عدد بدون علامت را نشان می‌دهد. مرحله مقداردهی اولیه[2] آورده نشده است ولی بهنگام بارکردن برنامه در کامپیوتر باید انجام شود. مقداردهی اولیه شامل آوردن مضروب و مضروب‌فیه به ترتیب در خانه‌های X و Y؛ دادن مقدار 8- به شمارنده؛ و دادن مقدار صفر به P می‌باشد. اگر این مکان‌ها مقداردهی نشوند، برنامه ممکن است با داده‌های غلط اجرا شود. برنامه، خود سر راست است و مراحل لیست شده در فلوچارت را دنبال می‌کند. توضیحات برنامه هم می‌تواند به دنبال کردن قدم به قدم رویه انجام کار کمک کند.

مثال فوق نشان داد که اگر کامپیوتری فاقد دستور ماشین برای کار مورد نظر می‌باشد، آن عمل را می‌توان با رشته‌ای از دستورالعمل‌ها برنامه‌نویسی کرد. بنابراین ما پیاده‌سازی نرم‌افزاری عمل ضرب را نشان دادیم. پیاده‌سازی سخت‌افزاری متناظر آن در بخش ۳-۱۰ آمده است.

جمع با دقت مضاعف

هنگامی‌که دو عدد 16 بیتی بی‌علامت درهم ضرب شوند، نتیجه حاصلضرب 32 بیتی خواهدبودکه باید در

1- Intermediate Sum 2- Initialization