جدول ۱۴-۶ برنامه ضرب دو عدد مثبت

```
        ORG 100
LOP,    CLE          / پاک کردن E
        LDA Y        / بارکردن مضروب‌فیه
        CIR          / انتقال بیت مضروب‌فیه به E
        STA Y        / ذخیره مضروب‌فیه جابجا شده
        SZE          / چک بیت، صفر است یا نه
        BUN ONE      / بیت 1 است، به ONE برو
        BUN ZRO      / بیت 0است، به ZRO برو
ONE,    LDA X        / بارکردن مضروب
        ADD P        / جمع با حاصلضرب جزیی
        STA P        / ذخیره حاصلضرب جزیی
        CLE          / پاک کردن E
ZRO,    LDA X        / بارکردن مضروب
        CIL          / چرخش به چپ
        STA X        / ذخیره مضروب جابجا شده
        ISZ CTR      / افزایش شمارنده
        BUN LOP      / شمارنده صفر نیست، تکرار حلقه
        HLT          / شمارنده صفر است، توقف
CTR,    DEC -8       / محل شمارنده
X,      HEX 000F     / محل ذخیره مضروب
Y,      HEX 000B     / محل ذخیره مضروب‌فیه
P,      HEX 0        / محل تشکل حاصلضرب
        END
```

دو کلمه حافظه ذخیره شود. هرگاه عددی در دو کلمه حافظه جای گیرد گوییم دقت مضاعف دارد. وقتی که یک حاصلضرب جزیی محاسبه می‌شود، لازم است یک عدد با دقت مضاعف، با مضروب شیفت یافته که خود عددی با دقت مضاعف است، جمع شود. برای دقت بیشتر، برنامه نویس ممکن است بخواهد از اعداد با دقت مضاعف استفاده کند و محاسبه را با عملوندهایی که دو کلمه از حافظه را اشغال می‌کنند انجام دهد. در این بخش برنامه‌ای را ارائه می‌دهیم که دو عدد با دقت مضاعف را با هم جمع می‌کند.

یکی از دو عدد با دقت مضاعف در دو حافظه متوالی، یعنی AL و AH ، قرار داده می‌شود که AL حاوی 16 بیت کم ارزشتر است. عدد دیگر در BL و BH قرار داده می‌شود. برنامه در جدول ۱۵-۶ لیست شده است. دو بخش کم ارزشتر با هم جمع شده و رقم نقلی به E منتقل می‌شود. AC پاک شده و بیت موجود در E به داخل کم‌ارزشترین بیت AC چرخش داده می‌شود. سپس دو بخش پرارزشتر با نقلی جمع می‌شوند و مجموع با دقت مضاعف در CL و CH ذخیره می‌شود.

اعمال منطقی

کامپیوتر پایه سه دستورالعمل ماشین منطقی AND ، CMA و CLA را داراست. دستورالعمل LDA را هم می‌توان یک دستورالعمل منطقی تصور کرد که یک عملوند منطقی را به AC منتقل می‌کند.

جدول ۱۵-۶ برنامه جمع دو عدد با دقت مضاعف

```
        LDA AL      / بار کردن بخش کم‌ارزشتر A
        ADD BL      / جمع بخش کم‌ارزشتر B ، نقلی در E
        STA CL      / ذخیره در بخش کم‌ارزشتر C
        CLA         / پاک کردن AC
        CIL         / چرخش برای آوردن نقلی به درون AC(0)
        ADD AH      / جمع کردن بخش باارزشتر A و نقلی
        ADD BH      / جمع کردن بخش پرارزشتر B
        STA CH      / ذخیره در بخش پرارزشتر C
        HLT
AL,     —           / محل عملوندها
AH,     —
BL,     —
BH,     —
CL,     —
CH,     —
```

در بخش ۴-۵ فهرست 16 عمل منطقی مختلف را ارائه نمودیم. هر شانزده عمل منطقی را می‌توان بصورت نرم‌افزاری پیاده‌سازی کرد زیرا هر تابع منطقی را می‌توان با AND و متمم‌سازی پیاده سازی نمود. مثلاً عمل OR در دستورات کامپیوتر پایه وجود ندارد. امّا با استفاده از تئوری دمورگان رابطه $x + y = (x'y')'$ را می‌توان نتیجه گرفت. عبارت دوم فقط از اعمال AND و متمم سازی تشکیل شده است. برنامه‌ای که عمل OR را روی دو عملوند منطقی A و B انجام می‌دهد بصورت زیر است:

```
LDA A       Load first operand A                    / اولین عملوند در A را بارکن
CMA         Complement to get Ā                     / با متمم سازی Ā را بدست بیاور
STA TMP     Store in a tempevary location           / حاصل را در یک مکان موقت ذخیره کن
LDA B       Load Second operand B                   / عملوند دوم را در B بارکن
CMA         Complement to get B̄                     / بامتمم سازی B̄ را بدست بیاور
AND TMP     AND with Ā to get Ā∧ B̄                  / حاصل را Ā با برای بدست آوردن Ā∧ B̄ ، AND کن
CMA         Complement again to get A ∧ B           / حاصل را مجدداً متمم کن تا AVB بدست آید
```

سایر اعمال منطقی را نیز می‌توان بطور مشابه با نرم‌افزار پیاده‌سازی کرد.

اعمال شیفت

اعمال شیفت چرخشی از جمله دستورات ماشین در کامپیوترها هستند. شیفت‌های مورد نظر دیگر عبارتند از شیفت‌های منطقی و شیفت‌های حسابی. این دو نوع شیفت را بسادگی با تعداد کمی از دستورالعمل‌ها می‌توان برنامه‌نویسی کرد.

شیفت منطقی مستلزم این است که رقم 0 به یکی از دو انتها اضافه شود. این بسادگی با پاک کردن E و

چرخش AC و E میسر است. بنابراین برای یک عمل منطقی شیفت به راست ما نیاز به دو دستورالعمل زیر داریم:

```
CLE
CIR
```

برای شیفت به چپ دو دستور زیر مورد نیاز است:

```
CLE
CIL
```

شیفت حسابی به نوع نمایش اعداد منفی بستگی دارد. برای کامپیوتر پایه، ما نمایش متمم 2 علامت‌دار را برگزیدیم. قواعد شیفت حسابی در بخش ۶-۴ لیست شده است. برای یک عمل شیفت به راست حسابی لازم است تا بیت علامت در سمت چپ‌ترین مکان بلاتغییر بماند. اما خود بیت علامت بداخل بیت مرتبه بالاتر عدد شیفت پیدا می‌کند. برنامه شیفت براست حسابی مستلزم این است تا E را برابر مقدار بیت علامت کنیم و سپس چرخش به راست را انجام دهیم. بنابراین:

```
CLE     E را با 0 پاک کن
SPA     اگر AC مثبت است گذر کن
CME     AC منفی است، E برابر 1 شود
CIR     چرخش E و AC
```

برای شیفت به چپ حسابی لازم است تا بیت اضافه شده به منتهی‌الیه سمت راست 0 باشد. این کار بسادگی با صفر کردن E قبل از عمل چرخش به چپ انجام می‌شود. بیت علامت نباید در حین شیفت تغییر نماید. با یک دستور چرخش، بیت علامت بداخل E منتقل می‌شود. سپس لازم است تا بیت علامت را پس از عمل با E مقایسه کنیم. اگر دو مقدار برابر باشند، شیفت حسابی بطور صحیحی انجام گرفته است، ولی اگر برابر نباشند، سرریز رخ داده است. سرریز بدان معنی است که عدد قبل از شیفت بیش از حد بزرگ بوده است و هنگامی که در 2 ضرب شده (توسط عمل شیفت) عدد حاصل از ظرفیت AC تجاوز کرده است.

۶-۷ زیرروال‌ها[1]

مکرراً اتفاق افتاده است که یک قطعه از کد در بخش‌های مختلفی از برنامه به طور تکراری نوشته شود. مسلماً بجای نوشتن کد در هر بار که لازم باشد بهتر است دستورات مشترک فقط یکبار نوشته شوند. مجموعه دستورالعمل‌های مشترکی که در یک برنامه بتوان از آنها بدفعات زیاد استفاده کرد زیر روال نام دارد. هر بار که زیر روال در بخش اصلی برنامه بکار رود، انشعابی به ابتدای زیرروال صورت

1- Subroutine

می‌گیرد. پس از اجرای زیرروال، انشعابی دیگر موجب بازگشت به برنامه اصلی می‌گردد.

زیر روال از رشته مستقلی از دستورالعمل‌ها تشکیل می‌شود که کار معینی را انجام می‌دهند. از هر جای برنامه اصلی می‌توان به زیر روال انشعاب کرد. در اینجا این سؤال مطرح می‌شود که زیرروال چگونه می‌فهمد که به چه محلی باز گردد، زیرا ممکن است از مکان‌های متعددی در برنامه اصلی به زیرروال مزبور انشعاب گردد. بنابراین لازم است آدرس بازگشت را جایی در کامپیوتر ذخیره کنیم تا زیرروال بداند به کجا بازگردد. چون انشعاب به زیر روال و بازگشت به برنامه اصلی عمل متداولی است، تمام کامپیوترها دستورات خاصی را برای ورود به زیر روال و بازگشت از آن در اختیار می‌گذارند.

در کامپیوتر پایه، رابط بین برنامه اصلی و یک زیرروال دستورالعمل BSA[1] می‌باشد. برای شرح چگونگی استفاده از این دستور، زیر روالی می‌نویسیم که محتوای یک انباره را چهار بار به چپ شیفت دهد. چهار بار شیفت یک کلمه برای پردازش اعداد دهدهی کد شده با دودویی، BCD، و یا کاراکترهای الفبا عددی عمل مفیدی است. چنین دستوری می‌توانست بعنوان دستورالعمل ماشین در کامپیوتر پیش‌بینی شود. اما چون این کار صورت نگرفته است، زیرروالی برای این کار ایجاد می‌شود. برنامه جدول ۱۶-۶ با بارکردن X در AC شروع می‌شود. دستورالعمل بعدی که با آن مواجه می‌شویم BSA

جدول ۱۶-۶ برنامه‌ای برای نمایش طریقه استفاده از زیرروال‌ها

مکان			
		ORG 100	برنامه اصلی /
100		LDA X	باردهی X /
101		BSA SH4	انشعاب به زیرروال /
102		STA X	ذخیره عدد جابجا شده /
103		LDA Y	بارکردن Y /
104		BSA SH4	انشعاب دوباره به زیرروال /
105		STA Y	ذخیره عدد جابجا شده /
106		HLT	
107	X,	HEX 1234	
108	Y,	HEX 4321	
			زیرروال چهار بار جابجایی /
109	SH4,	HEX 0	عمل ذخیره آدرس بازگشت /
10A		CIL	یک بار چرخش به چپ /
10B		CIL	
10C		CIL	
10D		CIL	چرخش به چپ برای چهارمین بار /
10E		AND MSK	صفر کردن AC(13-16) /
10F		BUN SH4 I	بازگشت به برنامه اصلی /
110	MSK,	HEX FFF0	عملوند پوشش /
		END	

1- Branch and Save Address

SH4 است. دستورالعمل BSA در مکان 101 است. زیر روال SH4 پس از اتمام کار خود باید به مکان 102 باز گردد. هنگامی که دستور BSA اجرا شود واحد کنترل آدرس بازگشت 102 را در مکانی که توسط آدرس سمبلیک SH4 تعریف می‌شود (که 109 است) ذخیره می‌کند. همچنین محتوای SH4+1 را بداخل شمارنده برنامه منتقل می‌سازد. پس از اجرای این دستورالعمل، مکان حافظه 109 حاوی دودویی معادل با شانزده شانزدهی 102 می‌باشد و شمارنده برنامه حاوی دودویی معادل با شانزده شانزدهی 10A خواهد بود. با این عمل آدرس بازگشت ذخیره شده است و اینک زیرروال با شروع از 10A اجرا می‌شود (زیرا این مقدار محتوای PC در برداشت سیکل بعدی است. محاسبات در زیر روال محتوای AC را چهار بار به چپ شیفت می‌دهد. برای انجام یک عمل شیفت منطقی چهار بیت پائین رتبه باید 0 شوند. این کار با ماسک کردن (پوشش دادن) FFF0 با محتوای AC انجام می‌شود. عمل ماسک، یک عمل AND منطقی است که هر جا بیت عملوند ماسک صفر باشند بیت متناظر در AC را پاک می‌کند و هر جا بیت‌های عملوند ماسک 1 باشند بیت‌های AC را بدون تغییر باقی می‌گذارند.

آخرین دستورالعمل، زیر روال کامپیوتر را به برنامه اصلی باز می‌گرداند. این عمل با دستورالعمل انشعاب غیرمستقیم و سمبل آدرس مورد استفاده برای نام زیر روال صورت می‌گیرد. آدرسی که کامپیوتر به آن انشعاب می‌کند SH4 نیست بلکه مقداری است که در مکان SH4 قرار دارد، زیرا دستورالعمل از نوع غیرمستقیم است. آنچه در SH4 یافت می‌شود آدرس بازگشت 102 می‌باشد که قبلاً در آنجا با دستورالعمل BSA ذخیره شد. کامپیوتر برای اجرای دستورالعمل واقع در مکان 102 باز می‌گردد. برنامه اصلی با ذخیره کردن عدد شیفت یافته در مکان X به کار خود ادامه می‌دهد. سپس عدد جدیدی از Y بداخل AC بار شده و انشعاب دیگری به زیرروال صورت می‌گیرد. این بار مکان SH4 حاوی آدرس برگشت 105 خواهد بود زیرا این مقدار مکان دستورالعمل بعدی پس از BSA است. عملوند جدید شیفت پیدا کرده و زیر روال به برنامه اصلی در مکان 105 باز می‌گردد.

از این مثال می‌بینیم که اولین مکان حافظه هر زیر روال نقش یک رابط بین برنامه اصلی و زیرروال را بعهده دارد. رویه انشعاب به زیرروال و بازگشت به برنامه اصلی پیوند[1] زیر روال نامیده می‌شود. عملی را که دستورالعمل BSA انجام می‌دهد معمولاً فراخوانی[2] زیرروال می‌نامند. آخرین دستورالعمل زیرروال نیز عملی را که بازگشت[3] از زیرروال می‌خوانند انجام می‌دهد.

روش بکار رفته در کامپیوتر پایه برای پیوند زیرروال عمدتاً در کامپیوترهایی یافت می‌شود که فقط یک ثبات پردازنده دارند. بسیاری از کامپیوترها دارای چندین ثبات پردازنده هستند و به برخی از آنها نام ثبات اندیس[4] (شاخص) داده شده است. در چنین کامپیوترهایی، یک ثبات اندیس برای پیوند زیرروال بکار می‌رود. دستورالعمل انشعاب به زیر روال، آدرس برگشت را در ثبات اندیس ذخیره می‌کند. دستورالعمل بازگشت از زیر روال، با انشعاب به آدرسی که در آن موقع در ثبات شاخص است صورت می‌گیرد.

1- Linkage 2- Call 3- Return
4- Index Register

پارامترهای زیرروال و داده‌های پیوند

هنگامی که زیرروالی فراخوانده شود، برنامه اصلی باید داده‌هایی راکه مایل است زیرروال با آنها کار کند به آن انتقال دهد. در مثال قبل، داده از طریق انباره منتقل شد. عملوند قبل از انشعاب در AC بار گردید. زیر روال عدد را شیفت داد و آن را در همان جا باقی گذاشت تا برنامه اصلی از آن استفاده کند. بطور کلی، لازم است تا زیرروال به داده‌های برنامه خواننده دسترسی داشته و نتایج را به آن بازگرداند. از انباره می‌توان برای یک پارامتر ورودی و یک پارامتر خروجی استفاده کرد. در کامپیوترهایی که ثبات‌های متعدد دارند تعداد بیشتری پارامتر بدین طریق قابل انتقال است. راه دیگری برای انتقال داده به یک زیر روال از طریق حافظه است. داده‌ها غالباً در مکان‌هایی از حافظه که بدنبال فراخوانی آمده‌اند قرار داده می‌شوند. همچنین می‌توان آنها را در بلاک‌هایی از حافظه نیز ذخیره کرد. سپس اولین آدرس بلاک در مکان بعد از فراخوانی قرار می‌گیرد. در هر حال، آدرس بازگشت همواره اطلاعات ارتباطی برای انتقال داده بین برنامه اصلی و زیرروال را فراهم می‌آورد.

بعنوان مثال، زیرروالی را در نظر بگیرید که عمل منطقی OR را انجام دهد. دو عملوند باید به زیر روال منتقل و زیر روال هم باید نتیجه عمل را بازگرداند. از انباره می‌توان برای انتقال یک عملوند و دریافت نتیجه استفاده کرد. عملوند دیگر در مکانی که بدنبال دستورالعمل BSA می‌آید وارد می‌شود. این مطلب در برنامه جدول ۱۷-۶ نشان داده شده است. اولین عملوند در مکان X بداخل AC بار می‌شود. عملوند دوم در مکان 202که بدنبال دستور BSA است ذخیره می‌شود. پس از انشعاب، اولین

جدول ۱۷-۶ برنامه‌ای برای نمایش اشتراک پارامترها

مکان			
		ORG 200	
200		LDA X	/ بارکردن عملوند اول در AC
201		BSA OR	/ انشعاب به زیرروال OR
202		HEX 3AF6	/ محل ذخیره عملوند دوم
203		STA Y	/ محل برگشت از زیرروال
204		HLT	
205	X,	HEX 7B95	/ محل ذخیره عملوند اول
206	Y,	HEX 0	/ محل ذخیره نتیجه
207	OR,	HEX 0	/ زیرروال OR
208		CMA	/ متمم کردن عملوند اول
209		STA TMP	/ ذخیره در مکان موقت
20A		LDA OR I	/ بار کردن عملوند دوم
20B		CMA	/ متمم کردن عملوند دوم
20C		AND TMP	/ AND کردن متمم عملوند اول
20D		CMA	/ متمم مجدد برای یافتن OR
20E		ISZ OR	/ افزایش آدرس بازگشت
20F		BUN OR I	/ بازگشت به برنامه اصلی
210	TMP	HEX 0	/ محل ذخیره موقت
		END	

مکان در زیر روال، عدد 202 را نگهمیدارد. دقت کنید که در این حالت، 202 آدرس بازگشت نیست بلکه آدرس عملوند دوم است. زیر روال، عمل OR را با متمم کردن عملوند اول که در AC قرار دارد و ذخیره آن در یک مکان موقت به نام TMP آغاز می‌کند. دومین عملوند توسط دستورالعمل غیرمستقیم در مکان OR به AC بار می‌شود. بیاد بیاورید که مکان OR حاوی عدد 202 است. با مراجعه غیرمستقیم دستور به این مکان عملوند موجود در مکان 202 در AC بار می‌شود. این عملوند متمم شده و سپس با عملوند ذخیره شده در TMP، AND می‌شود. متمم کردن نتیجه، حاصل عمل OR را می‌دهد.

بازگشت از زیرروال باید بگونه‌ای صورت گیرد که برنامه اصلی از خانه 203، که دستورالعمل بعدی در آن قرار دارد ادامه یابد. این کار با افزایش آدرس OR توسط دستورالعمل ISZ صورت می‌گیرد. اکنون مکان OR عدد 203 را نگهمیدارد و یک دستورالعمل BUN غیرمستقیم موجب بازگشت به محل صحیح می‌گردد.

امکان دارد بیش از یک عملوند بدنبال دستورالعمل BSA داشت. زیر روال باید آدرس برگشت ذخیره شده در اولین خانه خود را، بازاء هر عملوندی که از برنامه فراخواننده استخراج می‌کند، افزایش دهد. به علاوه برنامه فراخواننده می‌تواند یک یا چند مکان را برای نتایج محاسبه شده در زیر روال حفظ کند. اولین مکان در زیر روال باید برای این مکان‌ها قبل از بازگشت افزایش داده شود. اگر حجم زیادی از داده‌ها برای انتقال وجود داشته باشد، آنها را می‌توان بصورت بلاک ذخیره کرده و آدرس اولین قلم بعنوان پارامتر پیوندی بکار گرفته شود.

زیر روالی که یک بلاک داده را از آدرس 100 به بلاک دیگری از آدرس 200 منتقل می‌کند در جدول ۱۸-۶ آورده شده است. طول بلاک 16 کلمه می‌باشد. اولین دستورالعمل، انشعاب به MVE است. اولین بخش زیرروال سه پارامتر 100، 200 و 16- را از برنامه اصلی منتقل و در مکان‌های مربوطه‌شان قرار می‌دهد. اقلام داده از بلاک‌های خود بکمک دو اشاره‌گر برداشته می‌شوند. شمارنده انتقال تنها 16 قلم را تضمین می‌کند. وقتی که زیر روال کار خود را تمام می‌کند، داده‌های مورد نظر در بلاکی قرار دارند که با مکان 200 شروع می‌شود. بازگشت برنامه به دستورالعمل HLT است.

۸-۶ برنامه‌نویسی ورودی - خروجی

کاربران کامپیوتر برنامه‌ها را با استفاده از سمبل‌هایی می‌نویسند که بوسیله زبان برنامه‌نویسی بکار رفته تعریف شده‌اند. سمبل‌ها رشته‌هایی از کاراکترها هستند و به هر کاراکتر یک کد 8 بیت اختصاص می‌یابد بطوری که می‌توان آنرا در حافظه کامپیوتر ذخیره کرد. یک کاراکتر کد شده به دودویی هنگامی وارد کامپیوتر می‌شود که یک دستورالعمل INP [1] (ورودی) اجرا شود. یک کاراکتر کد شده به دودویی وقتی به دستگاه خروجی [2] منتقل می‌شود که یک دستورالعمل OUT [3] اجرا گردد. وسیله خروجی کد دودویی را تشخیص داده و کاراکتر مربوط به آن را تایپ می‌کند.

جدول ۱۹-۶ (الف) دستورات لازم برای وارد کردن یک کاراکتر و ذخیره آن را در حافظه نشان

1- Input 2- Output Device 3- Output

جدول ۱۸-۶ زیرروال انتقال بلاک داده‌ها

```
                        / برنامه اصلی
        BSA MVE         / انشعاب به زیر روال
        HEX 100         / آدرس شروع مبدأ داده‌ها
        HEX 200         / آدرس شروع مقصد داده‌ها
        DEC -16         / تعداد اقلامی که باید انتقال یابد
        HLT
MVE,    HEX 0           / زیرروال MVE
        LDA MVE I       / بارکردن آدرس مبدأ
        STA PT1         / ذخیره در اشاره گر اول
        ISZ MVE         / افزایش آدرس بازگشت
        LDA MVE I       / بارکردن آدرس مقصد
        STA PT2         / ذخیره در اشاره گر دوم
        ISZ MVE         / افزایش آدرس برگشت
        LDA MVE I       / بارکردن تعداد اقلام
        STA CTR         / ذخیره در شمارنده
        ISZ MVE         / افزایش آدرس برگشت
LOP,    LDA PT1 I       / بارکردن یک قلم از داده‌های مقصد
        STA PT2 I       / ذخیره در مقصد
        ISZ PT1         / افزایش اشاره گر مبدأ
        ISZ PT2         / افزایش اشاره گر مقصد
        ISZ CTR         / افزایش شمارنده
        BUN LOP         / 16 بار تکرار
        BUN MVE I       / بازگشت به برنامه اصلی
PT1,    —
PT2,    —
CTR,    —
```

جدول ۱۹-۶ برنامه ورود و خروج یک کاراکتر

```
(الف): ورودی یک کاراکتر
CIF,    SKI             / چک کردن پرچم ورودی
        BUN CIF         / پرچم برابر 0 است، انشعاب برای وارسی مجدد
        INP             / پرچم 1 است، دریافت کاراکتر
        OUT             / چاپ کاراکتر
        STA CHR         / ذخیره کاراکتر
        HLT
CHR,    —               / محل ذخیره کاراکتر
(ب): خروجی یک کاراکتر
        LDA CHR         / بارکردن کاراکتر در AC
COF,    SKO             / چک کردن پرچم خروجی
        BUN COF         / پرچم 0 است، انشعاب برای وارسی مجدد
        OUT             / پرچم برابر 1 است، ارسال کاراکتر
        HLT
CHR,    HEX 0057        / کاراکتر مورد نظر "W" است
```

می‌دهد. دستورالعمل SKI پرچم[1] ورودی را چک می‌کند تا وجود یک کاراکتر را برای انتقال دریابد. اگرپرچم ورودی 1 باشد دستور بعدی گذر می‌شود. دستورالعمل INP کاراکتر کد شده به دودویی را به AC (0–7) می‌فرستد. سپس کاراکتر توسط دستورالعمل OUT چاپ می‌شود. واحد پایانه‌ای[2] (ترمینال) که مستقیماً به کامپیوتر متصل است، با فشار دادن کلید روی صفحه کلید، کاراکتری را چاپ نمی‌کند. برای تایپ آن، لازم است دستورالعمل OUT برای چاپگر ارسال شود. باین ترتیب، کاربر از یک انتقال صحیح مطمئن خواهد شد. اگر دستورالعمل SKI بیت پرچم را برابر 0 بیابد، دستورالعمل متوالی بعدی را اجرا خواهد کرد. این دستورالعمل، انشعاب برای بازگشت و وارسی مجدد بیت پرچم است. چون دستگاه (وسیله) ورودی، خیلی آهسته‌تر از کامپیوتر است، دو دستورالعمل حلقه، قبل از انتقال یک کاراکتر به داخل انباره، چند بار اجرا خواهند شد.

جدول ۱۹–۶ (ب) لیست دستورات لازم برای چاپ یک کاراکتر را که قبلاً در حافظه ذخیره شده نشان می‌دهد. ابتدا کاراکتر در AC بار می‌شود. سپس پرچم خروجی چک می‌گردد. اگر پرچم 0 باشد، کامپیوتر در حلقه متشکل از دو دستور باقی‌می‌ماند و بیت پرچم را چک می‌نماید. وقتی که پرچم به 1 تغییر یابد، کاراکتر از انباره به چاپگر منتقل می‌گردد.

دستکاری کاراکتر

یک کامپیوتر صرفاً یک ماشین حساب نیست بلکه دستکاری کننده سمبل‌ها نیز هست. کاراکترهای کد شده به دودویی که سمبل‌ها را می‌سازند می‌توانند بوسیله دستورالعمل‌های کامپیوتر برای انجام انواع کارهای داده پردازی دستکاری شوند. یکی از این کارها فشرده‌کردن دو کاراکتر در یک کلمه است. این کار ساده است زیرا هر کاراکتر 8 بیت را اشغال می‌کند و یک کلمه حافظه از 16 بیت تشکیل شده است. برنامه لیست شده در جدول ۲۰–۶ عبارتست از زیرروالی بنام IN2 که دو کاراکتر را وارد کرده و آنها را در یک کلمه فشرده می‌سازد. کلمه فشرده شده در انباره باقی‌می‌ماند. توجه کنید که زیر روال SH4 (جدول ۱۶–۶) برای هشت بار شیفت به چپ انباره دوبار فراخوانی شود.

در بحث اسمبلر فرض شد که برنامه سمبلیک در بخشی از حافظه که گاهی بافر[3] نامیده می‌شود، ذخیره می‌گردد. برنامه سمبلیک تایپ شده از طریق وسایل ورودی داخل شده و در مکان‌های متوالی حافظه در بافر ذخیره می‌گردد. برنامه لیست شده در جدول ۲۱–۶ می‌تواند برای ورود برنامه سمبلیک از صفحه کلید و فشرده‌کردن دو کاراکتر در یک کلمه و ذخیره آن در یک بافر بکار رود. اولین آدرس بافر 500 است. اولین جفت کاراکتر در مکان 500 ذخیره می‌شود و سایر کاراکترها در مکان‌های متوالی ذخیره می‌شوند. برنامه از اشاره‌گر برای دنبال‌کردن مکان‌های خالی موجود در بافر استفاده می‌کند. در برنامه از شمارنده استفاده نشده است، بنابراین مادامی که کاراکترها موجود باشند و یا تا رسیدن بافر به 0 (پس از FFFF) خوانده می‌شوند. در شرایط عملی ممکن است محدود کردن ظرفیت بافر لازم بنظر برسد و از یک

1- Flag 2- Terminal 3- Buffer

جدول ۲۰-۶ زیرروالی برای دریافت و فشرده کردن دو کاراکتر

```
IN2,    —          / محل ورود به زیرروال
FST,    SKI
        BUN FST
        INP        / دریافت اولین کاراکتر
        OUT
        BSA SH4    / چهار بار شیفت به چپ
        BSA SH4    / چهار بار شیفت به چپ
SCD,    SKI
        BUN SCD
        INP        / دریافت دومین کاراکتر
        OUT
        BUN IN2 I  / بازگشت
```

جدول ۲۱-۶ برنامه ذخیره کاراکترهای ورودی در یک بافر

```
        LDA ADS    / بارکردن آدرس ابتدای بافر
        STA PTR    / آغاز اشاره گر
LOP,    BSA IN2    / پرش به زیرروال IN2 (جدول ٦-٢٠)
        STA PTR I  / ذخیره کلمه دو کاراکتری در بافر
        ISZ PTR    / افزایش اشاره گر
        BUN LOP    / انشعاب برای دریافت کاراکترهای دیگر
        HLT
ADS,    HEX 500    / آدرس ابتدای بافر
PTR,    HEX 0      / محل اشاره گر
```

شمارنده برای این منظور استفاده گردد. توجه داشته باشید که زیر روال IN2 جدول ۲۰-۶ برای وارد کردن و فشرده کردن جفت کاراکترها فراخوانی می‌شود.

در بحث مربوط به مرور دوم اسمبلر در بخش ۴-۶، گفته شد که یکی از متداول‌ترین علمیات یک اسمبلر جستجوی جدول[1] است. این عملی است که طی آن یک جدول برای یافتن یک سمبل مورد نظر جستجو می‌شود. جستجو ممکن است، مقایسه سمبل مفروض با هر یک از سمبل‌های ذخیره شده در جدول باشد. وقتی که تطابقی رخ دهد و یا اینکه هیچ یک از سمبل‌ها با آن مطابقت نداشته باشند جستجو خاتمه می‌یابد. اگر تطابق اتفاق بیفتد، اسمبلر کد دودویی معادل را استخراج می‌کند. برای مقایسه دو کلمه برنامه‌ای در جدول ۲۲-۶ ارائه شده است. عمل مقایسه بدین ترتیب صورت می‌گیرد که متمم 2 یک کلمه بدست آمده و با کلمه دوم جمع حسابی می‌شود. اگر نتیجه صفر باشد، دو کلمه برابرند و تطابق رخ داده است. اگر نتیجه صفر نباشد، کلمات یکسان نیستند. این برنامه را می‌توان بصورت زیر روالی در یک برنامه جستجوی جدول مورد استفاده قرار داد.

1- Table look up

جدول ۲۲-۶ برنامه مقایسه دو کلمه

```
        LDA WD1     / بارکردن کلمه اول
        CMA
        INC         / تشکیل متمم 2
        ADD WD2     / اضافه کردن کلمه دوم
        SZA         / گذر اگر AC صفر باشد
        BUN UEQ     / انشعاب به روال "اگر نامساوی"
        BUN EQL     / انشعاب به روال "اگر مساوی"
WD1,    —
WD2,    —
```

برنامه وقفه[1]

زمان اجرای برنامه‌های ورودی و خروجی-عمدتاً مربوط به زمانی است که کامپیوتر با صرف آن، منتظر 1 شدن پرچم وسیله خارجی می‌ماند. این اتلاف زمانی ناشی از حلقه انتظاری است که عمل چک کردن پرچم را انجام می‌دهد و لذا کامپیوتر را به کاری که جز اتلاف وقت زیادی نیست مشغول نگهمیدارد. این زمان انتظار می‌تواند حذف شود بشرطی که برای اطلاع دادن 1 شدن پرچم از قابلیت وقفه استفاده شود. مزیت استفاده از وقفه این است که انتقال اطلاعات با درخواست از سوی وسیله خارجی آغاز می‌شود. در این احوال کامپیوتر می‌تواند مشغول اجرای کارهای مفید دیگری باشد. واضح است که اگر برنامه دیگری در حافظه مقیم نشده باشد کاری که کامپیوتر انجام دهد وجود نخواهد داشت، بنابراین می‌تواند همان چک کردن پرچم‌ها را انجام دهد. قابلیت وقفه در محیط‌های چند برنامه‌ای که دو یا چند برنامه در حافظه مقیم‌اند مفید است.

در هر لحظه از زمان، حتی اگر چند برنامه در حافظه مقیم باشد، تنها یک برنامه می‌تواند اجرا شود. برنامه‌ای که در حال حاضر در حال اجرا شدن است، برنامه جاری خوانده می‌شود. سایر برنامه‌ها معمولاً منتظر داده‌های ورودی یا خروجی هستند. کار وقفه این است که در حین اجرای برنامه، به انتقال داده‌های برنامه دیگر بپردازد. برنامه جاری باید شامل یک دستور ION برای فعال کردن وقفه‌ها باشد. اگر قابلیت وقفه بکار گرفته نشود، برنامه باید دارای دستور IOF باشد تا وقفه را از کار بیندازد. (کلید startکامپیوتر نیز وقفه را از کار می‌اندازد).

قابلیت وقفه اجازه می‌دهد تا برنامه جاری به کار خود ادامه دهد تا اینکه وسیله داخلی یا خارجی پرچم خود را 1 نماید. هر وقت پرچمی 1 شد، کامپیوتر اجرای دستور جاری را تمام کرده و به وقفه پاسخ[2] می‌دهد. نتیجه این عمل این است که آدرس بازگشت در مکان 0 ذخیره می‌گردد. سپس دستورالعمل در مکان 1 اجرا می‌شود؛ باین ترتیب یک روال سرویس‌دهی برای انتقال ورودی یا خروجی آغاز می‌گردد. روال سرویس‌دهی در هر جا از حافظه می‌تواند ذخیره شود بشرطی که

1- Interrupt 2- Acknowledge

دستورالعملی برای انشعاب به ابتدای روال در خانه 1 ذخیره شود. روال سرویس‌دهی بایستی دستوراتی برای انجام کارهای زیر داشته باشد.

۱- ذخیره محتویات ثبات‌های پردازنده
۲- وارسی اینکه کدام پرچم 1 است
۳- سرویس‌دهی به وسیله‌ای که پرچمش 1 است
۴- بازگرداندن محتوای ثبات‌های پردازنده
۵- فعال کردن قابلیت وقفه
۶- بازگشت به برنامه جاری

محتوای ثبات‌های پردازنده قبل از وقفه و بعد از بازگشت به برنامه جاری باید یکسان باشند؛ در غیر اینصورت برنامه جاری ممکن است در وضعیت خطا کار کند. چون روال سرویس‌دهی ممکن است این ثبات‌ها را بکار گیرد، لازم است تا محتوای آنها را در ابتدای روال ذخیره کرده و در انتها نیز بازگردانده شوند. ترتیب وارسی پرچم‌ها اولویت هر وسیله را تعیین می‌کند. اگرچه ممکن است دو یا چند پرچم بطور همزمان نشانده[1] شوند، ولی در هر زمان به یک وسیله سرویس داده می‌شود. ترتیب سرویس‌دهی بر اساس اولویت وسایل است.

وقوع یک وقفه مانع وقوع وقفه‌های دیگر می‌گردد. روال سرویس باید قبل از بازگشت به برنامه جاری وقفه‌ها را فعال کند. در نتیجه کامپیوتر حین اجرای برنامه جاری قادر خواهد بود وقفه‌های بعدی را بپذیرد. قابلیت وقفه نباید تا زمانی که آدرس بازگشت در شمارنده برنامه قرار نگرفته است فعال شود.

مثالی از یک برنامه که به یک وقفه سرویس می‌دهد در جدول ۲۳-۶ ارائه شده است. مکان 0 برای آدرس بازگشت نگه‌داشته شده است. مکان 1 حاوی دستورالعمل انشعاب به ابتدای روال سرویس SRV است. بخشی از برنامه جاری که نشان داده شده حاوی یک دستورالعمل ION است که وقفه‌ها را فعال می‌کند. فرض کنید در حالی که کامپیوتر دستورالعمل واقع در مکان 103 را اجرا می‌کند وقفه‌ای رخ دهد. سیکل وقفه معادل دودویی عدد شانزده شانزدهی 104 را در مکان 0 ذخیره کرده و به مکان 1 انشعاب می‌نماید. دستورالعمل انشعاب در مکان 1، کامپیوتر را به روال SRV می‌فرستد.

روال سرویس‌دهی شش کار ذکر شده فوق را اجرا می‌نماید. محتوای AC و E در مکان‌های خاصی ذخیره می‌شوند. (این مکان‌ها در کامپیوتر پایه فقط ثبات‌های پردازشگر هستند. پرچم‌ها متوالیاً وارسی می‌شوند مثلاً ابتدا پرچم ورودی و سپس پرچم خروجی. اگر یکی یا هر دو پرچم 1 شوند (نشانده شوند) یک قلم از داده به و یا از حافظه مربوطه منتقل می‌شود. قبل از بازگشت به برنامه جاری محتوای قبلی E و AC بازیابی شده و قابلیت وقفه فعال می‌گردد. آخرین دستور، انشعابی را به آدرس ذخیره شده در مکان 0 موجب می‌شود. این آدرسی است که قبلاً در حین وقفه ذخیره شده است. از اینرو برنامه جاری

1- Set

جدول ۲۳-۶ برنامه سرویس‌دهی به یک وقفه

مکان			
0	ZRO,	—	محل ذخیره آدرس برگشت /
1		BUN SRV	شعاب به روال سرویس /
100		CLA	قسمتی از برنامه در حال اجرا /
101		ION	فعال کردن قابلیت وقفه /
102		LDA X	
103		ADD Y	شروع وقفه /
104		STA Z	برنامه بعد از وقفه به این مکان بازمی‌گردد /
•		•	
•		•	
•			روال سرویس‌دهی به وقفه /
200	SRV,	STA SAC	ذخیره محتوای AC /
		CIR	انتقال E به AC(1) /
		STA SE	ذخیره محتوای E /
		SKI	چک کردن پرچم ورودی /
		BUN NXT	پرچم صفر است، پرچم بعدی چک شود /
		INP	پرچم 1 است، دریافت کاراکتر /
		OUT	چاپ کاراکتر /
		STA PT1 I	ذخیره کاراکتر در بافر ورودی /
		ISZ PT1	افزایش اشاره گر ورودی /
	NXT,	SKO	چک کردن پرچم خروجی /
		BUN EXT	پرچم صفر است، خروج /
		LDA PT2 I	بارکردن کاراکتر از بافر خروجی /
		OUT	ارسال کاراکتر /
		ISZ PT2	افزایش اشاره گر خروجی /
	EXT,	LDA SE	بازیابی مقدار AC(1) /
		CIL	انتقال آن به E /
		LDA SAC	بازیابی محتوای AC/
		ION	فعال کردن وقفه /
		BUN ZRO I	بازگشت به برنامه در حال اجرا /
	SAC,	—	محل ذخیره AC /
	SE,	—	محل ذخیره E /
	PT1,	—	اشاره گر میانگیر ورودی /
	PT2,	—	اشاره گر میانگیر خروجی /

از مکان 104 به کار ادامه خواهد داد، یعنی به همان جایی که در آن وقفه رخ داده بود.

در یک نمونه کامپیوتر امکان دارد چندین وسیله ورودی یا خروجی به مدار وقفه متصل شده باشد. بعلاوه، منابع وقفه محدود به وسایل ورودی و خروجی محدود نیستند. وقفه‌ها را می‌توان برای اهداف دیگری همچون خطاهای پردازشی درونی یا اخطارهای ویژه بکار برد. بحث‌های بیشتر درباره وقفه‌ها و برخی مفاهیم پیشرفته که اهمیت موضوع را بیان کنند در بخش ۵-۱۱ آورده شده است.

مسائل

۶-۱ برنامه زیر در واحد حافظه کامپیوتر پایه ذخیره شده است. محتوای AC ، PC و IR را به شانزده شانزدهی در پایان اجرای هر دستورالعمل نشان دهید. تمام اعداد در جدول به شانزده شانزدهی هستند.

مکان	دستورالعمل
010	CLA
011	ADD 016
012	BUN 014
013	HLT
014	AND 017
015	BUN 013
016	C1A5
017	93C6

۶-۲ برنامه زیر لیستی از دستورالعمل‌ها در مبنای شانزده است. کامپیوتر با شروع از آدرس 100 دستورالعمل‌ها را اجرا می‌کند. محتوای AC و کلمه حافظه درون آدرس 103 بهنگام توقف کامپیوتر چیست؟

مکان	دستورالعمل
100	5103
101	7200
102	7001
103	0000
104	7800
105	7020
106	C103

۶-۳ لیست برنامه زبان اسمبلی تولید شده بوسیله یک کامپایلر را از برنامه فرترن زیر بنویسید. متغیرها را عدد صحیح تصور کنید.

```
SUM = 0
SUM = SUM + A + B
DIF = DIF - C
SUM = SUM + DIF
```

۶-۴ آیا از حرف I می‌توان بعنوان یک آدرس سمبلیک در برنامه زبان اسمبلی برای کامپیوتر پایه استفاده کرد؟ توضیح دهید.

۶-۵ اگر سطری از کد که حاوی شبه دستورالعمل ORG یا END است عنوان نیز داشته باشد در مرور اول اسمبلر (شکل ۱-۶) چه رخ خواهد داد. فلوچارت را تغییر دهید تا در این صورت پیغام خطا بدهد.

۶-۶ سطری از کد در یک برنامه زبان اسمبلی بصورت زیر است:

DEC – 35

(الف) نشان دهید که برای ذخیره سطر کد به چهار کلمه حافظه نیاز است و محتوای دودویی آنها را بنویسید.
(ب) نشان دهید که یک کلمه حافظه کد ترجمه شده دودویی را ذخیره می‌نماید. محتوای دودویی آنها را بنویسید.

۶-۷ الف) جدول سمبل آدرس تولید شده برای برنامه جدول ۱۳-۶ در حین مرور اول بوسیله اسمبلر را بدست آورید.
ب) برنامه ترجمه شده را به شانزده شانزدهی بنویسید.

۶-۸ شبه دستورالعمل BSS N (بلاک آغاز شده بوسیله سمبل) گاهی برای رزرو N کلمه حافظه برای گروهی از عملوندها به کار می‌رود. مثلاً سطر کد

A, BSS 10

به اسمبلر اطلاع می‌دهد که بلاکی از 10 (دهدهی) مکان را، با شروع از مکان متناظر با سمبل A خالی بگذارد. این مشابه جمله DIMENSION A(10) در فرترن است. فلوچارت شکل ۱-۶ را برای پردازش این شبه دستور تصحیح کنید.

۶-۹ فلوچارت شکل ۲-۶ را تصحیح کنید بطوری که اگر یک دستور سمبلیک با عنوان تعریف نشده باشد یک پیام خطا تولید شود.

۶-۱۰ نشان دهید که چگونه جداول MRI و non–MRI می‌توانند در حافظه ذخیره شوند.

۶-۱۱ لیست برنامه زبان اسمبلی (معادل دودویی دستورالعمل‌ها) تولید شده بوسیله کامپایلر را برای عبارت زیر بدست آورید:

IF·(A – B) 10, 20, 30

برنامه به عبارت 10 انشعاب می‌کند اگر $A - B < 0$ باشد؛ یا به 20 اگر $A - B = 0$ باشد؛ و یا به 30 اگر $A - B > 0$ باشد.

۶-۱۲ الف) توضیح دهید که اجرای برنامه زیر موجب انجام چه کاری می‌شود. مقدار مکان CTR وقتی که کامپیوتر متوقف می‌شود چیست؟

```
ORG 100
CLE
CLA
STA CTR
LDA WRD
SZA
BUN ROT
BUN STP
```

```
ROT,    CIL
        SZE
        BUN AGN
        BUN ROT
AGN,    CLE
        ISZ CTR
        SZA
        BUN ROT
STP,    HLT
CTR,    HEX 0
WRD,    HEX 62C1
        END
```

ب) جدول سمبل‌های آدرس بدست آمده در مرور اول اسمبلر را بنویسید.

ج) کد شانزده شانزدهی برنامه ترجمه شده را بنویسید.

۶-۱۳ یک حلقه برنامه با استفاده از اشاره‌گر و شمارنده بنویسید تا محتویات مکان شانزده شانزدهی 500 تا 5FF را 0 کند.

۶-۱۴ برنامه‌ای بنویسید تا دو عدد را با روش جمع تکراری، در هم ضرب کند. مثلاً، برای ضرب 4 × 5 برنامه 5 را چهار بار با هم جمع کند، یعنی 5 + 5 + 5 + 5.

۶-۱۵ برنامه ضرب جدول ۶-۱۴ مقدار دهی اولیه نشده است. پس از یکبار اجرای برنامه، مکان CTR مقدار صفر می‌گیرد. نشان دهید که اگر برنامه مجدداً از مکان 100 شروع شود، حلقه 65536 بار تکرار خواهد شد. دستورات لازم برای مقداردهی اولیه[1] را برای برنامه اضافه کنید.

۶-۱۶ برنامه‌ای برای ضرب دو عدد مثبت بدون علامت 16 بیتی بنویسید و حاصلضرب آنها را بصورت یک عدد بدون علامت با دقت مضاعف به دست آورید.

۶-۱۷ برنامه‌ای برای ضرب دو عدد علامت‌دار بنویسید. اعداد منفی ابتدا بصورت متمم 2 علامت‌دار نشان داده می‌شوند. حاصلضرب باید با دقت منفرد باشد و در صورت منفی بودن هم بصورت متمم 2 علامت دار نشان داده شود.

۶-۱۸ برنامه‌ای بنویسید که دو عدد با دقت مضاعف را از هم تفریق کند.

۶-۱۹ برنامه‌ای بنویسید که OR انحصاری منطقی دو عملوند منطقی را بدست آورد.

۶-۲۰ برنامه‌ای برای شیفت حسابی به چپ بنویسید که در صورت سرریز به OVF انشعاب کند.

۶-۲۱ برنامه‌ای برای تفریق دو عدد بنویسید. در برنامه فراخوانده، مفروق و مفروق‌منه بدنبال دستورالعمل BSA آمده‌اند. تفاضل به سومین مکان پس از BSA در برنامه اصلی بازمی‌گردد.

۶-۲۲ برنامه‌ای بنویسید که هر داده را در یک بلاک متمم کند. در برنامه فراخوانده، دو پارامتر

1- Initialize

آدرس شروع بلاک و تعداد کلمات بلاک پس از دستورالعمل BSA آمده‌اند.

۶-۲۳ برنامه‌ای بنویسید که E و AC را چهار بار به راست بچرخاند. اگر ابتدا محتوای AC برابر $(079C)_{16}$ و E = 1 باشد محتوای آنها پس از اجرای زیر روال چیست؟

۶-۲۴ برنامه‌ای بنویسید که کاراکترهای ورودی را بپذیرد، هر دو کاراکتر بصورت یک کلمه فشرده کند و متوالیاً در بافر حافظه ذخیره نماید. اولین آدرس بافر $(400)_{16}$ و اندازه آن $(512)_{10}$ است. اگر بافر سرریز کند کامپیوتر متوقف شود.

۶-۲۵ برنامه‌ای بنویسید که دو کاراکتر واقع در مکان WRD را بردارد و از حالت فشرده خارج کند و آنها را در بیت‌های 0 تا 7 از مکان‌های CH1 و CH2 ذخیره کند. بیت‌های 9 تا 15 باید صفر باشد.

۶-۲۶ فلوچارتی را برای برنامه‌ای بدست آورید که کد CR (0D در مبنای شانزده) را در بافر حافظه جستجو کند. بافر دارای ۲ کد در هر کلمه است. وقتی به کد CR برخورد شود، برنامه آن را به بیت‌های 0 تا 7 مکان LNE منتقل می‌سازد بدون آن که بیت‌های 8 تا 15 تغییر کنند.

۶-۲۷ روال سرویس‌دهی SRV را از جدول ۲۳-۶ به کد شانزده شانزدهی معادلش ترجمه کنید. فرض کنید که روال از مکان 200 شروع شده باشد.

۶-۲۸ یک روال سرویس‌دهی وقفه که تمام عملیات لازم را انجام دهد بنویسید ولی وسیله ورودی فقط اگر مکان MOD تماماً 1 باشد سرویس‌دهی شود. وسیله خروجی هم فقط اگر مکان MOD تماماً 0 باشد سرویس‌دهی گردد.

www.homework.ir

www.homework.ir مشاوره و انجام پروژه هاي دانشجويي
دانلود فصلهايي از كتب دانشگاهي

۷ کنترل ریز برنامه‌نویسی شده

۷-۱ حافظه کنترل
۷-۲ دنبال کردن آدرس
۷-۳ مثال ریزبرنامه
۷-۴ طراحی واحد کنترل

۷-۱ حافظه کنترل

وظیفه یک واحد کنترل در یک کامپیوتر دیجیتال بکارگیری دنباله‌هایی از ریزعمل‌هاست. تعداد انواع ریزعمل‌های موجود در یک سیستم معین محدود است. پیچیدگی سیستم دیجیتال به تعداد ریزعمل‌های متوالی که اجرا می‌کند بستگی دارد. هنگامی که سیگنال‌های کنترلی توسط سخت‌افزار حاصل از تکنیک‌های مدارهای منطقی رایج تولید شود، گوییم واحد کنترل سخت‌افزاری[1] است. ریز برنامه‌نویسی روش دیگری برای طراحی واحد کنترل یک کامپیوتر دیجیتال است. اصل ریز برنامه‌نویسی روشی شاخص و سیستماتیک در کنترل رشته ریزعمل‌ها در یک کامپیوتر دیجیتال است.

تابع کنترلی که مشخص کننده یک ریزعمل است یک متغیر دودویی می‌باشد. وقتی که این متغیر در یکی از حالات دودویی باشد، ریز عمل متناظر آن اجرا می‌شود. اگر متغیر کنترلی که در وضعیت دودویی مخالف باشد وضعیت ثبات‌های سیستم را تغییر نمی‌دهد. وضعیت فعال یک متغیر کنترل ممکن است بسته به کاربرد هر یک از دو حالت 1 یا 0 باشد. در سیستم مبتنی بر گذرگاه[2]، سیگنال‌های کنترلی که ریزعمل را مشخص می‌نمایند بصورت گروه‌هایی از بیت‌ها هستند که مسیرها را در مولتی‌پلکسرها، دیکدرها و واحدهای حساب و منطق انتخاب می‌نمایند.

واحد کنترل موجب اجرای یک سری مراحل متوالی از ریز عمل‌ها می‌شود. در هر زمان مفروض،

1- Hardware = Hardwired 2- Bus Organized System

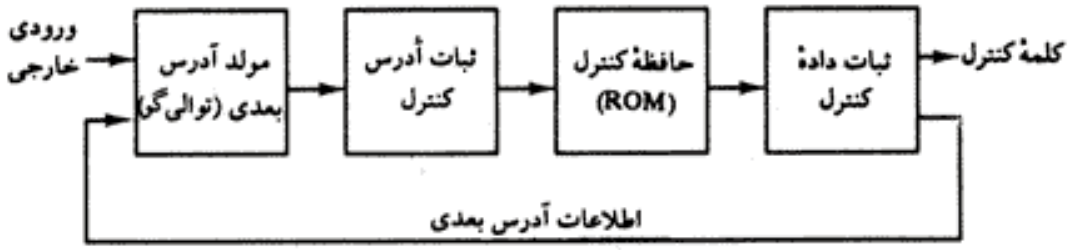
ریزعمل‌های معینی باید اجرا شوند، در حالیکه سایر ریزعمل‌ها راکد هستند. متغیرهای کنترلی در هر لحظه از زمان می‌توانند با رشته‌ای از 1 ها و 0 ها که کلمه کنترل نامیده می‌شود نمایش داده شوند. باین ترتیب می‌توان با برنامه‌نویسی کلمات کنترلی، عملیات مختلف را روی اجزاء سیستم انجام داد. واحد کنترلی که متغیرهای دودویی کنترل آن در حافظه ذخیره شده باشد واحد کنترل ریزبرنامه‌نویسی شده نام دارد. هر کلمه در واحد کنترل حاوی یک ریز دستورالعمل است. ریز دستورالعمل مشخص کننده یک یا چند ریزعمل برای سیستم است. رشته‌ای از ریزدستورالعمل‌ها، ریزبرنامه را تشکیل می‌دهند. با توجه به اینکه پس از نصب واحد کنترل در سیستم لازم نیست تغییری در ریزبرنامه داده شود، حافظه کنترل می‌تواند از نوع فقط خواندنی[1] (ROM) باشد. محتوای کلمات ROM ثابت است و آنها را بسادگی نمی‌توان با برنامه‌نویسی تغییر داد زیرا امکان نوشتن در ROM وجود ندارد. کلمات ROM حین ساخت واحد سخت‌افزار بطور دائمی ایجاد می‌شوند. بکارگیری یک ریزبرنامه بمعنی در اختیار قرار دادن تمام متغیرهای کنترل در کلمات ROM برای استفاده واحد کنترل از طریق خواندن‌های متوالی است. محتوای کلمه در ROM در یک آدرس مشخص یک ریزدستورالعمل را مشخص می‌کند.

در روش پیشرفته‌تری که برنامه‌نویسی دینامیکی نام دارد این امکان وجود دارد که ریزبرنامه ابتدا از طریق یک حافظه کمکی مانند دیسک مغناطیسی به سیستم بار شود. واحدهای کنترلی که از برنامه‌نویسی دینامیکی استفاده می‌کنند دارای حافظه‌هایی با قابلیت نوشتن هستند. این نوع حافظه را می‌توان برای نوشتن (در تغییر برنامه) بکار برد، ولی اکثراً برای خواندن مورد استفاده قرار می‌گیرند. حافظه‌ای که بخشی از واحد کنترل باشد حافظه کنترل[2] نامیده می‌شود.

کامپیوتری که از واحد کنترل ریزبرنامه‌نویسی شده استفاده می‌کند دو حافظه مجزا خواهد داشت: یک حافظه اصلی و یک حافظه کنترل. حافظه اصلی در اختیار کاربر است تا برنامه‌های خود را در آن ذخیره کند. محتوای حافظه اصلی بهنگام دستکاری داده‌ها و یا هر بار که برنامه عوض شود تغییر می‌نماید. برنامه کاربر در حافظه اصلی شامل دستورات ماشین و داده‌هاست. متقابلاً، حافظه کنترل یک ریزبرنامه ثابتی را نگهداری می‌کند و بوسیله هر کاربری قابل تغییر نیست. ریزبرنامه از ریزدستورالعمل‌هایی تشکیل شده که بنوبه خود انواع سیگنال‌های کنترل را برای اجرای ریزاعمال روی ثبات‌ها معین می‌کنند. هر دستورالعمل ماشین موجب اجرای رشته‌ای از ریزدستورالعمل‌های موجود در حافظه کنترل می‌شود. وظیفه این ریزدستورالعمل‌ها برداشت[3] دستورالعمل ماشین از حافظه اصلی، محاسبه آدرس موثر، اجرای عملی که توسط دستورالعمل ماشین مشخص شده، و بازگرداندن کنترل به فاز برداشت برای تکرار سیکل مربوط به دستورالعمل بعدی است.

آرایش کلی واحد کنترل ریزبرنامه‌نویسی شده در دیاگرام شکل ۱-۷ نشان داده شده است. فرض براین است که حافظه کنترل از نوع ROM بوده و همه اطلاعات کنترلی بصورت دائمی در آن ذخیره شده است. ثبات آدرس حافظه کنترل، آدرس ریزدستورالعمل را مشخص می‌کند، و ثبات داده کنترل

1- Read Only Memory 2- Control Memory 3- Fetch



شکل ۱-۷ سازمان کنترل ریزبرنامه‌نویسی شده

ریزدستورالعمل خوانده شده از حافظه را نگهمیدارد. ریزدستورالعمل حاوی یک کلمه کنترل است که یک یا چند ریزعمل را برای پردازشگر داده مشخص می‌نماید. به محض اجرای این ریزاعمال، کنترل باید آدرس بعدی را معین کند. مکان ریز دستورالعمل بعدی ممکن است خانه بعدی باشد، و یا ممکن است در جای دیگری از حافظه کنترل قرار داشته باشد. باین دلیل لازم است تعدادی از بیت‌های ریزدستورالعمل فعلی را برای کنترل ایجاد آدرس ریزدستورالعمل بعدی بکار برد. آدرس بعدی می‌تواند تابعی از شرایط ورودی خارجی نیز باشد. ضمن اجرای ریزاعمال، آدرس بعدی در مدار مولد آدرس بعدی تولید شده و سپس به ثبات آدرس کنترل منتقل می‌گردد تا ریزدستورالعمل بعدی را بخواند. بنابراین یک ریزدستورالعمل حاوی بیت‌هایی برای آغاز ریز اعمال در بخش پردازشگر داده و بیت‌هایی برای تعیین رشته آدرس‌ها برای حافظه کنترل است.

مولد آدرس بعدی گاهی اوقات توالی‌گر[1] ریزبرنامه نامیده می‌شود، زیرا آدرس‌های متوالی کلماتی را که از حافظه کنترل خوانده می‌شوند معین می‌سازد. آدرس ریزدستورالعمل بعدی بسته به ورودی توالی‌گر به چند طریق معین می‌شود. عملیات توالی‌گر ریزبرنامه عمدتاً عبارتست از افزایش ثبات آدرس کنترل به میزان یک واحد، بارکردن آدرس حافظه کنترل در ثبات آدرس کنترل، انتقال آدرس خارجی، یا بارکردن یک آدرس اولیه برای آغاز اعمال کنترل.

ثبات داده کنترل در حالی که آدرس بعدی محاسبه و از حافظه خوانده می‌شود، حاوی دستورالعمل فعلی است. ثبات داده گاهی اوقات ثبات خط لوله[2] هم خوانده می‌شود. این ثبات امکان اجرای ریزاعمال مشخص شده با کلمه کنترل، همزمان با تولید ریزدستورالعمل بعدی را فراهم می‌آورد. (برای توضیح بیشتر در مورد ثبات خط لوله به فصل‌های ۸ و ۹ مراجعه شود). این آرایش نیاز به یک پالس ساعت دو فاز دارد، که یکی از آنها به ثبات آدرس و دیگری به ثبات داده اعمال می‌شود.

سیستم می‌تواند با اعمال یک ساعت تک‌فاز به ثبات آدرس، بدون ثبات داده کنترل کارکند. کلمه کنترل و اطلاعات آدرس بعدی مستقیماً از حافظه کنترل دریافت می‌شود. باید دانست که ROM بمانند یک مدار ترکیبی کار می‌کند، که در آن مقدار آدرس ورودی و کلمه مربوط به آن آدرس خروجی آن است.

1- Sequencer 2- Pipeline Register

محتوای یک کلمه معین شده در ROM تا زمانی که مقدار آدرس در ثبات آدرس باقی‌بماند، در سیستم‌های خروجی باقی می‌ماند. در حافظه فقط خواندنی هیچ سیگنال خواندنی مورد نیاز نیست. هر پالس ساعت، ریزعمل‌های مشخص شده بوسیله کلمه کنترل را اجرا نموده و آدرس جدیدی را نیز به ثبات آدرس کنترل انتقال می‌دهد. در مثالی که بدنبال آمده است فرض خواهیم کرد پالس ساعت تک فاز باشد و لذا از ثبات داده کنترل استفاده نخواهد شد. باین ترتیب ثبات آدرس تنها جزء موجود در سیستم است که پالس‌های ساعت را دریافت می‌کند. دو جزء دیگر؛ توالی‌گر و حافظه کنترل مدارهای ترکیبی بوده و به پالس ساعت نیازی ندارند.

مزیت عمده کنترل ریزبرنامه نویسی شده این است که وقتی پیکربندی سخت‌افزاری تکمیل شد، سخت‌افزار اضافی و یا تغییر سیم‌بندی نخواهیم داشت. اگر بخواهیم کنترل متفاوتی را برای سیستم ایجاد کنیم، فقط لازم است مجموعه‌ای از دستورالعمل‌های متفاوتی را برای حافظه کنترل اختیار کنیم. پیکربندی سخت‌افزار برای عملیات متفاوت تغییر نمی‌کند؛ و تنها تغییری که باید صورت گیرد ریزبرنامه‌ای است که در حافظه کنترل جای داده می‌شود.

باید یادآوری کرد اکثر کامپیوترهای پایه‌ریزی شده بر مفهوم معماری کامپیوترهای کم‌دستور[1] یا RISC (به بخش ۸-۸ مراجعه شود) از کنترل سخت‌افزاری استفاده می‌کنند نه از حافظه کنترل ریزبرنامه. مثالی از کنترل سخت‌افزاری برای یک کامپیوتر ساده در بخش ۴-۵ ملاحظه شد.

۷-۲ توالی آدرس

ریزدستورالعمل‌ها بصورت گروهی در حافظه ذخیره می‌شوند و هر گروه نیز یک روال را تشکیل می‌دهند. هر دستورالعمل کامپیوتر دارای روال ریزبرنامه متعلق به خود در حافظه کنترل است که ریز اعمال مربوط به اجرای دستورالعمل را تولید می‌نماید. سخت‌افزاری که توالی آدرس حافظه کنترل را کنترل می‌نماید باید قادر به دنبال کردن ریز دستورالعمل‌های یک روال بوده و قابلیت انشعاب از یک روال به روال دیگر را هم داشته باشد. برای تحقق توالی دستورات در یک واحد ریزبرنامه کنترل، اجازه بدهید مراحلی راکه کنترل باید حین اجرای یک تک دستورالعمل کامپیوتر طی کند بشماریم.

هنگام روشن شدن کامپیوتر، آدرس اولیه‌ای در ثبات آدرس کنترل بار می‌شود. معمولاً این آدرس، آدرس اولین ریزدستورالعمل است که روال برداشت[2] دستورالعمل را فعال می‌سازد. اجرای روال برداشت می‌تواند با افزایش ثبات آدرس کنترل تا طی شدن کلیه ریزدستورالعمل‌های آن دنبال شود. در پایان روال برداشت، دستورالعمل در ثبات دستورالعمل کامپیوتر قرار گرفته است.

سپس حافظه کنترل باید به روالی مراجعه کند که طی آن آدرس موثر عملوند تعیین گردد. یک دستورالعمل ماشین ممکن است دارای بیت‌هایی باشد که روش‌های متعدد آدرس‌دهی مانند آدرس‌دهی غیرمستقیم و ثبات شاخص را مشخص کنند. روال محاسبه آدرس موثر در حافظه کنترل از طریق

1- Reduced Instruction Set Computer (RISC)

2- Fetch

ریزدستورالعمل انشعاب قابل دسترسی است، که این انشعابات مشروط بر وضعیت بیت‌های حالت دستورالعمل صورت می‌گیرد. وقتی که روال محاسبه آدرس موثر پایان یافت، آدرس عملوند در ثبات آدرس حافظه قرار خواهد داشت.

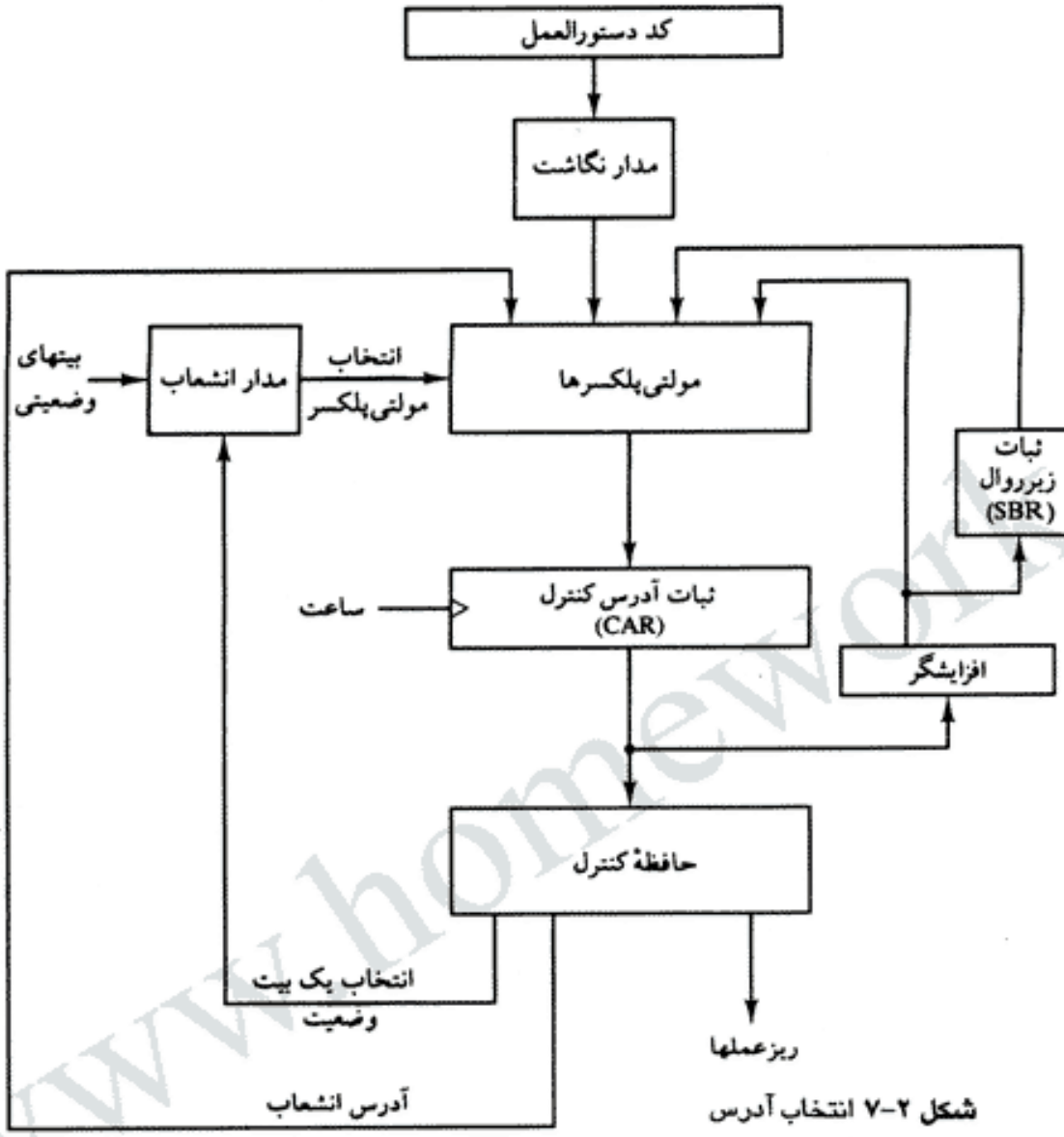
قدم بعدی تولید ریزاعمالی است که دستورالعمل برداشت شده از حافظه را اجرا می‌نمایند. مراحل ریزاعمالی که باید در ثبات‌های پردازنده تولیدشوند به کد عمل که بخشی از دستورالعمل است بستگی دارد. به هر دستورالعمل روال ریزبرنامه خاصی تعلق می‌گیرد که در مکان معینی از حافظه کنترل ذخیره شده است. تبدیل بیت‌های کد دستورالعمل به یک آدرس در حافظه کنترل که روال در آن قرار دارد نگاشت[1] نامیده می‌شود. رویه نگاشت قاعده‌ای است که کد دستورالعمل را به آدرس حافظه کنترل تبدیل می‌کند. هنگامی که به روال مورد نظر دسترسی پیدا شد، ریزدستورالعمل‌هایی که دستورالعمل را اجرا می‌کند می‌توانند با افزایش ثبات آدرس کنترل طی شوند، اما گاهی هم توالی ریزدستورالعمل‌ها به مقادیر معینی از بیت‌های وضعیت در ثبات‌های پردازشگر بستگی دارد. ریزبرنامه‌هایی که از زیرروال‌ها استفاده می‌کنند به یک ثبات خارجی برای نگهداری آدرس بازگشت نیاز دارند. آدرس‌های بازگشت نمی‌توانند در ROM ذخیره شوند زیرا ROM دارای قابلیت نوشتن نیست.

هنگامی که اجرای دستورالعملی پایان یابد، کنترل باید به روال برداشت (واکشی) بازگردد. این عمل با اجرای یک ریزدستورالعمل انشعاب غیرشرطی به ابتدای روال برداشت تحقق می‌یابد. بطور خلاصه، قابلیت‌های لازم برای توالی آدرس در حافظه کنترل عبارتند از:

۱- افزایش ثبات آدرس کنترل
۲- انشعاب غیرشرطی و یا شرطی، بسته به شرایط بیت‌های وضعیت
۳- فرآیند نگاشت (تبدیل) از بیت‌های دستورالعمل به آدرس حافظه کنترل
۴- امکاناتی برای فراخوانی زیرروال و بازگشت از آن

شکل ۲-۷ بلاک دیاگرام یک حافظه کنترل و سخت‌افزار لازم برای انتخاب آدرس ریزدستورالعمل بعدی را نشان می‌دهد. ریزدستورالعمل واقع در حافظه کنترل متشکل از مجموعه‌ای از بیت‌هاست که ریزاعمال را در ثبات‌های کامپیوتر موجب می‌شود و نیز مجموعه‌ای از بیت‌های دیگر برای تهیه آدرس بعدی است. دیاگرام چهار مسیری که توسط آنها ثبات آدرس کنترل[2] (CAR) آدرس را دریافت می‌نماید، نشان می‌دهد. افزایشگر محتوای ثبات آدرس کنترل را یک واحد افزایش می‌دهد تا ریزدستورالعمل بعدی را انتخاب نماید. عمل انشعاب پس از مشخص شدن آدرس انشعاب در یکی از میدان‌های ریزدستورالعمل انجام می‌شود. انشعاب شرطی با بکارگیری بخشی از ریزدستورالعمل بخاطر انتخاب بیت وضعیت معینی برای مشخص نمودن شرایط انشعاب، صورت می‌گیرد. آدرس خارجی (که بخشی از دستورالعمل است) از طریق مدار منطقی نگاشت به کنترل حافظه منتقل می‌شود. آدرس برگشت

1- Mapping 2- Control Address Register



شکل ۲-۷ انتخاب آدرس برای حافظه کنترل

برای زیر روال در ثبات خاصی‌که مقدارش بعداً بهنگام بازگشت مورد استفاده قرار می‌گیرد ذخیره می‌شود.

انشعاب شرطی

منطق انشعاب شکل ۲-۷ قابلیت تصمیم‌گیری را در واحد کنترل فراهم می‌آورد. حالت وضعیت سیستم، بیت‌های خاصی هستند که اطلاعات پارامتری همچون، نقلی خروجی یک جمع کننده، بیت علامت یک عدد، بیت‌های روش[1] (شیوه) در یک دستورالعمل، شرایط وضعیتی ورودی یا

1- Mode bit

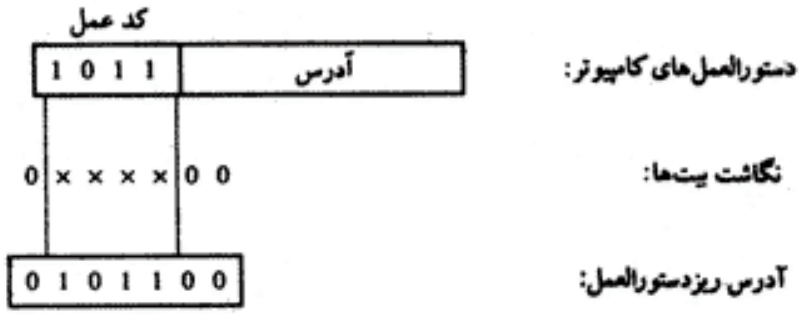
خروجی[1] را فراهم می‌کنند. اطلاعات موجود در این بیت‌ها قابل تست بوده و تصمیم اتخاذ شده می‌تواند بر اساس حالت آنها یعنی 1 یا 0 باشد. بیت‌های وضعیت، همراه با میدانی در ریزدستورالعمل که آدرس انشعاب را مشخص می‌نماید، تصمیمات انشعاب‌های شرطی راکه در مدار انشعاب تولید می‌شود کنترل می‌نمایند.

سخت‌افزار مدار انشعاب بطرق مختلفی قابل پیاده‌سازی است. در ساده‌ترین راه، شرط مشخص شده و انشعاب به آدرس معین شده با برقرار بودن شرط صورت می‌گیرد، در غیر اینصورت ثبات آدرس افزایش می‌یابد. این مدار می‌تواند با یک مولتی‌پلکسر پیاده‌سازی شود. فرض کنید که بیت‌های وضعیت جمعاً بتوانند هشت حالت داشته باشند، از سه بیت در ریزدستورالعمل می‌توان برای مشخص کردن هر یک از هشت حالت وضعیت استفاده کرد. این سه بیت متغیرهای انتخاب را برای مولتی‌پلکسر تشکیل می‌دهند. اگر بیت وضعیت انتخاب شده در حالت 1 باشد، خروجی مولتی‌پلکسر 1 است؛ در غیر اینصورت 0 است. خروجی 1 در مولتی‌پلکسر یک سیگنال کنترل برای انتقال آدرس انشعاب از ریزدستورالعمل به ثبات آدرس کنترل تولید می‌کند. خروجی 0 در مولتی‌پلکسر موجب افزایش ثبات آدرس می‌گردد. در این پیکربندی ریزبرنامه یکی از دو مسیر ممکن را بسته به مقدار بیت وضعیت دنبال خواهد کرد.

ریزدستورالعمل انشعاب غیرشرطی را می‌توان با بارکردن آدرس انشعاب از حافظه کنترل به ثبات آدرس کنترل ایجاد کرد. این کار را می‌توان با ثابت نگهداشتن مقدار یک بیت وضعیت در ورودی مولتی‌پلکسر در منطق 1 انجام داد. ارجاع به این بیت توسط خطوط انتخاب از حافظه کنترل موجب می‌شود تا آدرس انشعاب در ثبات آدرس کنترل بدون هر شرطی بار شود.

نگاشت[2] دستورالعمل

نوع خاصی از انشعاب هنگامی است که یک ریزدستورالعمل حکم کند که به اولین کلمه در حافظه کنترل که روال ریزبرنامه مربوط به یک دستورالعمل در آنجا واقع است انشعاب شود. بیت‌های وضعیت برای این نوع انشعاب، بیت‌های بخش کد عملیات در دستورالعمل هستند. مثلاً، کامپیوتری با قالب



شکل ۳-۷ نگاشت (تبدیل) کد دستورالعمل به آدرس ریزدستورالعمل

1- Input or output state Condition
2- Mapping

ساده‌ای برای دستورالعمل، مطابق شکل ۳-۷، کد عمل چهار بیتی دارد که می‌تواند تا 16 دستورالعمل متمایز را تشخیص دهد. فرض کنید حافظه کنترل 128کلمه داشته باشد، در نتیجه به آدرس هفت بیتی نیاز خواهد داشت. برای هر کد عملیات یک روال ریزبرنامه در حافظه وجود دارد که دستورالعمل را اجرا می‌کند. نمونه ساده‌ای از فرآیند نگاشت که کد عملیات 4 بیتی را به آدرس 7 بیتی برای حافظه کنترل تبدیل می‌کند در شکل ۳-۷ نشان داده شده است. این نگاشت بدین صورت انجام می‌شود که یک 0 در پرارزشترین مکان آدرس قرار داده می‌شود، چهار بیت کد عملیات منتقل می‌گردد، و دو بیت کم‌ارزشتر ثبات آدرس کنترل پاک می‌شوند. لذا برای هر دستورالعمل کامپیوتر می‌توان یک روال ریزبرنامه با ظرفیت چهار ریزدستورالعمل داشت. اگر روال به بیش از چهار ریزدستورالعمل نیاز داشته باشد، می‌توان از آدرس‌های 1000000 تا 1111111 استفاده کرد و اگر کمتر از چهار ریزعمل بکار رود، از مکان‌های حافظه بلااستفاده می‌توان برای روال‌های دیگر استفاده کرد.

می‌توان با استفاده از ROM در مشخص کردن تابع نگاشت، مفهوم نگاشت را تعمیم داد. در این آرایش، بیت‌های دستورالعمل آدرس ROM، نگاشت را مشخص می‌کنند. محتوای ROM نگاشت بیت‌های ثبات آدرس کنترل را تشکیل می‌دهند. به این ترتیب روال ریزبرنامه‌ای که دستورالعمل را اجرا می‌کند در هر جای مطلوبی در حافظه کنترل می‌تواند قرارگیرد. روش نگاشت انعطاف لازم را برای افزایش دستورالعمل‌ها، بعلت بالارفتن تقاضا فراهم می‌نماید.

گاهی تابع نگاشت با استفاده از مدار مجتمعی به نام قطعه منطقی برنامه پذیر[1] یا PLD نیز پیاده‌سازی می‌شود. از نظر مفهوم PLD شبیه ROM است با این تفاوت که از گیت‌های AND و OR با فیوزهای الکترونیکی داخلی استفاده می‌کند. اتصالات بین ورودیها، گیتهای AND، گیت‌های OR، و خروجیها را می‌توان مانند ROM برنامه‌نویسی کرد. هر تابع نگاشتی که با عبارات بولی بیان شود به آسانی با یک PLD تحقق می‌یابد.

زیر روال‌ها[2]

زیر روال‌ها برنامه‌هایی هستند که بوسیله سایر روال‌ها برای انجام کار خاصی مورد استفاده قرار می‌گیرند. زیرروال را می‌توان از هر نقطه‌ای در متن اصلی ریزبرنامه فراخوانی کرد. غالباً بسیاری از ریزبرنامه‌ها حاوی بخش‌های یکسانی از کد هستند. می‌توان ریزدستورالعمل را با بکارگیری زیرروال‌هایی که بخش‌های مشترکی از ریزکدها دارند کاهش داد. مثلاً، رشته ریزاعمال لازم برای تولید آدرس موثر عملوند یک دستورالعمل برای تمام دستورالعملهای ارجاع به حافظه مشترک است. این رشته می‌تواند زیرروالی باشد که از درون روال‌های متعدد دیگر برای اجرای محاسبات آدرس موثر فراخوانی شود.

ریزبرنامه‌هایی که از زیرروال استفاده می‌کنند باید امکاناتی برای ذخیره آدرس بازگشت بهنگام فراخوانی، و بازیابی آدرس بهنگام برگشت از زیرروال داشته باشد. این کار را می‌توان با قرار دادن

1- Programmable Logic Device

2- Subroutine

خروجی افزایش یافته از ثبات آدرس کنترل در ثبات زیرروال و انشعاب به ابتدای زیرروال انجام داد. ثبات زیرروال سپس می‌تواند بعنوان منبعی جهت انتقال آدرس بازگشت به روال اصلی مورد استفاده قرار گیرد. بهترین راه ایجاد مجموعه‌ای از ثباتها برای ذخیره آدرس‌های زیرروال‌ها، سازماندهی ثبات‌ها به شکل یک پشته[1] از نوع آخرین ورودی- اولین خروجی[2] (LIFO) می‌باشد. استفاده از پشته در یک زیر روال فراخوانی و بازگشت از زیرروال‌ها در بخش ۸-۷ مفصلاً شرح داده شده است.

۳-۷ مثال ریزبرنامه

وقتی که آرایش یک کامپیوتر و واحد کنترل ریزبرنامه‌نویسی شده آن مسجل شد، وظیفه طراح، تولید ریز کد برای حافظه کنترل است. این تولید کد را ریز برنامه‌نویسی می‌نامند که مشابه برنامه‌نویسی معمولی بزبان ماشین است. برای درک بهتر این فرآیند، در اینجا یک کامپیوتر دیجیتال ساده را ارائه داده و نشان می‌دهیم. کامپیوتر مورد بحث در این مثال مشابه کامپیوتر پایه معرفی شده در فصل ۵ است ولی دقیقاً همانند آن نیست.

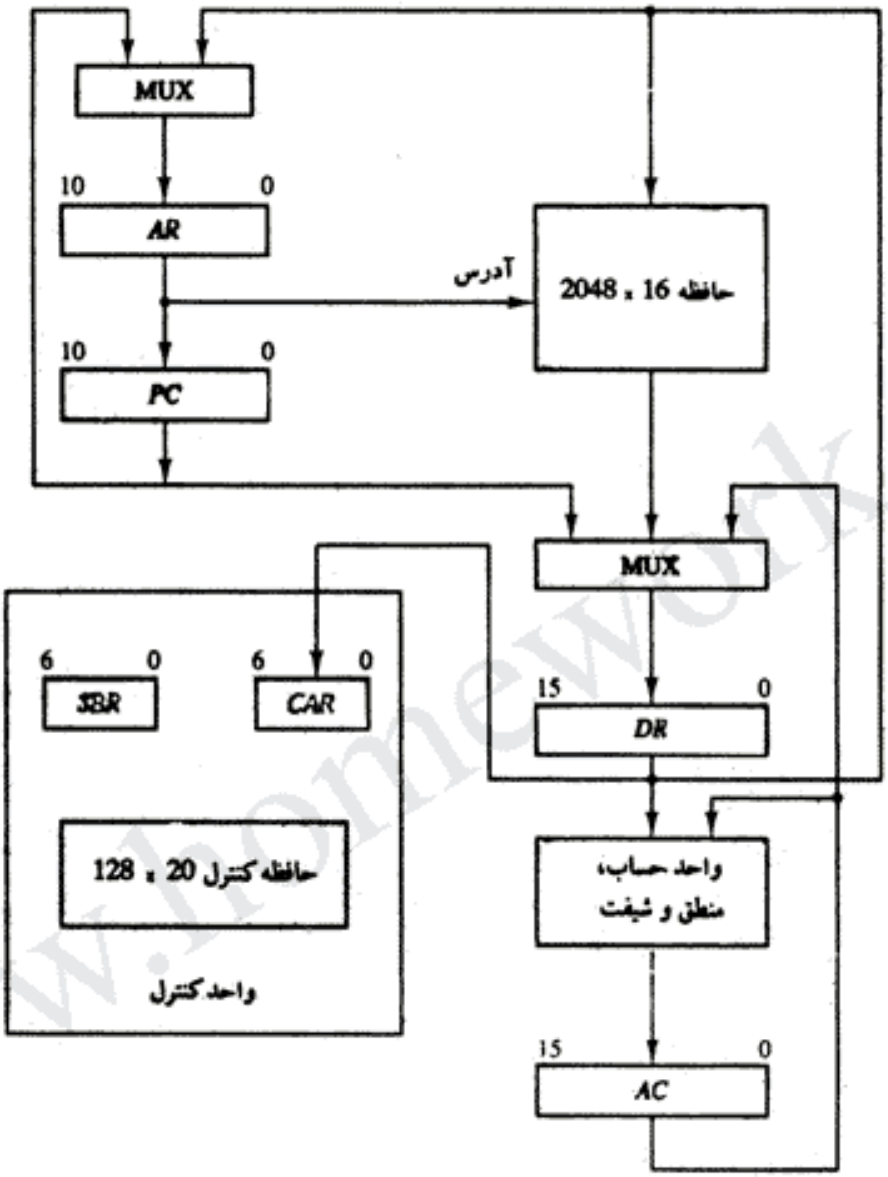
آرایش کامپیوتر

بلاک دیاگرام کامپیوتر مورد نظر در شکل ۴-۷ نشان داده شده است. این کامپیوتر از دو واحد حافظه تشکیل شده است: یکی حافظه اصلی برای ذخیره دستورالعمل‌ها و داده‌ها، و دیگری حافظه کنترل برای ذخیره ریزبرنامه. چهار ثبات به واحد پردازنده و دو ثبات به واحد کنترل اختصاص یافته‌اند. ثبات‌های پردازنده عبارتند از شمارنده برنامه[3]، PC، ثبات آدرس AR، ثبات داده DR و ثبات انباره AC. کار این ثبات‌ها مشابه کامپیوتر پایه‌ای است که در فصل ۵ معرفی شد (شکل ۳-۵ ملاحظه شود). واحد کنترل دارای ثبات آدرس کنترل CAR و یک ثبات زیرروال SBR است. حافظه کنترل و ثبات‌های متعلق به آن، همان‌طور که در شکل ۲-۷ دیده می‌شود، بصورت یک واحد کنترل ریزبرنامه‌نویسی شده شناخته می‌شوند.

انتقال اطلاعات بین ثبات‌های پردازنده از طریق مولتی‌پلکسر صورت می‌گیرد و نه با استفاده از گذرگاه مشترک. DR می‌تواند اطلاعات را از AC، PC و یا حافظه دریافت کند. PC فقط از AR اطلاعات دریافت می‌کند. واحد حساب، منطق و شیفت ریزاعمال را برروی داده‌های دریافتی از AC و DR انجام داده و حاصل را در AC قرار می‌دهد. دقت کنید که حافظه آدرس خود را از AR دریافت می‌کند. داده ورودی که در حافظه نوشته می‌شود از طریق DR وارد، و داده خوانده شده از حافظه هم فقط به DR می‌رود.

قالب دستورالعمل‌های کامپیوتر در شکل ۵-۷ (الف) نشان داده شده است. این قالب از سه میدان تشکیل شده است: یک میدان 1 بیتی برای آدرس‌دهی غیرمستقیم که با I مشخص شده است، یک کد

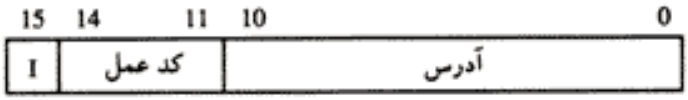
1- stack 2- Last In First Out 3- Program Counter



شکل ۴-۷ پیکربندی سخت‌افزار کامپیوتر

عملیات[1] 4 بیتی (opcode) و یک میدان آدرس 11 بیتی. شکل ۵-۷ (ب) چهار دستور از شانزده دستورالعمل‌های ارجاع به حافظه را نشان می‌دهد. دستورالعمل ADD، عملوند موجود در آدرس موثر را با محتوای AC جمع می‌کند. دستورالعمل BRANCH، انشعابی را به آدرس موثر موجب می‌شود بشرطی که عملوند درون AC منفی باشد. اگر AC منفی نباشد برنامه با دستورالعمل بعدی ادامه می‌یابد. AC در صورتی منفی است که بیت علامت (بیت منتهاالیه سمت چپ ثبات) 1 باشد. دستورالعمل STORE محتوای AC را به کلمه‌ای از حافظه که آدرس موثر آن را مشخص می‌کند انتقال

1- Operation Code



(الف) قالب دستورالعمل

سمبل	کد عمل	توضیح
ADD	0000	$AC \leftarrow AC+M[EA]$
BRANCH	0001	If $(AC<0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA]$, $M[EA] \leftarrow AC$

EA آدرس مؤثر است

(ب) چهار دستورالعمل کامپیوتر

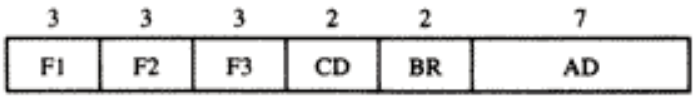
شکل ۵-۷ دستورالعمل‌های کامپیوتر

می‌دهد. دستورالعمل EXCHANGE داده‌هایی را بین AC و کلمه حافظه مشخص شده بوسیله آدرس موثر تعویض می‌کند.

در دنباله بحث نشان داده خواهد شد که هر دستورالعمل کامپیوتر بایستی ریزبرنامه‌نویسی شود. بخاطر پرهیز از پیچیده شدن مثال ریز برنامه‌نویسی، در اینجا فقط چهار دستورالعمل مورد بررسی قرار گرفته است. باید دانست که 12 دستورالعمل دیگر می‌تواند به مجموعه اضافه شود و هر دستورالعمل باید توسط روند شرح داده شده در ذیل ریز برنامه نویسی گردد.

قالب ریزدستورالعمل‌ها

قالب ریزدستورالعمل‌ها برای حافظه کنترل در شکل ۶-۷ نشان داده شده است. 20 بیت هر دستورالعمل به چهار بخش عملیاتی تقسیم شده است. سه میدان F1 ، F2 و F3 ریزاعمال را برای کامپیوتر مشخص می‌کنند. میدان CD بیت‌های شرطی وضعیت را انتخاب می‌نماید. میدان BR نوع



F1، F2، F3 :میدان‌های ریزعمل

CD : شرط انشعاب

BR : میدان انشعاب

AD : میدان آدرس

شکل ۶-۷ قالب کد ریزدستورالعمل (20 بیت)

انشعاب بکار رفته را معین می‌کند. میدان AD آدرس انشعاب را داراست. میدان آدرس هفت بیت عرض دارد، زیرا حافظه کنترل دارای $2^7 = 128$ کلمه است.

گفتیم که ریزاعمال به سه میدان جزیی تقسیم شده‌اند که هر یک سه بیتی هستند. هر سه بیت در هر میدان برای مشخص کردن هفت ریزعمل مستقل، مطابق جدول ۱-۷، انکد شده‌اند. لذا در مجموع 21 ریزعمل وجود خواهد داشت. برای هر ریزدستور نمی‌توان بیش از سه ریزعمل انتخاب کرد، یعنی هر یک برای یک میدان. اگر کمتر از سه ریزعمل بکار رود، یک یا چند میدان از کد 000 بمعنی هیچکار[1] استفاده خواهند کرد. مثلاً یک ریزدستورالعمل می‌تواند بطور همزمان دو ریزعمل را در F2 و F3 و هیچ عمل را در F1 انجام دهد.

$$DR \leftarrow M[AR] \qquad \text{با } F2 = 100$$

و

$$PC \leftarrow PC + 1 \qquad \text{با } F3 = 101$$

9 بیت میدان ریز عملیات عبارت از 101 100 000 خواهند بود. توجه به این نکته که نمی‌توان دو یا چند عمل متضاد را به طور همزمان اجرا کرد اهمیت دارد. مثلاً میدان ریز اعمال 000 001 010 مفهومی ندارد زیرا این ریز اعمال، عملیاتی مانند پاک کردن AC و تفریق DR از AC را بطور همزمان نشان می‌دهد.

هر ریز عمل در جدول ۱-۷ بوسیله یک عبارت انتقال ثبات تعریف شده و سمبلی جهت استفاده در برنامه‌های سمبلیک به آن اختصاص داده شده است. همه سمبل‌های ریزعمل‌های انتقالی از پنج حرف استفاده می‌کنند. دو حرف اول ثبات مبدا را مشخص می‌کند، سومین حرف همواره T است، و دو حرف آخر ثبات مقصد را معین می‌نماید.

میدان شرط[2] CD دو بیتی است که برای تعیین چهار بیت وضعیت شرطی طبق جدول ۱-۷ انکد شده است. اولین حالت همیشه 1 است، ارجاع به CD = 00 (یا سمبل U) همیشه با شرط درست مواجه می‌شود. اگر این حالت همراه با میدان BR (انشعاب) بکار رود یک عمل انشعاب بدون شرط حاصل می‌شود. بیت غیرمستقیم I را پس از خواندن یک دستورالعمل از حافظه می‌توان در بیت 15 ثبات DR بدست آورد. بیت علامت AC بیت وضعیت بعدی را تشکیل می‌دهد. مقدار صفر، که با Z مشخص شده است، یک متغیر دودویی است که اگر تمام بیت‌ها در AC صفر شود، این متغیر 1 خواهد شد. ما هنگامی که ریزبرنامه‌هایی را به فرم سمبلیک بنویسیم از چهار سمبل U، I، S، Z استفاده خواهیم کرد.

میدان BR دو بیتی است. این میدان به همراه میدان آدرس AD برای انتخاب آدرس ریزدستور بعدی بکار می‌رود. همانطور که در جدول ۱-۷ ملاحظه می‌شود، وقتی BR = 00 باشد، کنترل عمل پرش (JMP) را اجرا می‌کند (که مشابه عمل انشعاب است)، و هنگامی که BR = 01 باشد عمل فراخوانی زیرروال (CALL) انجام خواهد شد. دو عمل یکسان هستند با این تفاوت که ریزعمل فراخوانی آدرس

1- No operation 2- Condition Field

جدول ۱-۷ سمبل‌ها و کدهای دودویی برای میدانهای ریزدستورالعمل‌ها

F1	ریزعمل	سمبل
000	هیچکار	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	ریزعمل	سمبل
000	هیچکار	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	ریزعمل	سمبل
000	هیچکار	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

CD	شرط	سمبل	توضیح
00	همیشه = 1	U	انشعاب غیرشرطی
01	$DR(15)$	I	بیت آدرس غیرمستقیم
10	$AC(15)$	S	بیت علامت AC
11	$AC = 0$	Z	مقدار صفر در AC

BR	سمبل	عملکرد
00	JMP	$CAR \leftarrow AD$ اگر شرط برابر 1 باشد $CAR \leftarrow CAR + 1$ اگر شرط برابر صفر باشد
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ اگر شرط برابر 1 باشد $CAR \leftarrow CAR + 1$ اگرشرط برابر 0 باشد
10	RET	$CAR \leftarrow SBR$ بازگشت از زیرروال tine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

بازگشت را در ثبات زیرروال SBR ذخیره می‌کند. اعمال پرش و فراخوانی به میدان CD وابسته هستند. اگر شرایط بیت وضعیت در میدان CD برابر 1 باشد، آدرس بعدی واقع در میدان AD به ثبات آدرس کنترل CAR منتقل می‌گردد. در غیراینصورت CAR یک واحد اضافه می‌شود.

بازگشت از زیرروال در صورتی انجام می‌شود که میدان BR برابر 10 باشد. این وضعیت موجب می‌شود آدرس بازگشت از SBR به CAR منتقل شود. نگاشت بیت‌های کد عملیات دستورالعمل به آدرسی برای CAR در صورتی انجام می‌شود که میدان BR برابر 11 باشد. این نگاشت در شکل ۳-۷ نشان داده شده است. بیت‌های کد عملیات پس از خواندن یک دستورالعمل از حافظه در DR(11-14) قرار دارد. توجه کنید که دو حالت آخر در میدان BR مستقل از مقادیر میدان‌های CD و AD است.

ریز دستورالعمل‌های سمبلیک

سمبل‌های ارائه شده در جدول ۱-۷ می‌توانند برای مشخص کردن ریزدستورالعمل‌ها به فرم سمبلیک بکارروند. یک ریزبرنامه سمبلیک می‌تواند بکمک اسمبلر به معادل دودویی‌اش تبدیل شود. اسمبلر یک ریزبرنامه از نظر مفهوم مشابه اسمبلر معمولی در کامپیوتر است که در بخش ۳-۶ تعریف شد. ساده‌ترین و سرراست‌ترین روش تنظیم یک زبان اسمبلی برای یک ریزبرنامه، تعریف سمبل‌ها برای هر میدان ریز دستورالعمل و امکان تعریف آدرس سمبلیک توسط خود کاربران است.

هر سطر از ریزبرنامه زبان اسمبلی، یک ریزدستورالعمل سمبلیک را تعریف می‌کند. هر دستورالعمل سمبلیک به پنج میدان تقسیم می‌شود: عنوان، ریزاعمال، CD، BR، AD. این میدان‌ها اطلاعات زیر را مشخص می‌نمایند.

۱- میدان عنوان می‌تواند خالی و یا یک آدرس سمبلیک باشد. هر عنوان با یک علامت نقل‌قول (:) خاتمه می‌یابد.

۲- میدان ریزعمل‌ها از یک، دو و یا سه سمبل تعریف شده در جدول ۱-۷ تعریف می‌شود و بوسیله کاما از هم جدا می‌شوند. نمی‌توان بیش از یک سمبل از هر میدان F داشت. هرگاه ریزدستورالعمل هیچ عملی نداشته باشد از سمبل NOP[1] استفاده می‌شود. اسمبلر این سمبل را به 9 عدد 0 ترجمه می‌کند.

۳- میدان CD دارای یک حرف U، I، S و یا Z است.

۴- میدان BR حاوی یکی از چهار سمبل تعریف شده در جدول ۱-۷ است.

۵- میدان AD مقدار میدان آدرس ریز دستورالعمل را به یکی از سه طریق زیر مشخص می‌کند.

(الف) با یک آدرس سمبلیک، که باید بصورت عنوان هم وجود داشته باشد.

(ب) با سمبل NEXT برای مشخص کردن آدرس متوالی بعدی

(ج) اگر میدان BR حاوی سمبل RET، یا MAP باشد، میدان AD خالی گذاشته می‌شود و اسمبلر آن را به هفت عدد صفر تبدیل می‌نماید.

1- No Operation

ما از شبه دستور ORG نیز برای تعریف مبداء، یا اولین آدرس یک روال ریزبرنامه استفاده می‌کنیم. مثلاً سمبل ORG 64 به اسمبلر اطلاع می‌دهد که ریزدستورالعمل بعدی را در حافظه کنترل در آدرس‌دهی 64 قرار دهد، که معادل آدرس دودویی 000 1000 است.

روال برداشت[1]

حافظه کنترل دارای 128 کلمه است، و هر کلمه 20 بیت دارد. برای ریزبرنامه‌نویسی حافظه کنترل، لازم است تا مقادیر بیتی هر 128 کلمه را معین کنیم. 64 کلمه اول (آدرس 0 تا 63) بوسیله روال‌های 16 دستورالعمل اشغال می‌شوند. 64 کلمه آخر را می‌توان برای هر هدف دیگری بکار برد. آدرس 64 شروع مناسبی برای روال برداشت (یا واکشی) است. ریزدستورالعمل‌های لازم برای این روال عبارتند از:

AR ← PC
DR ← M[AR] , PC ← PC + 1
AR ← DR (0 − 10) , CAR (2 − 5) ← DR (11 − 14) , CAR (0, 1, 6) ← 0

آدرس دستورالعمل از PC به AR منتقل شده و سپس دستورالعمل از حافظه به DR خوانده می‌شود. چون هیچ ثبات دستورالعملی وجود ندارد، کد دستورالعمل در DR باقی می‌ماند. بخش آدرس به AR منتقل و سپس کنترل بوسیله نگاشت بخش کد عملیات دستورالعمل از DR به CAR به یکی از 16 روال انتقال می‌یابد.

روال برداشت به سه ریزدستورالعمل نیاز دارد، که در آدرس 64 ، 65 و 66 حافظه کنترل قرار داده می‌شوند. با استفاده از قواعد زبان اسمبلی که قبلاً تعریف شد، ما می‌توانیم ریزبرنامه سمبلیک را برای روال برداشت بنویسیم:

```
        ORG 64
FETCH:  PCTAR          U    JMP    NEXT
        READ, INCPC    U    JMP    NEXT
        DRTAR          U    MAP
```

ترجمه ریزبرنامه سمبلیک به دودویی، ریزبرنامه دودویی زیر را تولید می‌کند. مقادیر بیت‌ها از جدول ۱-۷ بدست می‌آیند.

آدرس دودویی	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

1- Fetch Roution

سه ریز دستورالعمل که روال برداشت را تشکیل می‌دهند به سه فرم مختلف نشان داده شدند. نمایش انتقال ثباتی عمل انتقال بین ثبات‌ها را که هر ریزدستورالعمل پیاده می‌کند نشان می‌دهد. نمایش سمبلیک برای نوشتن ریزبرنامه در قالب زبان اسمبلی مفید است. نمایش دودویی محتوای داخلی واقعی را که باید در حافظه کنترل ذخیره شود نشان می‌دهد. مرسوم است که ریزبرنامه‌ها را به شکل سمبلیک بنویسند و سپس با استفاده از یک برنامه اسمبلر ترجمه دودویی آن را بدست آورند.

ریزبرنامه سمبلیک

اجرای سومین ریزدستورالعمل در روال برداشت (MAP) سبب می‌شود به آدرس 0xxxx00 انشعاب شود که در آن xxxx چهار بیت کد عملیات است. مثلاً اگر دستورالعمل، ADD باشد که کد عمل آن 0000 است، ریزدستورالعمل MAP آدرس 0000000 که آدرس شروع روال ADD در حافظه کنترل می‌باشد را به CAR منتقل می‌سازد. اولین آدرس برای روال‌های BRANCH و STORE بترتیب عبارتند از 00 0001 0 (دهدهی 4) و 00 0010 0 (دهدهی 8). اولین آدرس سیزده روال دیگر در آدرس‌های 12، 16 ، 20 ، ... و 60 می‌باشند. باین ترتیب چهار کلمه در حافظه کنترل به هر روال اختصاص می‌یابد.

در هر روال باید ریزدستورالعمل‌هایی برای محاسبه آدرس موثر و اجرای دستورالعمل در نظر بگیریم. روش آدرس‌دهی غیرمستقیم در همه دستورالعمل‌های حافظه‌ای می‌تواند بکار رود. با ذخیره دستورالعمل‌های مربوط به آدرس‌دهی غیرمستقیم به صورت یک روال، می‌توان در تعداد کلمه‌های حافظه کنترل صرفه‌جویی کرد. این زیرروال، که با INDRCT نشان داده شده، درست پس از روال برداشت قرار دارد، جدول ۲-۷. این جدول ریزبرنامه سمبلیک را برای روال برداشت و روال‌های ریزدستورالعملی که چهار دستورالعمل کامپیوتر اجرا می‌کنند را نیز نشان می‌دهد.

برای اینکه ببینیم چگونه انتقال و برگشت از زیرروال غیرمستقیم INDRCT رخ می‌دهد، فرض کنید که ریزدستورالعمل MAP در انتهای روال برداشت موجب انشعاب به آدرس 0 ، یعنی مکانی که روال ADD در آن قرار دارد، شود. اولین ریزدستورالعمل در روال ADD زیرروال INDRCT را با شرط بیت وضعیت I ، فرامی‌خواند. اگر I = 1 باشد انشعاب به INDRCT رخ داده و آدرس برگشت (آدرس 1 در این حالت) در ثبات زیر روال SBR ذخیره می‌شود. زیرروال INDRCT دو ریزدستورالعمل دارد.

INDRCT : READ U JUMP NEXT
DRTAR U RET

بخاطر آورید که یک آدرس غیرمستقیم، بخش آدرس دستورالعمل را بعنوان آدرسی که آدرس موثر در آن قرار دارد تلقی می‌کند و آنرا عملوند تصور نمی‌نماید. بنابراین حافظه برای بدست آوردن آدرس موثر باید دستیابی شود و سپس به AR منتقل گردد. برگشت از زیرروال (RET) آدرس SBR را به CAR انتقال می‌دهد، و لذا به دومین ریزدستورالعمل روال ADD بازمی‌گردد.

جدول ۲-۷ بخشی از برنامه سمبلیک

عنوان	ریزعمل‌های	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

اجرای دستورالعمل ADD بوسیله ریزدستورالعمل واقع در آدرس‌های 1 و 2 انجام می‌گردد. اولین ریز دستورالعمل عملوند را از حافظه به DR می‌خواند. دومین ریز دستورالعمل ریزعمل جمع با محتوای DR و AC انجام داده و سپس به ابتدای روال برداشت بازمی‌گردد.

دستورالعمل BRANCH موجب انشعابی به آدرس موثر می‌گردد بشرطی که $AC < 0$ باشد. AC هنگامی کوچکتر از صفر است که بیت علامت در آن منفی باشد که این از 1 بودن بیت وضعیت S آشکار می‌گردد. روال BRANCH در جدول ۲-۷ با وارسی مقدار S آغاز می‌شود. اگر S برابر 0 باشد، انشعابی رخ نمی‌دهد و ریزدستورالعمل بعدی موجب پرش به عقب به روال برداشت، بدون تغییر محتوای PC، می‌گردد. اگر S برابر 1 باشد، اولین ریزدستورالعمل JMP کنترل را به مکان OVER منتقل می‌سازد. ریزدستورالعمل واقع در این مکان زیرروال INDRCT را، اگر $I = 1$ باشد، فرامی‌خواند. سپس آدرس

موثر از AR به PC منتقل و زیربرنامه به روال برداشت پرش می‌کند.

روال STORE مجدداً زیرروال INDRCT را، اگر I = 1 باشد، فرامی‌خواند. محتوای AC به DR منتقل می‌شود. یک عمل نوشتن در حافظه، برای ذخیره کردن DR در مکان مشخص شده بوسیله آدرس موثر در AR، اجرا می‌شود.

روال EXCHANGE عملوند را از آدرس موثر خوانده و آنرا در DR قرار می‌دهد. محتوای DR و AC در سومین دستورالعمل با یکدیگر تعویض می‌شوند. این تعویض هنگامی میسر است که ثباتها از نوع حساس به لبه[1] باشند (شکل ۲۳-۱). محتوای اصلی AC که حالا در DR است مجدداً در حافظه ذخیره می‌شود.

توجه کنید که جدول ۲-۷ بخشی از ریزبرنامه را داراست. تنها چهار دستور از 16 دستور کامپیوتر ریزبرنامه‌نویسی شدند. همچنین کلمات حافظه کنترل در مکان‌های 69 تا 127 مورد استفاده قرارنگرفتند. دستوراتی مانند ضرب، تقسیم که نیاز به رشته ریزاعمال طولانی‌تری دارند برای اجرا به بیش از چهار ریز دستورالعمل نیاز دارند. کلمات حافظه کنترل از 69 تا 127 می‌توانند برای این منظور بکار روند.

ریز برنامه دودویی

ریزبرنامه سمبلیک فرم مناسبی برای نوشتن ریزبرنامه‌هایی است که افراد می‌توانند بخوانند و بفهمند. ولی این روش ذخیره ریزبرنامه در حافظه نیست. برنامه سمبلیک باید با یکی از دو روش برنامه اسمبلی یا کاربر به دودویی ترجمه شود بشرطی که ریزبرنامه مانند این مثال ساده باشد.

فرم دودویی معادل ریزبرنامه در جدول ۳-۷ لیست شده است. آدرس‌های حافظه کنترل به هر دو صورت دهدهی و دودویی داده شده است. همانطور که در جدول ۱-۷ دیده شد، محتوای دودویی هر ریزدستورالعمل از سمبل‌ها و معادل‌های دودویی آنها بدست می‌آید.

دقت کنید که آدرس 3 در ریزبرنامه سمبلیک فاقد معادل است زیرا روال ADD فقط سه ریز دستورالعمل در مکان‌های 0، 1 و 2 دارد. روال بعدی از آدرس 4 شروع می‌شود. هرچند آدرس 3 بکارنرفته است ولی باید یک مقدار دودویی برای هر یک از کلمات حافظه کنترل تعیین شود. می‌توانیم مقدار تماماً 0 را در این کلمه قرار دهیم زیرا این مکان هیچوقت بکار نخواهد رفت. با این وجود اگر خطای پیش‌بینی نشده‌ای رخ دهد، یا اگر یک سیگنال پارازیت[2] (نویز) مقدار 3 را در CAR قرار دهد، پرش به آدرس 64 که ابتدای روال برداشت است عاقلانه است.

ریزبرنامه دودویی لیست شده در جدول ۳-۷ محتوای کلمات حافظه کنترل را مشخص می‌نماید. وقتی که از ROM برای حافظه کنترل استفاده شود، لیست دودویی ریزبرنامه جدول درستی را برای ساخت قطعه فراهم می‌آورد. این عمل ساخت یک فرآیند سخت‌افزاری است و با ایجاد ماسکی برای ROM که 1 ها و 0 ها را برای هر کلمه بوجود می‌آورد، تشکیل شده است. پس از آنکه اتصالات درونی در حین ساخت سوزانده شدند بیت‌های ROM تثبیت می‌گردند. ROM از بسته‌های مدار مجتمع، IC،

1- Edge– Triggered 2- Noise

جدول ۳-۷ بخشی از ریزبرنامه دودویی برای حافظه کنترل

ریزروال	آدرس		ریزدستورالعمل دودویی					
	دهدهی	دودویی	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

ساخته شده و در صورت لزوم می‌توان آنها را تعویض کرد. برای اصلاح مجموعه دستورالعمل‌های کامپیوتر، لازم است یک ریزبرنامه جدید تولید و ROM جدیدی ماسک گردد. قطعه قبلی را برداشته و جدید را به جای آن قرار می‌دهیم.

اگر بخواهیم حافظه کنترل قابل نوشتن باشد، ROM با RAM جایگزین می‌شود. مزیت بکارگیری RAM برای حافظه کنترل این است که ریزبرنامه بسادگی با نوشتن یک الگوی جدید از 1 ها و 0 ها بدون اعمال روش‌های سخت‌افزاری میسر است. قابلیت نوشتن در حافظه کنترل این انعطاف را ایجاد می‌کند که با تغییر ریزبرنامه تحت کنترل پردازنده مجموعه دستورالعمل‌های کامپیوتر را بطور پویا تغییر داد. با این وجود اکثر سیستم‌های ریزبرنامه نویسی شده از ROM برای حافظه کنترل استفاده می‌کنند زیرا از RAM ارزانتر و سریعتر است و نیز هر کاربری نمی‌تواند معماری سیستم را تغییر دهد.

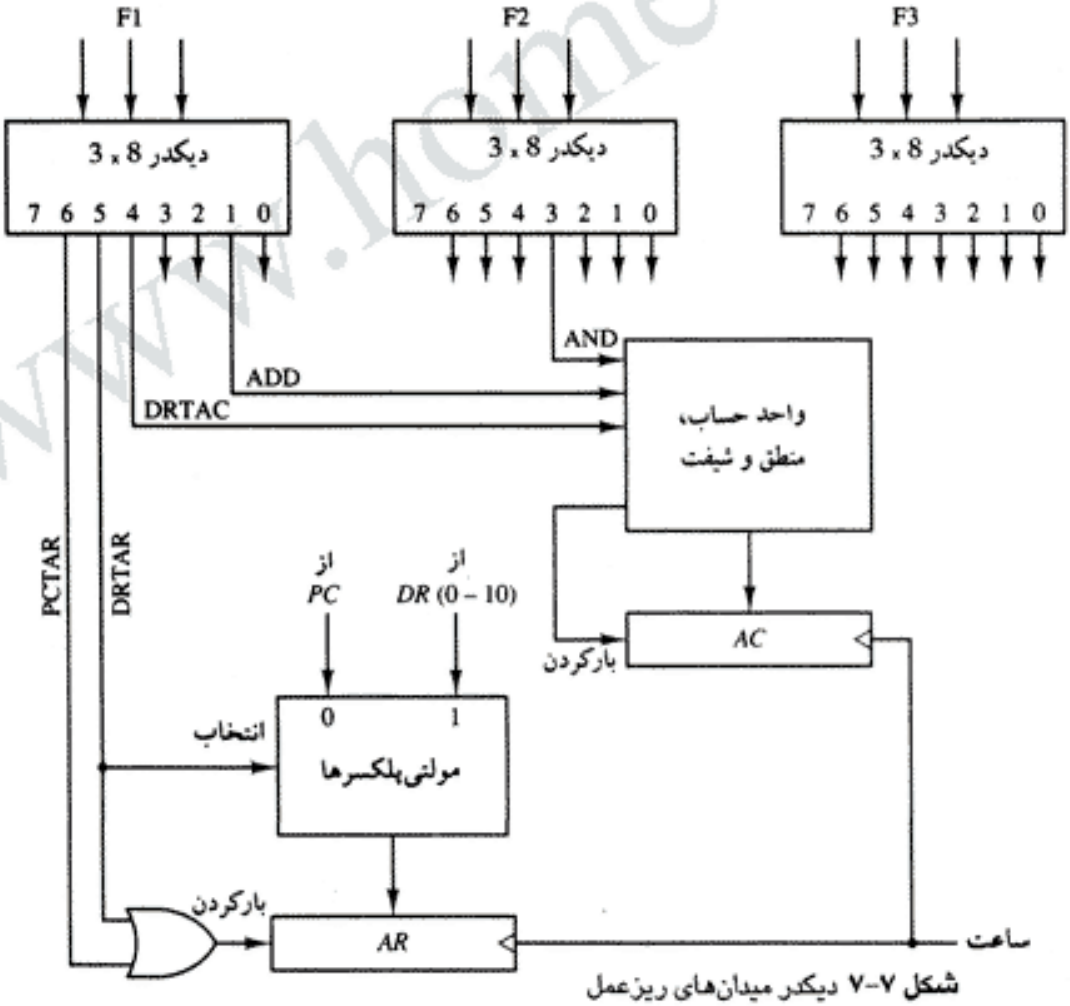
۷-۴ طراحی واحد کنترل

بیت‌های ریزدستورالعمل معمولاً به میدانی‌هایی تقسیم می‌شود، که هر میدان کار جداگانه و مستقلی را بعهده دارد. انواع میدان‌های مختلفی که در قالب‌های دستورالعمل‌ها باآنها مواجه می‌شویم،

بیت کنترلی لازم را برای انجام ریزعمل اعمال در سیستم، بیت‌های خاصی را برای مشخص کردن نحوه محاسبه آدرس بعدی، و میدان آدرسی را برای انشعاب، در اختیار می‌گذارند. تعداد بیت‌های کنترلی را که موجب اجرای ریزعمل‌ها می‌شوند می‌توان با دسته‌بندی متغیرهای جدا از هم به شکل میدان‌های مختلف و کدگذاری K بیت از هر میدان برای مشخص کردن 2^k ریزعمل، کاهش داد. هر میدان نیاز به یک دیکدر برای ایجاد سیگنال‌های کنترلی مربوطه دارد. این عمل اندازه بیت‌های ریزدستورالعمل را کاهش می‌دهد ولی مستلزم سخت‌افزار اضافی در خارج از حافظه کنترل است. روش فوق زمان تأخیر سیگنالهای کنترلی را نیز افزایش می‌دهد زیرا این سیگنالها باید از مدار دیکدر عبور نمایند.

انکد کردن بیت‌های کنترل در مثال برنامه‌نویسی بخش قبل نشان داده شد. 9 بیت میدان ریز عمل به سه میدان جزیی سه بیتی تقسیم شدند. خروجی حافظه کنترل هر میدان جزئی باید دیکد می‌شد تا ریز عمل‌های متمایز ایجاد می‌شدند. خروجی دیکدرها نیز به ورودی‌های مناسب در واحد پردازنده متصل می‌شدند.

شکل ۷-۷ سه دیکدر و بعضی از اتصالاتی که باید از خروجی‌هایشان ساخته شود را نشان می‌دهد. هر یک از سه میدان ریزدستورالعمل موجود در خروجی حافظه کنترل با یک دیکدر 8 × 3 دیکد می‌شود



شکل ۷-۷ دیکدر میدان‌های ریزعمل

تا هشت خروجی را فراهم آورد. هر یک از این خروجی‌ها بایستی به مدار مناسبی برای آغاز ریزعمل مربوطه، مطابق جدول ۱-۷، متصل شود. مثلاً وقتی که $F_1 = 101$ (5 دودویی) باشد، گذر لبه پالس ساعت بعدی محتوای DR(0 – 10) را به AR (که در جدول ۱-۷ با سمبل DRTAR نشان داده‌شده است) منتقل می‌نماید. بطور مشابه اگر $F_1 = 110$ (6 دودویی) باشد، انتقالی از PC به AR (که با سمبل PCTAR نشان داده شده است) صورت می‌گیرد. همانطورکه در شکل ۷-۷ دیده می‌شود، خروجی‌های 5 و 6 دیکدر F_1 به ورودی بارکردن AR وصل شده‌اند بطوری که وقتی هر یک از خروجی‌ها فعال گردد، اطلاعات از مولتی‌پلکسرها به AR منتقل می‌گردند. مولتی‌پلکسرها اطلاعات را، وقتی که خروجی 5 فعال است از DR، و وقتی این خروجی غیر فعال باشد از PC دریافت می‌نمایند. انتقال به AR، با گذر لبه پالس ساعت و هنگامی که خروجی 5 یا 6 دیکدر فعال باشد انجام می‌شود. سایر خروجی‌های دیکدر که انتقال بین ثبات‌ها را انجام می‌دهند باید به روش مشابهی مورد بررسی قرار گیرند.

واحد حساب، منطق، شیفت را می‌توان مطابق شکل‌های ۱۹-۵ و ۲۰-۵ طراحی کرد. به جای استفاده از گیت‌ها در تولید سیگنالهای کنترلی AND، ADD و DR در شکل ۱۹-۵، این ورودی‌ها بترتیب از خروجی دیکدرهای متناظر AND، ADD و DRTAC می‌آیند، شکل ۷-۷. سایر خروجی‌های دیکدرها که مربوط به عمل روی AC است باید به نحوی مشابه به واحد حساب، منطق، شیفت متصل گردد.

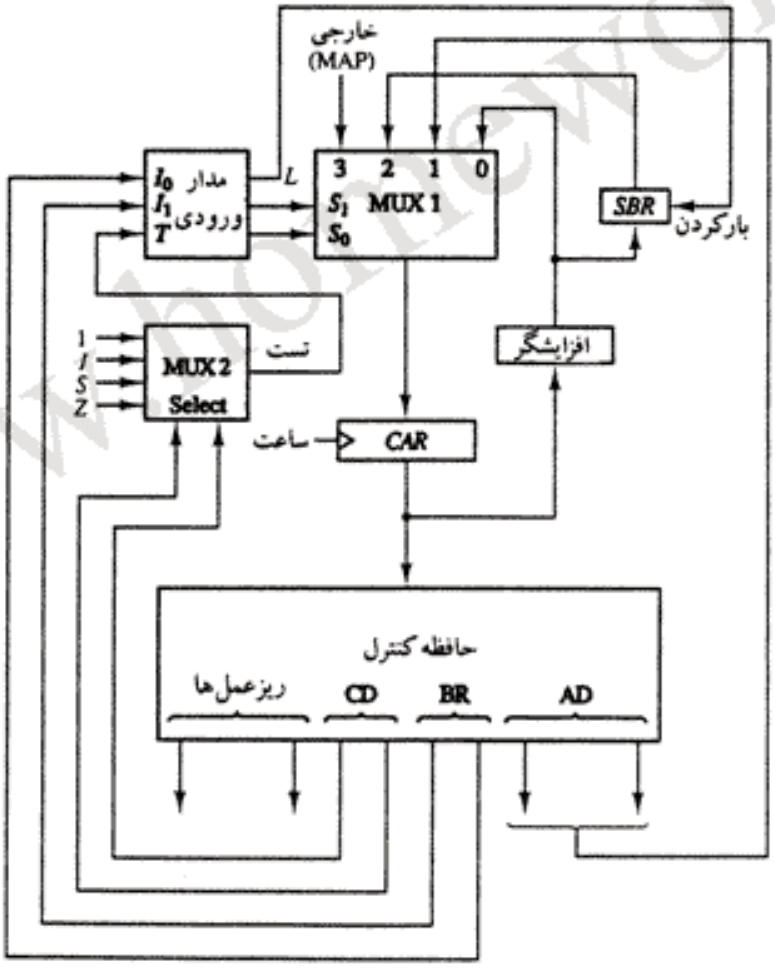
توالی‌گر ریزبرنامه

اجزاء اصلی واحد کنترل ریزبرنامه نویسی شده حافظه کنترل و مدارات انتخاب آدرس بعدی هستند. بخش انتخاب آدرس، توالی‌گر ریزبرنامه خوانده می‌شود. یک توالی‌گر ریزبرنامه را می‌توان با اجزاء دیجیتال، مناسب یک کاربرد خاص ساخت. با این وجود، همان‌طور که واحدهای ROM بزرگی به شکل بسته‌های مدار مجتمع وجود دارند، توالی‌گرهای همه منظوره هم وجود دارند که برای ساخت واحدهای کنترل ریزبرنامه‌نویسی شده مناسب‌اند. برای تضمین پذیرش وسیع، باید سازمان داخلی مدار مجتمع توالی‌گر بصورتی باشد که بتوان آن را در گستره وسیعی از کاربردها مورد استفاده قرار داد.

هدف از بکارگیری توالی‌گر ریزبرنامه، ارائه آدرس به حافظه کنترل است بطوری که یک ریزدستورالعمل از آن خوانده شده و اجرا گردد. مدار آدرس بعدی توالی‌گر، منبع آدرس بعدی، که در ثبات آدرس کنترل ذخیره خواهد شد را مشخص می‌کند. انتخاب منبع آدرس بوسیله بیت‌های اطلاعات آدرس بعدی که توالی‌گر آنها را از ریزدستورالعمل جاری دریافت می‌کند هدایت می‌شود. توالی‌گرهای تجاری در داخل خود دارای مجموعه‌ای از ثبات‌های داخلی هستند که برای ذخیره موقت آدرسها هنگام اجرای حلقه‌های ریزبرنامه و فراخوانی زیرروال‌ها بکار می‌روند. در بعضی از توالی‌گر یک ثبات خروجی وجود دارد که می‌توان از آن به عنوان ثبات آدرس برای حافظه کنترل استفاده کرد.

برای تشریح ساختار داخلی یک توالی‌گر نوعی ریزبرنامه، یک توالی‌گر خاصی که برای استفاده در ریزبرنامه کامپیوتر مناسب است و در بخش قبلی ارائه کردیم را نشان می‌دهیم. بلاک دیاگرام توالی‌گر ریزبرنامه در شکل ۸-۷ نمایش داده شده است. حافظه کنترل باین علت در شکل آمده است که

عکس‌العمل بین توالی‌گر و حافظه متصل به آن ملاحظه شود. در مدار دو مولتی‌پلکسر وجود دارد. اولین مولتی‌پلکسر آدرسی را از یکی از چهار منبع انتخاب کرده و آنرا به ثبات آدرس کنترل CAR هدایت می‌نماید. دومین مولتی‌پلکسر مقدار بیت وضعیت انتخاب شده را تست کرده و نتیجه تست را به یک مدار منطقی ورودی اعمال می‌نماید. خروجی CAR آدرس را برای حافظه کنترل تهیه می‌نماید. محتوای CAR افزایش یافته و به یکی از ورودی‌های مولتی‌پلکسر و نیز به ثبات زیر روال SBR اعمال می‌شود. سه ورودی دیگر مولتی‌پلکسر شماره ۱ از میدان آدرس ریز دستورالعمل جاری، از خروجی SBR، و از منبع خارجی که دستورالعمل را نگاشت می‌کند می‌آیند. اگر چه دیاگرام فقط یک ثبات زیر روال را نشان می‌دهد، یک توالی‌گر دارای پشته‌ای از ثباتها با عمق چهار تا هشت سطح می‌باشد. باین ترتیب، بطور همزمان چند زیرروال می‌توانند فعال باشند. اعمال پوش[1] و پوپ[2]، به همراه اشاره‌گر



شکل ۸-۷ توالی‌گر ریزبرنامه برای حافظه کنترل

1- Push 2- POP

پشته، آدرس برگشت را در حین اجرای ریزدستورالعمل‌های فراخوانی و بازگشت ذخیره و بازیابی می‌کنند.

میدان CD (شرط) در ریزدستورالعمل، یکی از بیت‌های وضعیت را در مولتی‌پلکسر دوم انتخاب می‌کند. اگر بیت انتخاب شده 1 باشد، متغیر T (تست) برابر 1 خواهد بود؛ در غیر اینصورت برابر 0 است. مقدار T به همراه دو بیت از میدان BR (انشعاب) به یک مدار منطقی ورودی می‌رود. مدار منطقی ورودی یک توالی‌گر خاص، نوع اعمالی که توالی‌گر می‌تواند انجام دهد را معین می‌کند. از جمله اعمال توالی‌گرها عبارتند از: افزایش، انشعاب یا پرش، فراخوانی و بازگشت از زیرروال، بارکردن آدرس خارجی، پوش یا پاپ پشته. سایر اعمال توالی‌گرهای تجاری سه یا چهار ورودی علاوه بر ورودی T دارند و لذا محدوده وسیعتری از عملیات را در اختیار می‌گذارند.

مدار منطقی ورودی شکل ۸-۷ دارای سه ورودی I_0، I_1 و T و سه خروجی S_0، S_1 و L می‌باشد. متغیرهای S_0 و S_1 یکی از منابع آدرس را برای CAR انتخاب می‌کنند. متغیر L ورودی بارکردن SBR را فعال می‌نماید. مقادیر دودویی دو متغیر، انتخاب مسیر موردنظر را در مولتی‌پلکسر معین می‌کنند. مثلاً، با $S_1\ S_0 = 10$، ورودی شماره ۲ مولتی‌پلکسر انتخاب و مسیری بین SBR و CAR بوجود می‌آید. توجه کنید که هر یک از چهار ورودی و خروجی MUX1 دارای هفت بیت آدرس هستند.

جدول درستی مدار منطقی ورودی در جدول ۴-۷ نشان داده شده است. ورودی‌های I_1 و I_0 همان بیت‌های میدان BR هستند. تابع مشخص شده برای هر وارده از جدول ۱-۷ گرفته شده است. مقادیر بیت‌های S_1 و S_0 با توجه به عمل مورد نظر و مسیری که انتخاب لازم را برقرار می‌کند تعیین می‌شود. اگر شرط بیت وضعیت در حین اجرای ریز دستورالعمل فراخوانی (BR = 01) برقرار باشد (T = 1)، مقدار افزایش یافته CAR در ثبات زیرروال بار می‌شود. با استفاده از این جدول می‌توان توابع ساده شده را برای مدار منطقی ورودی بدست آورد.

$$S_1 = I_1$$
$$S_0 = I_1\ I_0 + I'_1\ T$$
$$L = I'_1\ I_0\ T$$

این مدار را می‌توان با سه گیت AND، یک OR و یک معکوس کننده ساخت.

جدول ۴-۷ جدول درستی مدار منطقی ورودی توالی‌شمار ریزبرنامه

BR میدان		ورودی			MUX 1		بارکردن SBR
		I_1	I_0	T	S_1	S_0	L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	×	1	0	0
1	1	1	1	×	1	1	0

توجه کنید که مدار افزایشگر در توالی‌گر شکل ۸-۷ شمارنده‌ای که با فلیپ فلاپ ساخته شده باشد نیست. بلکه مداری ترکیبی است که با گیت ساخته شده است. یک مدار افزایشگر ترکیبی را می‌توان با اتصالی سری مدارهای نیم‌جمع‌کننده (شکل ۸-۴) طراحی کرد. رقم نقلی خروجی از یک طبقه باید به ورودی طبقه بعدی وصل شود. یکی از ورودی‌های کم‌ارزشترین طبقه باید برابر 1 باشد تا عمل افزایش یک واحدی صورت گیرد.

مسائل

۷-۱ تفاوت بین ریزپردازنده و ریزبرنامه چیست؟ آیا می‌توان ریزپردازنده‌ای بدون ریزبرنامه طراحی کرد؟ آیا همه کامپیوترهای ریزبرنامه‌نویسی شده ریزپردازنده هستند؟

۷-۲ تفاوت بین کنترل سخت‌افزاری و ریزبرنامه‌نویسی شده را بیان کنید. آیا می‌توان یک کنترل سخت‌افزاری را همراه با حافظه کنترل داشت؟

۷-۳ این اصطلاحات را تعریف کنید: (الف) ریزعمل؛ (ب) ریزدستورالعمل؛ (ج) ریزبرنامه؛ (د) ریزکد.

۷-۴ سازمان یک کنترل ریزبرنامه‌نویسی شده که در شکل ۱-۷ نشان داده شده است دارای تأخیرهای انتشاری زیر است. 40ns برای تولید آدرس بعدی، 10ns برای انتقال آدرس بداخل ثبات آدرس کنترل، 40ns برای دستیابی به حافظه کنترل، ROM ، 10ns برای انتقال ریزدستورالعمل به ثبات داده کنترل، و 40ns برای اجرای ریزعمل مورد نظر که بوسیله کلمه کنترل مشخص شده است. حداکثر فرکانس پالس ساعتی که کنترل می‌تواند بکار برد چیست؟ اگر ثبات داده کنترل مورد استفاده قرار نگیرد، فرکانس ساعت چیست؟

۷-۵ سیستم شکل ۲-۷ از یک حافظه کنترل 1024 کلمه‌ای 32 بیتی استفاده می‌کند. ریزدستورالعمل‌ها دارای سه میدان هستند، میدان ریزعمل‌ها 16 بیتی است.

(الف) میدان آدرس انشعاب چند بیتی است

(ب) اگر 16 بیت وضعیت وجود داشته باشد، چند بیت منطق انشعاب برای انتخاب یک بیت وضعیت لازم است.

(ج) چند بیت برای انتخاب ورودی مولتی‌پلکسر باقی‌مانده است

۷-۶ حافظه کنترل شکل ۲-۷ دارای 4096 کلمه 24 بیتی است.

الف) در ثبات آدرس کنترل چند بیت وجود دارد.

ب) هر یک از چهار ورودی که بداخل مولتی‌پلکسر می‌روند چند بیتی‌اند.

ج) تعداد ورودی‌های هر مولتی‌پلکسر چند است و چند مولتی‌پلکسر لازم است.

۷-۷ با استفاده از روش نگاشت که در شکل ۳-۷ توصیف شد، برای هر یک از کدهای عملیاتی زیر

اولین ریزدستورالعمل را بنویسید: (الف) 0010؛ (ب) 1011؛ (ج) 1111.

۷-۸ یک رویه نگاشت تنظیم کنید که هشت ریزدستورالعمل متوالی را برای هر روال در اختیار بگذارد. کد عمل شش بیتی است و حافظه کنترل 2048 کلمه دارد.

۷-۹ توضیح دهید که چگونه با استفاده از یک حافظه فقط خواندنی می‌توان عمل نگاشت از کد دستورالعمل به آدرس ریز دستورالعمل را انجام داد. مزیت این روش نسبت به روش شکل ۳-۷ چیست.

۷-۱۰ چرا در پیکربندی کامپیوتر شکل ۴-۷ به دو مولتی‌پلکسر نیاز داریم. آیا راه دیگری برای انتقال اطلاعات چند منبع به یک مقصد مشترک وجود دارد؟

۷-۱۱ با استفاده از جدول ۱-۷، برای ریزاعمال زیر، 9 بیت میدان ریزعمل را مشخص کنید.

الف) $DR \leftarrow DR + 1$ و $AC \leftarrow AC + 1$

ب) $DR \leftarrow M[AR]$ و $PC \leftarrow PC + 1$

ج) $AC \leftarrow DR$ و $DR \leftarrow AC$

۷-۱۲ با استفاده از جدول ۱-۷، ریزاعمال سمبلیک زیر را به عبارات انتقال ثباتی و دودویی تبدیل کنید.

الف) READ, INCPC

ب) ACTDR, DRTAC

ج) ARTPC, DRTAC, WRITE

۷-۱۳ فرض کنید که می‌خواهیم روال ADD، که در جدول ۲-۷ لیست شد را به دو ریزدستورالعمل زیر تبدیل کنیم:

```
ADD :  READ  I  CALL  INDR2
       ADD   U  JMP   FETCH
```

زیر روال INDR2 چه باید باشد

۷-۱۴ در زیر ریزبرنامه سمبلیک یک دستورالعمل در کامپیوتر بخش ۳-۷ آمده است

```
ORG  40
NOP     S    JMP    FETCH
NOP     Z    JMP    FETCH
NOP     I    CALL   INDRCT
ARTPC   U    JMP    FETCH
```

الف) عملی که پس از اجرای دستورالعمل انجام می‌شود را مشخص کنید.

ب) چهار ریزدستورالعمل را به فرم معادل دودویی آنها تبدیل کنید.

۷-۱۵ کامپیوتر بخش ۳-۷ دارای ریزبرنامه دودویی زیر است.

آدرس	ریزبرنامه دودویی
60	0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1
61	1 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0
62	0 0 1 0 0 1 0 0 0 1 0 1 0 0 1 1 1 1 1 1
63	1 0 1 1 1 0 0 0 0 1 1 1 1 0 1 1 1 1 0 0

الف) آنرا به ریزبرنامه سمبلیک، مطابق جدول ۲-۷، ترجمه کنید. (FETCH در آدرس 64 و INDRCT در آدرس 67 است.)

ب) تمام اشکالاتی را که با اجرای این ریزبرنامه در کامپیوتر پیش می‌آید بنویسید.

۷-۱۶ دستورات زیر را به کامپیوتر بخش ۳-۷ اضافه کنید (EA آدرس موثر است): برنامه سمبلیک را برای هر روال مطابق جدول ۲-۷ بنویسید. (توجه کنید که مقدار AC نباید تغییر کند مگر اینکه دستورالعمل تغییری در AC را مشخص نماید.)

سمبل	کد عمل	شکل سمبلیک	شرح
AND	0100	$AC \leftarrow AC \wedge M[EA]$	AND
SUB	0101	$AC \leftarrow AC - M[EA]$	تفریق
ADM	0110	$M[EA] \leftarrow M[EA] + AC$	جمع با حافظه
BTCL	0111	$AC \leftarrow AC \wedge \overline{M[EA]}$	صفر کردن بیت‌ها
BZ	1000	If $(AC = 0)$ then $(PC \leftarrow EA)$	انشعاب در صورت صفر بودن AC
SEQ	1001	If $(AC = M[EA])$ then $(PC \leftarrow PC + 1)$	گذر در صورت برابری
BPNZ	1010	If $(AC > 0)$ then $(PC \leftarrow EA)$	انشعاب اگر مثبت و غیرصفر

۷-۱۷ یک روال ریزبرنامه برای دستورالعمل ISZ (افزایش و گذر در صورت صفر بودن) که در فصل ۵ (جدول ۴-۵) تعریف شد بنویسید. از قالب دستورالعمل بخش ۳-۷ استفاده کنید. توجه کنید که شرط وضعیتی DR = 0 در میدان CD کامپیوتر بخش ۳-۷ وجود ندارد با این وجود می‌توانید AC و DR را با هم تعویض کرده و با بیت Z چک کنید که آیا AC = 0 است یا نه؟

۷-۱۸ روال‌های ریزبرنامه سمبلیک را برای دستورالعمل BSA (انشعاب و ذخیره آدرس) که در فصل ۵ تعریف شد بنویسید (جدول ۴-۵). از قالب ریزدستورالعمل بخش ۳-۷ استفاده کنید. تعداد ریزدستورالعمل‌ها را حداقل کنید.

۷-۱۹ نشان دهید که چگونه خروجی‌های 5 و 6 دیکدر F3 در شکل ۷-۷ به شمارنده برنامه PC وصل می‌شوند.

۷-۲۰ نشان دهید که چگونه یک میدان ۹ بیتی ریزعمل در ریزدستورالعمل را می‌توان به زیر

میدان‌هایی تقسیم کرد تا 46 ریزعمل را مشخص نمایند؟ چند ریزعمل می‌توان در یک ریزدستورالعمل مشخص کرد؟

۷-۲۱ کامپیوتری دارای 16 ثبات، یک ALU[1] با 32 عمل و یک شیفت دهنده با هشت عمل است که کلاً به یک سیستم گذرگاه مشترک متصل‌اند.

الف) برای یک ریز عمل کلمه کنترلی بنویسید.

ب) تعداد بیت‌ها را در هر میدان کلمه کنترل مشخص کنید و یک طرح انکد کردن کلی را ارائه نمائید.

ج) بیت‌های کلمه کنترل ریزعمل $R_4 \leftarrow R_5 + R_6$ را نشان دهید.

۷-۲۲ فرض کنید که منطق ورودی توالی‌گر ریزبرنامه شکل ۸-۷ چهار ورودی I_2، I_1، I_0 و T (تست) و سه خروجی S_1، S_0 و L را داراست. عملیات اجرا شده در واحد در جدول زیر لیست شده است. یک مدار منطقی ورودی با استفاده از حداقل گیت‌ها طراحی کنید.

I_2	I_1	I_0	عمل
0	0	0	افزایش CAR اگر T=1، پرش به AD اگر T=0
×	0	1	پرش غیرشرطی به AD
1	0	0	افزایش غیرشرطی CAR
0	1	0	پرش به AD اگر T=1، افزایش CAR اگر T=0
1	1	0	فراخوانی زیرروال اگر T=1، افزایش CAR اگر T=0
0	1	1	بازگشت غیرشرطی از زیرروال
1	1	1	نگاشت غیرشرطی آدرس خارجی

۷-۲۳ یک افزایشگر ترکیبی ۷ بیتی برای توالی‌گر ریزبرنامه شکل ۸-۷ طراحی کنید (شکل ۸-۴). با اضافه کردن یک ورودی کنترل D افزایشگر را تکمیل کنید. وقتی D = 0 است، مدار یک واحد افزایش می‌یابد، ولی وقتی D = 1 است مدار دو واحد افزایش می‌یابد.

۷-۲۴ بین MUX2 و مدار منطقی ورودی در شکل ۸-۷ یک گیت OR انحصاری اضافه کنید. یکی از ورودی‌های گیت از خروجی تست مولتی‌پلکسر می‌آید. ورودی دیگر گیت از بیتی با نام P[2] (بمعنی قطبیت) در ریزدستورالعمل حاصل از حافظه کنترل می‌آید. خروجی گیت به ورودی T در مدار منطقی ورودی متصل می‌شود. کنترل P چه کاری انجام می‌دهد.

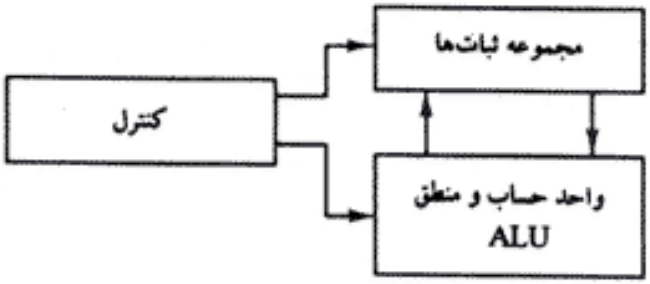
1- Arithmetic Logic Unit 2- Polarity

۸ واحد مرکزی پردازش

۸-۱ مقدمه
۸-۲ سازمان ثبات‌های عمومی
۸-۳ سازمان پشته
۸-۴ قالب دستورالعمل‌ها
۸-۵ روشهای آدرس دهی
۸-۶ انتقال و دستکاری داده‌ها
۸-۷ کنترل برنامه
۸-۸ کامپیوتر کم دستور RISC

۸-۱ مقدمه

بخشی از کامپیوتر که عمده پردازش داده‌ها را انجام می‌دهد واحد مرکزی پردازش نام داشته و آنرا CPU نیز می‌خوانند. CPU از سه بخش عمده تشکیل یافته است، شکل ۸-۱. مجموعه ثبات‌ها، داده‌های میانی که در حین اجرای دستورالعمل بکار می‌روند را ذخیره می‌نمایند. واحد حساب و منطق (ALU) ریزاعمال لازم برای اجرای دستورات را انجام می‌دهد. واحد کنترل مدیریت انتقال داده‌ها را دربین ثبات بعهده داشته و ALU را در انجام اعمال هدایت می‌کند.



شکل ۸-۱ اجزاء مهم یک CPU

www.homework.ir مشاوره و انجام پروژه هاي دانشجويي دانلود فصلهايي از كتب دانشگاهي

CPU انواع متعددی از توابع[1] را که توسط نوع خاصی دستور دیکته می‌شود اجرا می‌نماید. گاهی اوقات معماری[2] کامپیوتر بوسیله برنامه نویس، که دستورات زبان ماشین را بکار می‌برد، بعنوان ساختار و رفتار کامپیوتر تعریف می‌شود. این ساختار و رفتار قالب دستورالعمل‌ها، روش‌های آدرس دهی، مجموعه دستورالعمل‌ها و سازمان کلی ثبات‌های CPU است.

یکی از مرزهای مشترک دیدگاه‌های طراح کامپیوتر و برنامه نویس از ماشین، بخشی از CPU است که به مجموعه دستورات مربوط می‌شود. از نظر طراحان، مجموعه دستورالعمل‌ها، مشخصات طراحی CPU را فراهم می‌آورند. بخش عمده طراحی یک CPU انتخاب سخت افزار برای اجرای دستورات ماشین است. کاربری که کامپیوتر را در زبان ماشین یا اسمبلی برنامه ریزی می‌کند باید از مجموعه ثبات‌ها، ساختمان حافظه، نوع داده‌هایی که دستورالعمل‌ها می‌توانند با آنها کار می‌کنند و عملی که هر دستور اجرا می‌نماید آگاه باشد. مثال‌هایی از CPUهای ساده در فصل ۵ و ۷ آورده شد. این فصل سازمان و معماری یک CPU را با تاکید به نظرات کاربر کامپیوتر توصیف می‌کند. ابتدا بطور مختصر چگونگی ارتباط ثبات‌ها را با ALU از طریق گذرگاه‌ها توضیح داده و طرز کار پشته حافظه را تشریح می‌کنیم. سپس انواع قالب‌های دستورات موجود، روش‌های آدرس دهی بکار رفته در بازیابی داده‌ها از حافظه و انواع دستوراتی که عموماً در کامپیوترها دیده می‌شوند را نشان خواهیم داد. آخرین بخش مفهوم کامپیوترهای کم دستور[3] (RISC) را بیان دارد.

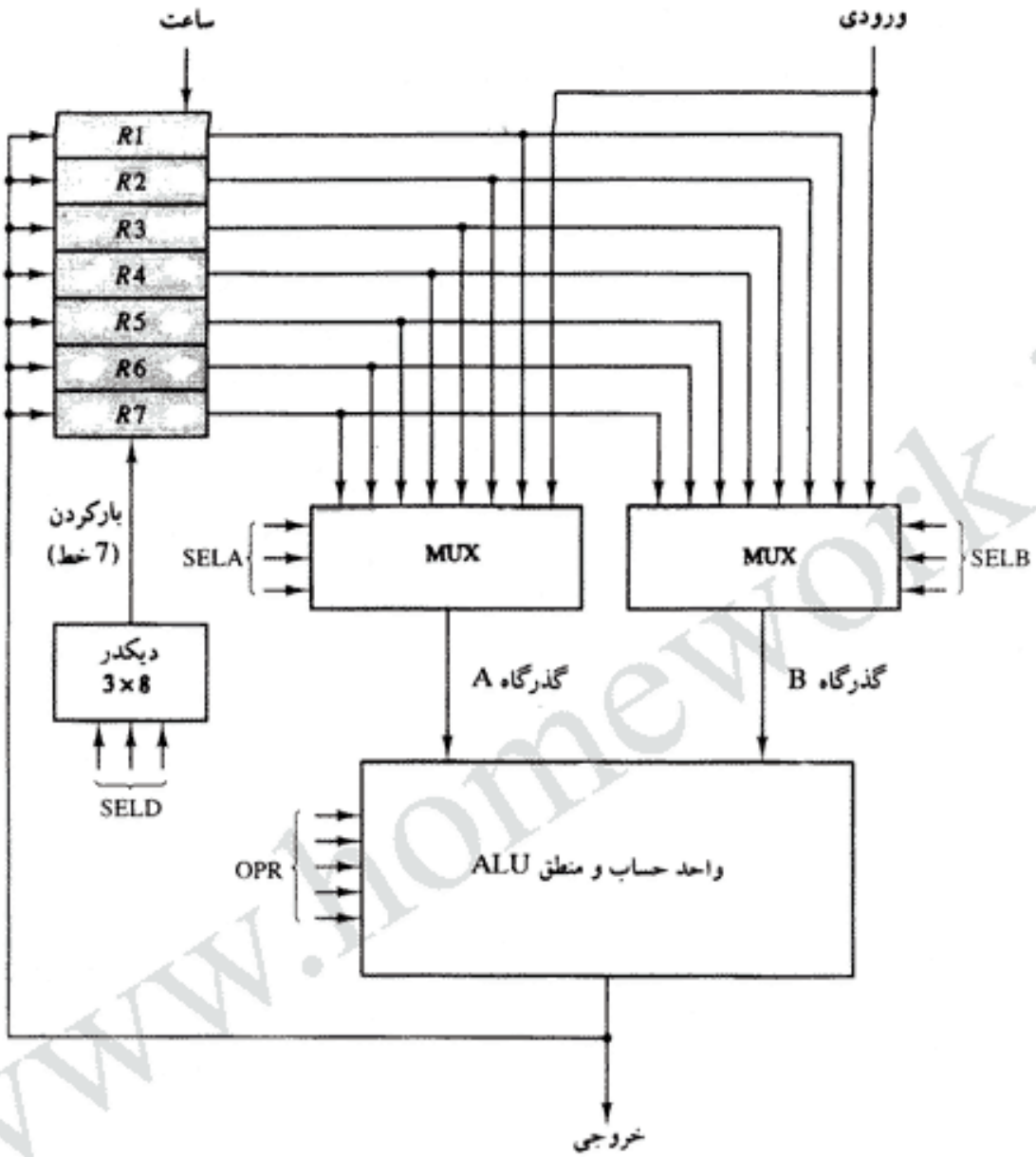
۲-۸ سازمان ثبات‌های عمومی

در مثال برنامه نویسی فصل ۶ دیدیم که مکان‌های حافظه برای ذخیره کردن اشاره‌گرها، شمارنده‌ها، آدرس برگشت نتایج موقت و حاصلضرب جزیی[4] در ضرب لازم هستند. ارجاع به مکان‌های حافظه برای اهداف فوق ،چیزی جز اتلاف وقت نیست، زیرا دستیابی به حافظه وقت گیرترین عمل در یک کامپیوتر است. بهترین و مفیدترین راه برای نگهداری این مقادیر میانی ذخیره کردن آنها در ثبات‌های پردازشگر است. وقتی که تعداد زیادی ثبات در CPU در نظر گرفته شد، بهترین راه بهره‌گیری از آنها ارتباطشان از طریق یک سیستم گذرگاه مشترک است. این ثبات‌ها تنها در انتقال مستقیم داده‌ها با هم ارتباط برقرار نمی‌کنند بلکه در حین اجرای انواع ریزعمل‌ها هم بهم وصل می‌شوند. از اینرو لازم است واحد مشترکی را فراهم آوریم تا تمام ریزاعمال حسابی، منطقی و شیفت را در پردازشگر اجرا نماید.

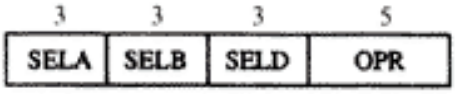
یک سازمان گذرگاه برای هفت ثبات CPU در شکل ۲-۸ نشان داده شده است. خروجی هر ثبات به دو مولتی پلکسر (MUX) وصل شده تا دو گذرگاه A و B را بسازد. خطوط انتخاب در هر مولتی پلکسر، یک ثبات یا داده ورودی را برای گذرگاه خاص انتخاب می‌نمایند. گذرگاه‌های A و B ورودی‌های واحد حساب - منطق (ALU) مشترک را تشکیل می‌دهند. عمل انتخاب شده در ALU

1- Function 2- Architecture

3- Reduced Instruction Set Computer 4- Partial Product



(الف) بلاک دیاگرام



(ب) کلمه کنترل

شکل ۲-۸ مجموعه ثبات‌ها با ALU مشترک

معین کننده ریزعمل‌های حسابی منطقی است که باید اجرا شوند. نتایج ریز عمل‌ها هم در خروجی و یا در ورودی همه ثبات‌ها قابل دسترسی است. ثباتی که اطلاعات را از گذرگاه خروجی دریافت می‌کند توسط یک دیکدر انتخاب می‌شود. دیکدر ورودی بارکردن، ثبات مزبور را فعال کرده و لذا یک مسیر

انتقال را بین داده در گذرگاه خروجی و ورودی ثبات انتخاب شده برقرار می‌سازد.

واحد کنترلی که سیستم گذرگاه CPU را اداره می‌کند، جریان اطلاعات بین ثبات‌ها و ALU را با انتخاب بخش‌های مختلف جهت می‌بخشد. مثلاً برای اجرای عمل

$$R_1 \leftarrow R_2 + R_3$$

کنترل باید متغیرهای انتخاب دودویی مناسب را برای ورودی‌های انتخاب کننده زیر تهیه نماید

۱- انتخاب کننده مولتی پلکسر A (SELA): برای قرار دادن محتوای R_2 روی گذرگاه A

۲- انتخاب کننده مولتی پلکسر B (SELB): برای قرار دادن محتوای R_3 روی گذرگاه B

۳- انتخاب‌گر عملیات در ALU (OPR): برای انجام عمل جمع حسابی A + B

۴- انتخاب‌گر دیکدرمقصد (SELD): برای انتقال محتویات گذرگاه خروجی به R_1

چهار متغیر کنترل فوق در واحد کنترل تولید و باید در آغاز یکی از پالس‌های ساعت در دسترس باشند. در طول یک سیکل ساعت، داده‌ها از دو ثبات منبع از طریق گیت‌های مولتی پلکسرها به ALU، سپس به گذرگاه خروجی و از آنجا به ورودی ثبات‌های مقصد می‌رسند. بنابراین با رسیدن پالس ساعت بعدی اطلاعات دودویی از گذرگاه خروجی به ثبات R_1 وارد می‌شود. برای دستیابی به یک واکنش سریع، ALU از مدارات سریع العمل ساخته شده است. گذرگاه‌ها از مولتی پلکسرها و گیت‌های سه حالته مطابق آنچه در بخش (۳-۴) دیده شد پیاده سازی شده‌اند.

کلمه کنترل

در سیستم، چهارده ورودی انتخاب دودویی وجود دارد که ترکیب مقدار عددی آنها یک کلمه کنترل[1] را تشکیل می‌دهد. کلمه کنترل 14 بیتی در شکل ۲-۸ (ب) تعریف شده‌است. این کلمه از چهاربخش تشکیل می‌شود. سه بخش از آن هر یک سه بیت و یک بخش پنج بیتی است. سه بیت SELA یک ثبات مبدأ را برای ورودی A از واحد ALU انتخاب می‌کنند. سه بیت مربوط به SELB هم یک ثبات دیگر را برای ALU از طریق ورودی B انتخاب می‌نمایند. سه بیت SELD ثبات مقصد را بکمک دیکدر و هفت خروجی مربوطه‌اش انتخاب می‌نمایند. پنج بیت OPR یکی از عملیات ALU را برمی‌گزینند. چهارده بیت کلمه کنترل فوق وقتی کلاً به ورودی‌های انتخاب اعمال شوند یک ریزعمل خاصی را مشخص خواهند کرد.

جدول (۱-۸) کد انتخاب ثبات‌ها را مشخص می‌نماید. سه بیت دودویی ستون اول کد در جدول کد دودویی هر یک از سه میدان را مشخص می‌کند. ثباتی که توسط میدان SELA و SELB و SELD انتخاب می‌شود عدد دهدهی مربوطه‌اش معادل با عدد دودویی کد مربوطه‌اش می‌باشد. هرگاه SELA

1- Control Word

جدول ۱-۸ تخصیص کد به میدانهای انتخاب ثبات‌ها

کد دودویی	SELA	SELB	SELD
000	ورودی	ورودی	هیچ
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

یا SELB برابر 000 باشد، مولتی پلکسر مربوطه داده را از ورودی خارجی می‌پذیرد. وقتی SELD=000 باشد هیچیک از ثبات‌ها انتخاب نشده ولی محتوای گذرگاه روی خروجی خارجی قرار می‌گیرد.

ALU اعمال حسابی و منطقی را انجام می‌دهد. بعلاوه CPU باید قابلیت انجام شیفت را داشته باشد. مدار شیفت دهنده می‌تواند در ورودی ALU قرار گرفته و امکان پیش شیفت [1] را فراهم آورد و یا پس از ALU بوده و امکان پس شیفت [2] را بوجود آورد. در برخی مواقع این عمل را خود ALU انجام می‌دهد. یک مدار حساب، منطق و شیفت در بخش (۷-۴) طراحی شد. جدول عملیات این ALU در جدول (۸-۴) دیده شد. کدگذاری [3] عملهای ALU برای CPU از بخش (۷-۴) گرفته شده و در جدول (۲-۸) آورده شده است. میدان OPR از پنج بیت تشکیل شده و هر عمل با یک نام سمبلیک مشخص شده است.

جدول ۲-۸ تخصیص کد به اعمال ALU

OPR انتخابگر	عمل	سمبل
00000	انتقال A	TSFA
00001	افزایش A	INCA
00010	جمع A+B	ADD
00101	تفریق A−B	SUB
00110	کاهش A	DECA
01000	AND ثبات‌های A و B	AND
01010	OR ثبات‌های A و B	OR
01100	XOR ثبات‌های A و B	XOR
01110	متمم کردن A	COMA
10000	شیفت A به راست	SHRA
11000	شیفت A به چپ	SHLA

1- Preshift 2- Postshift 3- Encode

مثال‌هایی از ریزعمل‌ها

برای تعیین یک ریزعمل در CPU به یک کلمه کنترل 14 بیتی نیاز است. کلمه کنترل مربوط به یک ریزعمل معین را می‌توان از متغیرهای انتخاب بدست آورد. مثلاً برای ریز عمل تفریق که با عبارت زیر مشخص می‌شود:

$$R_1 \leftarrow R_2 - R_3$$

R_2 را برای ورودی A از ALU، R_3 را برای ورودی B از ALU و R_1 را بعنوان ثبات مقصد و عمل تفریق A−B را در ALU مشخص می‌نماید. بنابراین کلمه کنترل از چهارمیدان تشکیل و هر میدان از لیست کدهای جدول (۱-۸) و (۲-۸) بدست می‌آید. کلمه کنترل دودویی برای ریزعمل تفریق برابرست با 00101 001 011 010که بطریق زیر بدست می‌آید.

بخش:	SELA	SELB	SELE	OPR
سمبل:	R_2	R_3	R_1	SUB
کلمه کنترل:	010	011	001	00101

کلمه کنترل برای تفریق و چند دستور دیگر در جدول (۳-۸) آورده شده است.

ریزعمل‌های افزایش و انتقال، ورودی B از ALU را بکار نمی‌برند. در این موارد برای میدان B خط تیره (-) قرار داده می‌شود. برای هر میدان بکار نرفته، مانند فوق، در کلمه کنترل عدد دودویی 000 را تخصیص خواهیم داد هر چند که هر عدد دودویی دیگر نیز قابل استفاده است. برای قرار دادن محتوای هر ثبات در خروجی، محتوا را در ورودی A از ALU قرار می‌دهیم ولی هیچیک از ثبات‌ها برای پذیرش داده انتخاب نمی‌شوند. عمل TSFA در ALU، داده را از ثبات به ALU و از آن جا به خروجی هدایت می‌کند. انتقال مستقیم از ورودی به خروجی توسط کلمه کنترلی تماماً 0 (برای میدان B هم 000 در نظر گرفته شود) انجام می‌گردد. یک ثبات هم می‌تواند با انجام عمل OR

جدول ۳-۸ مثالهایی از ریزعمل‌های CPU

ریزعمل	مشخص کردن سمبل				کلمه کنترل
	SELA	SELB	SELD	OPR	
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

انحصاری[1] (XOR) صفرشود، زیرا $x \oplus x=0$

باتوجه به این مثال‌ها واضح است که ریزعمل‌های بسیار دیگری را نیز در CPU می‌توان ایجاد کرد. یک روش مناسب تر در تولید کلمات کنترل با بیت‌های متعدد ذخیره سازی آنها در یک واحد حافظه است. این حافظه که کلمات کنترل را ذخیره می‌کند حافظه کنترل خوانده می‌شود. با خواندن متوالی کلمات کنترل می‌توان رشته ریزعمل‌های دلخواه را برای CPU ایجاد کرد.

این نوع کنترل به کنترل ریزبرنامه نویسی[2] شده معروف است. این نوع واحد کنترل در شکل (۸-۷) نشان داده شد. کلمه دودویی کنترل برای CPU از خروجی‌هائی که در حافظه کنترل با "ریزعملها" مشخص شده می‌آیند.

۸-۳ سازمان پشته

یک ویژگی مفید که در CPU اغلب کامپیوترها وجود دارد پشته یا لیست اولین ورودی - آخرین خروجی[3] (LIFO) است. پشته یک وسیله ذخیره سازی است که اطلاعات را بنحوی ذخیره می‌کند که همیشه آخرین کمیت ذخیره شده اولین کمیت بازیابی شونده می‌باشد. عمل یک پشته را می‌توان با رویهم چیدن سینی‌ها مقایسه کرد. آخرین سینی در بالای مجموعه سینی‌ها اولین سینی برداشته شده از آن خواهد بود.

دریک کامپیوتر دیجیتال،پشته اصولاً یک واحد حافظه است به همراه ثبات آدرسی که تنها می‌تواند بشمارد. (پس از یک مقدار اولیه‌ای که می‌توان در آن بارکرد). ثباتی که آدرس پشته را نگهداری می‌نماید اشاره‌گر پشته[4] نامیده می‌شود (SP)، زیرا مقدار آن همیشه به بالاترین کمیت در پشته اشاره می‌کند. برخلاف پشته‌ای ازسینی‌ها که در آن خود سینی را می‌توان خارج یا داخل پشته کرد، ثبات‌های فیزیکی پشته فقط همواره برای خواندن و نوشتن در دسترس قرار دارند. در واقع کلمه است که می‌تواند در پشته درج یا حذف شود.

دو عمل پشته عبارتند ازدرج یا حذف کمیت. عمل درج داده پوش[5] (فشار دادن یا هول دادن) نامیده می‌شود زیرا عمل را می‌توان بصورت فشار دادن یک کمیت جدید در بالای آن تصور کرد. عمل حذف را پاپ[6] (کشیدن یا برداشت) گویند زیرا عمل را می‌توان بصورت حذف یک کمیت از بالای آن در نظر گرفت. با این وجود چیزی بداخل پشته فشار داده و یا کشیده نمی‌شود. این اعمال باافزایش یا کاهش ثبات اشاره‌گر پشته شبیه سازی می‌گردند.

پشته‌های ثباتی

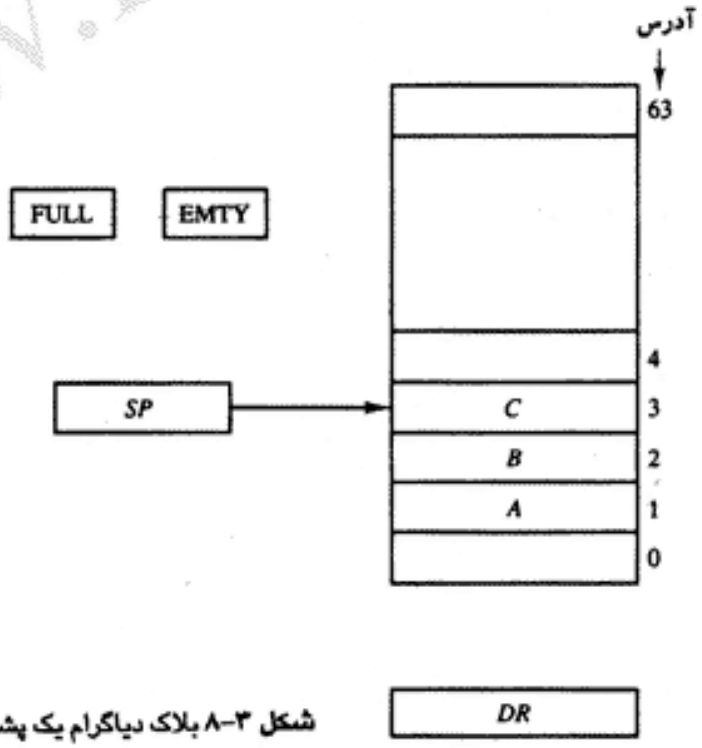
یک پشته می‌تواند دربخشی از یک حافظه بزرگ یا بصورت مجموعه‌ای از تعداد معینی حافظه یا

1- Exclusive OR
2- Microprogrammed Control
3- Last – In , First – Out
4- Stack Pointor
5- Push
6- Pop

ثبات قرار داشته باشد. شکل ۳-۸ سازمان یک پشته ثباتی 64کلمه‌ای را نشان می‌دهد. محتوای اشاره‌گر SP را عددی دودویی تشکیل می‌دهد که مقدار آن برابر آدرس کلمه‌ای است که در آن لحظه در بالای پشته قرار دارد. سه کمیت A و B و C بترتیب درپشته قرارگرفته‌اند. کمیت C در بالای پشته واقع و لذا محتوای SP برابر 3 است. برای برداشتن کمیت فوقانی، پشته با خواندن، کلمه حافظه را از آدرس 3 برداشته و محتوای SP را یک واحد کم می‌کند. حال کمیت B در بالای پشته است زیرا اشاره‌گر پشته آدرس 2 را در خود دارد. برای وارد کردن کمیت جدید، با نوشتن در پشته اشاره‌گر SP یک واحد اضافه می‌شود. توجه کنید که هرچند کمیت C خوانده شد ولی بطور فیزیکی از بین نرفته است. این مطلب اهمیتی ندارد زیرا وقتی در پشته چیزی نوشته شود، کمیت جدید در همان محل نوشته می‌شود.

دریک پشته 64کلمه‌ای ، اشاره‌گر پشته 6 بیتی است زیرا $2^6=64$ است. چون اشاره‌گر SP تنها دارای 6 بیت است نمی‌تواند از 63 (دودویی 111111) تجاوز کند، و لذا SP فقط می‌تواند شش رقم پائین رتبه را در خود جای دهد. بطورمشابه وقتی 000000 یک واحد کم شود نتیجه 111111 خواهد بود. هرگاه پشته پرباشد، ثبات یک بیتی FULL برابر 1 می‌شود و هر وقت پشته خالی باشد، ثبات یک بیتی EMTY برابر 1 می‌گردد. DR ثباتی است که آنچه باید در پشته نوشته شود و یا از آن خوانده شود را در خود نگه می‌دارد.

درابتدا که SP پاک می‌شود، EMTY برابر 1 و FULL هم 0 می‌گردد بدین ترتیب اشاره‌گر پشته به آدرس 0 اشاره نموده و پشته خالی است. اگر پشته پرنباشد (FULL=0) یک کمیت جدید با عمل پوش



شکل ۳-۸ بلاک دیاگرام یک پشته ۶۴ کلمه‌ای

وارد آن می‌شود. عمل پوش با ریز اعمال زیر صورت می‌گیرد:

SP ← SP + 1	افزایش اشاره‌گر پشته
M [SP] ← DR	نوشتن داده در مکان بالای داده
If(SP= 0) then (FULL ←1)	آیا پشته پر است یا نه
EMTY ← 0	علامت خالی نبودن پشته

اشاره‌گر پشته برای اشاره به کلمه بالاتر افزایش می‌یابد. سپس اجرای عمل نوشتن در حافظه سبب انتقال DR به مکان بالای پشته می‌گردد. توجه کنید که SP حاوی آدرس مکان بالای پشته است و M[SP] کلمه‌ای از حافظه است که توسط آدرس فعلی موجود در اشاره‌گر پشته SP مشخص می‌گردد. اولین کمیت ذخیره شده در پشته در آدرس 1 قرار دارد. آخرین کمیت در آدرس 0 ذخیره شده است. اگر اشاره‌گر پشته به 0 برسد، پشته پراست و بنابراین FULL=1 خواهد بود. این وضعیت هنگامی رخ می‌دهد که بالاترین کمیتی که متعلق به آخرین پوش است در مکان 63 قرار گرفته باشد، و پس از افزایش SP، آخرین کمیت جدید در مکان 0 ذخیره خواهد شد. به محض ذخیره شدن این کمیت اخیر در مکان 0 حافظه، دیگر محل خالی وجود ندارد. اگر کمیتی در پشته نوشته شده باشد مسلماً دیگر پشته خالی نیست و لذا EMTY=0 می‌گردد.

اگر پشته خالی نباشد (یعنی EMTY=0) یک کمیت از پشته حذف می‌شود. عمل پاپ از ریز عملهای زیر تشکیل می‌شود.

DR ← M[SP]	خواندن داده از مکان بالای پشته
SP ← SP−1	کاهش اشاره‌گر پشته
If (SP=0) then (EMTY ← 1)	آیا پشته خالی است
FULL ← 0	علامت پرنبودن پشته

کمیت بالای پشته از پشته به DR منتقل می‌شود. سپس اشاره‌گر پشته کاهش می‌یابد. اگر مقدار آن به صفر برسد، پشته خالی است و EMTY = 1 می‌شود. اگر وضعیت هنگامی رخ دهد که کمیت خوانده شده در مکان 1 باشد به محض خوانده شدن این کمیت ، SP کاهش یافته و به 0 می‌رسد که مقدار اولیه SP بوده است. توجه کنید که اگر یک عمل پاپ کمیتی را از مکان 0 بخواند و سپس SP کاهش یابد، SP به 111111 تغییر می‌یابد که معادل عدد دهدهی 63 است. در این آرایش، کلمه موجود در آدرس 0 آخرین کمیت پشته را دریافت می‌نماید. همچنین توجه کنید که اگر در FULL=1 پشته پوش شود یا در EMTY=1 پشته پاپ شود خطای عملیاتی رخ می‌دهد.

پشته حافظه‌ای

پشته می‌تواند بعنوان یک واحد مستقل طبق شکل ۳-۸ وجود داشته باشد و یا در یک حافظه

RAM متصل به CPU پیاده سازی شود. پیاده سازی پشته در CPU بدین ترتیب صورت می‌گیرد که بخشی از حافظه به کمک یک ثبات پردازشگر به پشته اختصاص می‌یابد. شکل ۴-۸ بخشی از حافظه کامپیوتر را که به سه قسمت شده نشان می‌دهد: این سه قسمت به برنامه، داده و پشته تخصیص یافته است. شمارنده برنامه PC به آدرس دستور بعدی اشاره می‌کند. ثبات آدرس AR به آرایه ای از داده‌ها اشاره می‌کند. اشاره‌گر پشته نیز به بالای پشته اشاره می‌نماید. سه ثبات به یک گذرگاه آدرس مشترک متصلند و همواره یکی از آنها می‌تواند آدرسی را برای حافظه فراهم آورد. PC در طول فاز برداشت[1] برای خواندن یک دستورالعمل بکار می‌رود. AR درطول فاز اجرا برای خواندن عملوند بکار گرفته می‌شود. SP نیز بهنگام پوش یا پاپ کمیت‌ها را به پشته می‌فرستد یا از آن برمی‌دارد.



شکل ۴-۸ حافظه کامپیوتر با قطعه‌های برنامه، داده و پشته

1- Fetch

همانطور که در شکل ۴-۸ ملاحظه می‌شود، مقداراولیه SP برابر 4001 بوده و پشته با کاهش آدرس پرمی‌شود. بنابراین اولین کمیت در آدرس 4000 ذخیره می‌گردد، دومین کمیت در 3999 و بالاخره آخرین کمیت می‌تواند در آدرس 3000 ذخیره گردد.

در اینجا فرض می‌کنیم که کمیت‌ها در پشته با ثبات داده DR ارتباط برقرار می‌کنند. کمیت جدید با عمل پوش طبق عبارات زیر وارد می‌شود.

$$SP \leftarrow SP - 1$$
$$M[SP] \leftarrow DR$$

دیده می‌شود که اشاره‌گر کاهش یافته و به آدرس کلمه بعدی اشاره می‌نماید. عمل نوشتن در حافظه هم کلمه را از DR وارد قسمت فوقانی پشته می‌نماید. یک کلمه نیز با عمل پاپ طبق عبارات زیر پاک می‌شود.

$$DR \leftarrow M[SP]$$
$$SP \leftarrow SP + 1$$

کمیت فوقانی از پشته خوانده شده و در DR قرار می‌گیرد. سپس اشاره‌گر پشته یک واحد اضافه شده و به مکان بعدی در پشته اشاره می‌نماید.

بیشتر کامپیوترها، سخت افزاری برای سرریزشدن[1] پشته (پرترشدن ازحداکثر) و یا فروریز[2] (خالی‌ترشدن از حداقل) ندارند. حدود پشته می‌تواند با دو ثبات پردازشگر چک شود: یکی از آنها تا حد بالا (دراین حالت 3000) و دیگری حد پائین (دراین حالت 4000) را نگه میدارند. پس از یک عمل پوش، SP با ثبات حدبالا و پس از پاپ با ثبات حد پائین مقایسه می‌شود.

دو ریزعمل لازم برای پوش یا پاپ عبارتنداز (۱) دستیابی به حافظه توسط SP و (۲) بهنگام کردن[3] SP. اینکه کدام ریزعمل ابتدا انجام می‌شود و یا اینکه تصحیح SP با افزایش و یا کاهش صورت می‌گیرد بستگی به سازمان پشته دارد. در شکل ۴-۸ گسترش پشته با کاهش آدرس حافظه همراه است. پشته می‌تواند با افزایش آدرس هم مطابق شکل ۳-۸ گسترش یابد. در این حالت SP برای پوش افزایش یافته و برای پاپ کاهش می‌یابد. می‌توان پشته را طوری طراحی کرد که SP به مکان خالی بعدی در بالای پشته اشاره کند. در این حالت ترتیب اعمال جزیی با یکدیگر تعویض می‌شوند.

اشاره‌گر پشته ابتدا با یک مقداراولیه‌ای بار می‌شود. این مقدار اولیه باید آدرس پایین یک پشته در حافظه باشد. از آن پس SP بطور اتوماتیک با هر پوش یا پاپ افزایش یاکاهش می‌یابد. مزیت یک پشته حافظه‌ای این است که CPU می‌تواند بدون تعیین آدرس از آن استفاده نماید زیرا آدرس همیشه بطور اتوماتیک در اشاره‌گر بهنگام (آماده) است

1- Overflow 2- Under flow 3- Update

نمایش لهستانی معکوس[1]

سازمان پشته برای ارزیابی عبارات حسابی بسیار مفید است. روشهای ریاضی معمول در نوشتن عبارات حسابی وقتی توسط یک کامپیوتر ارزیابی شوند مشکلاتی را ایجاد می‌نمایند. عبارات حسابی معمول بصورت نمایش میانوندی[2] نوشته شده و عملگر بین دو عملوند قرار می‌گیرد. عبارت ساده حسابی زیرا را در نظر بگیرید

A * B + C * D

علامت ستاره (که در واقع ضرب می‌باشد) بین دو عملوند A و B یا C و D قرار گرفته است. علامت جمع بین دو حاصلضرب قرار گرفته است. برای محاسبه این عبارت لازم است A*B را بدست آورده، حاصلضرب را ذخیره و پس از انجام C*D مجموع دو حاصلضرب را بدست آوریم. ازاین مثال می‌بینیم در محاسبه عبارات میانوندی لازم است جستجوهایی رو به جلو و عقب در عبارت صورت گیرد تا عمل بعدی انجام پذیرد.

ریاضی دان لهستانی لوکاسویچ[3] نشان داد که عبارات ریاضی می‌توانند بصورت نمایش پیشوندی[4] نشان داده شوند. این نمایش عبارات اغلب نمایش لهستانی خوانده می‌شوند و در آن عملگر را قبل از عملوندها قرار می‌دهند. نمایش پسوندی[5] یا نمایش لهستانی معکوس (RPN) عملگر را بعدازعملوندها قرار می‌دهد. مثال‌های زیر سه فرم نمایش عبارات را نشان می‌دهد.

نمایش میانوندی A + B
نمایش پیشوندی لهستانی +AB
نمایش پسوندی لهستانی AB+

روش لهستانی معکوس برای عملیات پشته مناسب است، عبارت

A * B + C * D

در این روش بشکل زیر در می‌آید

AB * CD * +

و بطریق زیر ارزیابی می‌شود: عبارت را از چپ به راست جستجو نمایید، وقتی که به عملگری رسیدید آنرا بر روی دو عملوند قبلی اعمال کنید. سپس دو عملوند را حذف وآنها را با حاصل جایگزین نمائید. این عمل ادامه می‌یابد تا اینکه دیگر به عملگری برخورد نشود.

1- Reverse Polish Notation, RPN
2- Infix notation
3- Lukasiewicz
4- Prefix (or polish notation)
5- Postfix notation (or reverse polish notation)

در مثال فوق عملگر * را پس از A و B می‌بینیم. عمل A * B را اجرا و A و B و * را با حاصلضرب جایگزین می‌کنیم

(A * B) CD * +

که در آن (A * B) حاصلضرب است. عملگر بعدی هم * است و دو عملگر قبلی آن C و D می‌باشند. بنابراین عمل C * D اجرا و حاصل عبارتی است با دو عملوند و یک عملگر

(A * B) (C * D) +

عمل بعدی + است و عملوندهایی که باید با هم جمع شوند دوحاصلضرب قبلی می‌باشند، بنابراین دو کمیت را جمع می‌کنیم تا نتیجه کلی بدست آید.

در تبدیل از نمایش میانوندی به نمایش لهستانی معکوس باید سلسله مراتب اولویت عمل‌ها مورد استفاده در نمایش میانوندی در نظر گرفته شود. این اولویت بندی حکم می‌کند که ابتدا همه عمل‌های داخل پرانتز درونی را انجام دهیم، سپس داخل بقیه پرانتزها را انجام دهیم، آنگاه اعمال ضرب و تقسیم و بالاخره جمع و تفریق اولویت‌های بعدی را دارند. عبارت زیر را در نظر بگیرید.

(A + B) * [C * (D + E) + F]

برای محاسبه این عبارت ما ابتدا باید داخل پرانتزهای (A + B) و (D + E) را ارزیابی کنیم. سپس باید عبارت داخل کروشه بدست آید. ضرب C * (D +E) بر جمع با F اولویت دارد چون ضرب بر جمع مقدم است. آخرین عمل ضرب بین پرانتز و کروشه است. عبارت فوق را می‌توان با توجه به روش لهستانی معکوس به عبارتی بدون پرانتز تبدیل کرد. عبارت تبدیل شده چنین است:

AB + DE + C * F + *

با پیشروی از چپ به راست ابتدا A و B را جمع می‌کنیم، سپس حاصل جمع D و E را بدست می‌آوریم. در این حالت داریم

(A + B) (D + E) C * F + *

که در آن (A + B) و (D + E) هر کدام یک کمیت تک مقداری حاصل از جمع می‌باشند. دو عملوند مربوط به * بعدی C و (D + E) هستند. این دو عدد در هم ضرب شده و حاصل با F جمع می‌شود. آخرین * نیز ضرب دو جمله (A + B) و حاصل ضرب قبلی را انجام می‌دهد.

ارزیابی عبارات حسابی

روش RPN به همراه یک مجموعه پشته ثباتی بهترین روش شناخته شده برای ارزیابی عبارات

حسابی است. این روش در بسیاری از ماشین‌های حساب و نیز برخی از کامپیوترها رایج است. پشته خصوصاً برای حل مسائل طولانی و پیچیده از جمله محاسبات زنجیره‌ای مفید است. زیرا عبارات حسابی می‌توانند بصورت RPN بدون پرانتز بیان گردند.

روش مذکور بدین ترتیب انجام می‌شود که ابتدا عبارات حسابی به فرم معادل RPN در می‌آیند عملوندها بداخل پشته به همان ترتیبی که نشان داده شده‌اند هدایت می‌شوند. آغاز هر عمل بستگی به این دارد که ماشین حساب یا کامپیوتر بکار رفته باشد. در ماشین حساب، عملگرها از طریق صفحه کلید وارد می‌شوند. در کامپیوتر اعمال توسط دستوراتی که دارای بخش کد عملیاتی هستند و آدرس هم نیاز ندارند شروع می‌گردد. ریزعمل‌های زیر با پشته هرگاه که یک عملگر در ماشین حساب وارد شود و یا عملگری در کامپیوتر توسط کنترل تأیید گردد اجرا می‌گردد: (۱) دو عملوند که در بالاترین مکان پشته قرار دارند برای اجرای عمل مورد نظر انتخاب می‌شوند، و (۲) پشته پاپ می‌شود و نتیجه عمل بجای عملوند پائین‌تر می‌نشیند. با پوش کردن متوالی عملوندها بداخل پشته و انجام اعمال تعریف شده فوق، عبارت بطور صحیح ارزیابی شده و نتیجه نهایی در بالای پشته باقی می‌ماند.

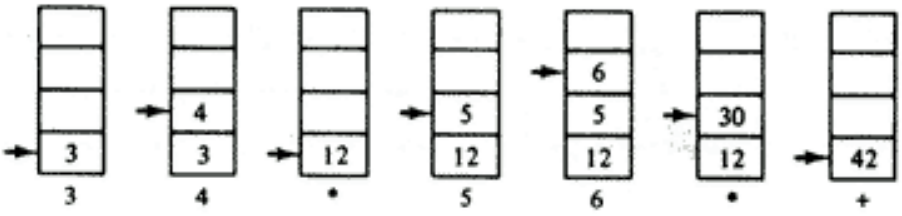
مثال عددی زیر می‌تواند این روش را واضح تر بیان نماید. عبارت حسابی زیر را در نظر بگیرید.

$$(3 * 4) + (5 * 6)$$

در RPN عبارت بصورت زیر است

$$34 * 56 * +$$

حال عملیات پشته طبق آنچه در شکل ۵-۸ آمده است را ملاحظه نمائید. هر قسمت از شکل یک مرحله از کار پشته را نشان می‌دهد و فلش همواره به بالای پشته اشاره می‌کند. با مرور عبارت از چپ به راست با دو عملوند مواجه می‌شویم. ابتدا عدد 3 و سپس 4 بداخل پشته فرستاده می‌شوند. نماد بعدی عملگر ضرب * است. این نماد موجب ضرب دو عدد واقع در بالاترین مکان پشته می‌شود. سپس پشته پاپ شده و حاصلضرب در بالای پشته قرار می‌گیرد، و در واقع جایگزین دو عملوند می‌گردد. آنگاه در عبارت فوق با دو عملوند دیگر یعنی 5 و 6 مواجه می‌شویم و آنها را بداخل پشته پوش می‌کنیم. نتیجه عمل پشته که حاصل از * بعدی است دو عدد اخیر را با حاصلضرب جایگزین می‌سازد. آخرین عمل ، جمع بر روی دو عدد بالای پشته است تا عدد نهایی 42 حاصل گردد.



شکل ۵-۸ عمل‌هایی که برای محاسبه 6 × 5 + 4 × 3 روی پشته انجام می‌شود

ماشین حساب علمی[1] که از پشته داخلی استفاده می‌کنند نیاز به تبدیل عبارات به RPN توسط کاربر را دارند. کامپیوترهایی که CPU آنها دارای پشته است نیاز به برنامه‌ای دارند تا تبدیل را برای کاربر انجام دهد. اغلب کامپایلرها، بدون توجه به سازمان CPU، تمام عبارات حسابی را به RPN تبدیل می‌کنند زیرا این بهترین روش برای تبدیل عبارات حسابی به دستورالعمل زبان ماشین است. بنابراین یک CPU با سازمان پشته مسلماً در برخی کاربردها مفیدتر از CPU بدون پشته است.

۸-۴ قالب دستورالعمل‌ها

ساختار فیزیکی و منطقی کامپیوترها معمولاً در کتابچه راهنمایشان که به همراه سیستم است تشریح می‌شود. چنین کتابچه‌های ساختمان داخلی CPU، از جمله ثبات‌های موجود و قابلیت‌های منطقی آنها را توضیح می‌دهد. آن‌ها فهرست همه دستورات پیاده‌سازی شده با سخت افزار، قالب کد آنها و نهایتاً تعریف دقیقی برای هر دستور را فراهم می‌آورند. یک کامپیوتر معمولاً انواع متنوعی از قالب دستورات را داراست. وظیفه واحد کنترل در CPU این است تا هر کد دستورالعمل را تفسیر و تابع کنترل لازم برای پردازش دستورالعمل را تولید کند.

قالب یک دستورالعمل معمولاً در یک چهارگوشه کشیده می‌شود که در آن بیت‌های دستورالعمل آن طور که در کلمه حافظه یا ثبات کنترل ظاهر می‌شوند آمده است. بیت‌های دستورالعمل به گروه‌هایی بنام میدان[2] تقسیم می‌شود. متداول‌ترین میدان‌هایی که در قالب دستورالعمل‌ها دیده می‌شوند، عبارتند از :

۱- میدان کد عملیات که عمل را مشخص می‌کند

۲- میدان آدرس که آدرس حافظه یا ثبات پردازشگر را معین می‌نماید.

۳- میدان روش[3] (شیوه) که نحوه تعیین آدرس مؤثر یا عملوند را مشخص می‌نماید.

تحت شرایط خاصی، میدان‌های خاص دیگری نیز گاهی اوقات بکار گرفته می‌شوند، مثلاً میدانی که تعداد شیفت‌های یک دستورالعمل از نوع شیفت را معین می‌کند.

میدان کد عملیات یک دستورالعمل عبارتست از گروهی از بیت‌ها که انواع اعمال پردازشگر مانند جمع، تفریق، مکمل‌سازی و شیفت را تعریف می‌نمایند. متداول‌ترین اعمال موجود درکامپیوترها در بخش ۶-۸ بحث شده است. بیت‌هایی که روش را برای دستور تعریف می‌کنند راه‌های مختلفی را برای انتخاب عملوند از آدرس مفروض معین می‌نمایند. روشهای مختلف آدرس دهی که برای کامپیوترهای دیجیتال می‌توان ردیف کرد در بخش ۵-۸ ارائه شده‌اند. در این بخش درباره میدان آدرس یک دستورالعمل بحث خواهیم نمود و اثر میدان‌های چند آدرسه در یک دستور مورد بررسی قرار خواهند گرفت.

اعمال مشخص شده توسط دستورات کامپیوتر بر روی داده‌های ذخیره شده در حافظه یا ثبات‌های پردازنده اجرا می‌شود. عملوندهای موجود درحافظه توسط آدرس حافظه مشخص می‌شوند. عملوندهایی

1- Scientific Calculator 2- Field 3- Mode

که در ثبات‌های پردازشگر قرار دارند توسط آدرس ثبات معین می‌گردند. یک آدرس ثبات یک عدد k بیتی است که 2^k ثبات را در CPU تعریف می‌کند. بنابراین یک CPU با 16 ثبات R_0 تا R_{15} دارای میدان آدرس ثبات چهاربیتی است. مثلاً عدد دودویی 0101 ثبات R_5 را مشخص می‌کند.

کامپیوترها ممکن است دستوراتی با طول‌های مختلف داشته و تعداد آدرس‌های آن‌ها متفاوت باشد. تعداد میدان‌های آدرس در قالب دستور به سازمان داخلی ثبات‌های آن بستگی دارد. اغلب کامپیوترها یکی از سه نوع CPU زیر را استفاده می‌کنند:

۱- سازمان تک انباره‌ای
۲- سازمان چند ثباتی
۳- سازمان پشته‌ای

مثالی از نوع تک انباره‌ای همان کامپیوتر فصل ۵ است. تمام اعمال بکمک یک انباره صورت می‌گیرد. قالب دستور در این نوع کامپیوترها از یک آدرس استفاده می‌نماید. مثلاً، دستوری که یک جمع را تعریف می‌کند بصورت زبان اسمبلی زیر می‌باشد.

```
ADD    X
```

که X آدرس عملوند است. دستور ADD در این حالت طبق عبارت زیر عمل می‌نماید

$$AC \leftarrow AC + M[X]$$

AC ثبات انباره و M[X] کلمه حافظه واقع در آدرس X را مشخص می‌نماید.

مثالی از نوع چند ثباتی در شکل ۱-۷ نشان داده شد. قالب دستور در این نوع کامپیوترها به سه میدان آدرس ثبات نیاز دارد. بنابراین دستور برای یک عمل جمع بفرم اسمبلی بشکل زیر می‌تواند نوشته شود

```
ADD    R1, R2, R3
```

که عمل $R_1 \leftarrow R_2 + R_3$ را تداعی می‌کند. تعداد میدان‌های آدرس در دستور از سه به دو قابل تقلیل است بشرطی که ثبات مقصد یکی از دو ثبات مبدأ باشد. بنابراین دستور

```
ADD    R1, R2
```

عمل $R_1 \leftarrow R_2 + R_1$ را تداعی خواهد کرد. در این دستور فقط آدرس ثبات‌های R_1 و R_2 مورد نیازند.

کامپیوترهایی که ثبات‌های متعددی را دارا هستند از دستور انتقال (move) با نماد خلاصه MOV استفاده می‌کنند. با این ترتیب دستور

```
MOV    R1, R2
```

بیانگر عمل $R_1 \leftarrow R_2$ (یا $R_2 \leftarrow R_1$ که به نوع کامپیوتر بستگی دارد) می‌باشد. بنابراین این دستورات به دو میدان آدرس برای تعیین مبدأ و مقصد نیاز دارند.

کامپیوترهای چند ثباتی دو یا سه میدان آدرس را در ساختمان دستورالعمل بکار می‌برند. هر میدان

آدرس یک ثبات پردازنده یا آدرس حافظه را معین می‌نماید. دستور زیر :

ADD R_1, X

مربوط به عمل $R_1 \leftarrow R_1 + M[X]$ است. این دستور دو میدان آدرس یکی برای R_1 و دیگری برای آدرس حافظه X دارد.

CPU سازمان یافته با پشته در شکل ۴-۸ ارائه شد. کامپیوترهایی از این نوع دارای دستورات پوش و پاپ می‌باشند که تنها یک میدان آدرس برای آنها نیاز است. بنابراین دستور

PUSH X

کلمه‌ای در آدرس X را دربالای پشته می‌نشاند. اشاره‌گر پشته بطور اتوماتیک تصحیح می‌گردد. دستورات محاسباتی در کامپیوترهای سازمان یافته با پشته احتیاجی به آدرس ندارند زیرا اعمال مزبور بر روی دو کمیت که در بالای پشته قرار دارند اجرا می‌شود.

دستورالعمل ADD در کامپیوتر پشته‌ای متشکل از یک کد عمل ولی بدون آدرس است. این عمل وظیفه بازگرفتن [1] دو عدد در بالای پشته را بعهده داشته و پس از جمع آنها، حاصل را در پشته می‌نشاند [2] در این دستور نیازی به عملوندی همراه با میدان آدرس نیست زیرا تمام عملوندها در پشته واقعند.

همانطور که اشاره شد اکثر کامپیوترها در یکی از سه سازمان فوق قرار می‌گیرند. برخی کامپیوترها ترکیبی از چند سازمان را دارا هستند. مثلاً ریزپردازنده 8080 از Intel دارای هفت ثبات در CPU است که یکی از آنها انباره است. در نتیجه پردازشگر دارای برخی خواص چند ثباتی و بعضی خواص نوع انباره‌ای است. تمام دستورات حساب و منطق به همراه دستورات ذخیره و بارکردن از ثبات انباره استفاده می‌نمایند. بنابراین دستورات مذکور دارای یک میدان آدرس هستند. از طرف دیگر دستوراتی که داده‌ها را بین ثبات‌ها نقل و انتقال می‌دهند دارای دو میدان آدرس برای ثبات هستند. بعلاوه پردازنده Intel 8080 دارای یک اشاره‌گر پشته بوده و دستوراتی برای نشاندن و بازگرفتن داده از پشته حافظه‌ای دارد. با این وجود پردازشگر دارای دستورات نوع آدرس صفر [3] که از مشخصه‌های CPU با سازمان پشته است نمی‌باشد.

برای تشریح تأثیر مقدار آدرس‌ها در برنامه کامپیوتر عبارت زیر را ارزیابی می‌کنیم

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

در این بحث دستورات صفر، یک، دو و سه آدرسی بکار خواهند رفت. ما دستورات ADD ، SUB ، MUL و DIV را برای چهار عمل اصلی بکار خواهیم برد و از دستور MOV برای انتقال و LOAD و STORE برای بار و ذخیره کردن بین حافظه و انباره AC استفاده می‌نمائیم. همچنین فرض می‌کنیم عملوندها در آدرسهای A و B و C و D بوده و نتیجه در آدرس X قرار گیرد.

1- Pop 2- Push

3- Zero – address type instruction

دستورات سه آدرسه

کامپیوترهایی که قالب دستورات آن‌ها سه آدرسه است هرمیدان آدرس را برای مشخص نمودن ثبات پردازشگر یا عملوندی در حافظه می‌توانند بکار برند. برنامه‌ای در زبان اسمبلی که عبارت X = (A+B) * (C+D) را محاسبه کند به همراه توضیحات مربوط به انتقال از ثباتهای هر دستور در زیر آمده است.

ADD	R_1, A, B	$R_1 \leftarrow M[A] + M[B]$
ADD	R_2, C, D	$R_2 \leftarrow M[C] + M[D]$
MUL	X, R_1,R_2	$M[X] \leftarrow R_1 * R_2$

فرض براین است که پردازشگر دارای دو ثبات R_1 و R_2 است. نماد M[A] بمعنی وجود عملوند در آدرس A است.

مزیت دستورات سه آدرسه کوتاهی برنامه در ارزیابی عبارات محاسباتی است. عیب این دستورات تخصیص تعداد زیاد بیت برای معین کردن سه آدرس است. نمونه‌ای از کامپیوترهای تجاری که اینگونه دستورات را بکار می‌برند Cyber 170 است. قالب دستورات در کامپیوتر Cyber محدود به سه آدرس ثبات یا دو آدرس ثبات و یک آدرس حافظه است.

دستورات دو آدرسه

دستورات دو آدرسه در کامپیوترهای تجاری بسیار مرسوم هستند. در اینجا نیز هر میدان آدرس یک ثبات پردازشگر یا یک کلمه حافظه را معین می‌کنند. برنامه ارزیابی عبارت X=(A+B) * (C+D) بفرم زیر است.

MOV	R_1, A	$R_1 \leftarrow M[A]$
ADD	R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV	R_2, C	$R_2 \leftarrow M[C]$
ADD	R_2, D	$R_2 \leftarrow R_2 + M[D]$
MUL	R_1, R_2	$R_1 \leftarrow R_1 * R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

دستور MOV، عملوند را ازو یا به حافظه وثبات‌های پردازنده انتقال می‌دهد. اولین نمادی که در دستور آمده است هم یکی از منابع و هم مقصد مربوط به نتیجه عمل است.

دستورات یک آدرسه

دستورات یک آدرسه از انباره AC برای تمام دستکاری‌های روی داده استفاده می‌کنند. ضرب و تقسیم نیاز به یک ثبات دیگر نیز دارد. با این وجود، ما از این ثبات دوم صرفنظر کرده و فرض خواهیم کرد که AC

قادر است نتایج تمام عملیات را در خود جای دهد. برنامه ارزیابی X=(A+B)*(C+D) بقرار زیر است:

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

دیده می‌شود که تمام عملیات بین انباره AC و عملوند حافظه صورت گرفته است. T آدرس یک مکان حافظه موقت است تا نتایج میانی را ذخیره نماید.

دستورات صفرآدرسه

یک کامپیوتر سازمان یافته با پشته، میدان آدرس را برای دستورات ADD و MUL بکار نمی‌برد. با این وجود دستورات پوش و پاپ برای تعیین عملوندی که با پشته تبادل اطلاعات می‌کند نیاز به آدرس دارند. برنامه زیر نشان می‌دهد که چگونه X=(A+B) * (C+D) بصورت یک برنامه برای این نوع کامپیوتر درآمده است (TOS بمعنی بالای پشته[1] است)

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

برای ارزیابی عبارات حسابی در کامپیوتر پشته‌ای، لازم است تا عبارات بشکل RPN در آیند. بعلت عدم وجود میدان آدرس در این نوع کامپیوتر نام صفر آدرسه به آنها اتلاق شده است.

دستورات RISC

مزایای یک معماری کامپیوتر کم دستور[2] (RISC) در بخش ۸-۸ تشریح شده است. مجموعه دستورات پردازشگر RISC به هنگام تبادل اطلاعات CPU با حافظه به LOAD و STORE محدود

1-Top of stack

2- Reduced Instruction Set Computer

می‌شود. سایر دستورات در داخل CPU و بدون ارجاع به حافظه انجام می‌شود. همانطور که اشاره شد برنامه‌ها برای CPUهای نوع RISC از دستورات LOAD و STORE تشکیل یافته‌اند که دارای یک آدرس حافظه و یک آدرس ثبات‌اند ولی دستورات نوع محاسباتی سه آدرس ثبات از کل ثبات‌های پردازشگر را مشخص می‌نمایند. برنامه زیر عبارت $X = (A+B) * (C+D)$ را مشخص می‌نماید.

LOAD	R_1 , A	$R_1 \leftarrow M[A]$
LOAD	R_2 , B	$R_2 \leftarrow M[B]$
LOAD	R_3 , C	$R_3 \leftarrow M[C]$
LOAD	R_4 , D	$R_4 \leftarrow M[D]$
ADD	R_1 , R_1,R_2	$R_1 \leftarrow R_1 + R_2$
ADD	R_3 , R_3, R_4	$R_3 \leftarrow R_3 + R_4$
MUL	R_1 , R_1 ,R_3	$R_1 \leftarrow R_1 * R_3$
STORE	X , R_1	$M[X] \leftarrow R_1$

دستور LOAD عملوندها را از حافظه به ثبات‌های CPU انتقال می‌دهد. دستورات ADD و MOV هم داده‌های درون ثباتها را بدون دستیابی به حافظه اجرا می‌نمایند. سپس نتیجه محاسبات با استفاده از دستور STORE در حافظه ذخیره می‌گردد.

۵-۸ روشهای آدرس دهی [1]

میدان عمل یک دستور، عملی را که قرار است انجام شود مشخص می‌نماید. این عمل بر روی داده‌هایی که در ثبات‌ها و یا حافظه‌ها ذخیره شده‌اند اجرا می‌شود. روش انتخاب عملوندها در حین اجرای برنامه به روش آدرس دهی دستور وابسته است. روش آدرس دهی، قاعده‌ای را برای تفسیر و تصحیح آدرس دستورالعمل مشخص می‌کند. کامپیوترها تکنیک روشهای آدرس دهی را به یک یا هر دو دلیل زیر بکار می‌برند.

۱- به کاربر، بدلیل در اختیار داشتن اشاره‌گر، شمارنده‌های حلقه، اندیس دهی داده‌ها و تغییر مکان برنامه انعطاف پذیری لازم را می‌دهند.

۲- تعداد بیت‌های میدان آدرس دستورالعمل را کاهش می‌دهند.

در اختیار داشتن روشهای آدرس دهی به برنامه نویسان زبان اسمبلی با تجربه، انعطاف لازم را ارائه می‌دهد تا از نظر تعداد دستورات و زمان اجرا برنامه‌های بهتری را تهیه نمایند.

1- Addressing modes

برای شناخت انواع روشهای آدرس دهی که در این بخش ارائه خواهد شد، بهتر است ابتدا سیکل عملیات یک کامپیوتر را بدانیم. واحد کنترل یک کامپیوتر به گونه‌ای طراحی شده تا وارد یک سیکل دستورالعمل متشکل از سه فاز عمده زیر گردد.

۱- برداشت[1] دستورالعمل از حافظه

۲- دیکدکردن دستور

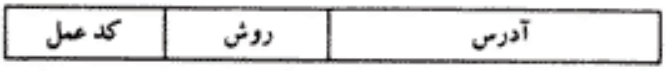
۳- اجرای[2] دستور

در کامپیوتر ثباتی بنام شمارنده برنامه[3] (یا PC) قرار دارد که دستورات یک برنامه ذخیره شده در حافظه را دنبال می‌کند. PC آدرس دستوری را که قرار است بعداً اجرا شود نگهداشته و هر وقت دستوری از حافظه برداشته شد یک واحد به محتوای آن اضافه می‌شود. دیکد شدن در مرحله ۲، عملی را که باید انجام شود، روش آدرس دهی دستور و مکان عملوند را مشخص می‌نماید. سپس کامپیوتر دستور را اجرا نموده و به مرحله ۱ برای برداشت دستور بعدی باز می‌گردد.

در برخی از کامپیوترها روش آدرس دهی دستور توسط یک کددودویی مجزا، درست مثل کدعمل مشخص می‌شود. دسته‌ای دیگر از کامپیوترها از یک کد برای تعیین و روش آدرس دهی دستور استفاده می‌نمایند. دستورات ممکن است با انواع روشهای آدرس دهی تعریف شوند و گاهی اوقات دو یا سه روش آدرس دهی در یک دستور بصورت ترکیب بکار می‌روند.

مثالی از قالب دستورالعمل با میدان روش آدرس دهی مجزا در شکل ۶-۸ نشان داده شده است. کد عمل، عملی راکه قرار است اجرا شود مشخص می‌نماید. میدان روش برای تعیین مکان عملوند بکار می‌رود. در دستورالعمل ممکن است میدان آدرس وجود داشته و یا نداشته باشد. اگر میدان آدرس وجود داشته باشد متعلق به آدرس حافظه و یا ثبات پردازشگر است .بعلاوه همانطور که در بخش قبل گفته شد، دستور ممکن است بیش ازیک میدان آدرس داشته باشد و هر میدان می‌تواند روش آدرس دهی خاص خود را دارا باشد.

روش ضمنی: در این روش آدرس دهی عملوندها در تعریف خود دستور مستترند. مثلاً دستور "مکمل کردن انباره " یک دستور از نوع ضمنی است زیرا عملوند موجود درانباره در تعریف دستور آمده است. در واقع تمام دستورات ارجاع به ثبات که از ثبات انباره استفاده می‌کنند از این دسته‌اند.



شکل ۶-۸ قالب دستورالعمل با میدان روش (شیوه)

1- Fetch 2-Execute 3- Program Counter

دستورات صفر آدرسه در کامپیوترهای پشته‌ای نیز از نوع روش ضمنی هستند زیرا عملوندها در بالای پشته قرار دارند و برای آنها نیازی به آدرس نیست.

روش بلافصل: در این روش عملوند جزئی از دستورالعمل است. به بیان دیگر یک دستورالعمل از این نوع بجای میدان آدرس دارای میدان عملوند است. میدان عملوند، خود عملوندی است که باید همراه با کد عمل در دستور مورد استفاده قرار گیرد. دستورات بلافصل برای مقداردهی اولیه ثبات‌ها با یک عدد ثابت مناسبند.

قبلاً ذکر شده که میدان آدرس دستورالعمل ممکن است یک حافظه یا یک ثبات پردازشگر را مشخص نماید. اگر میدان آدرس، یک ثبات را معین کند گوئیم دستور از نوع روش ثباتی است.

روش ثباتی: در این روش عملوندها در ثبات‌های درون CPU قرار گرفته‌اند. ثبات مورد نظر از میدان آدرس دستور انتخاب می‌شود. یک میدان k بیت می‌تواند 2^k ثبات را مشخص نماید.

روش غیرمستقیم ثباتی: در این روش دستورالعمل، ثباتی را در CPU مشخص می‌سازد که محتوای آن آدرس عملوند در حافظه می‌باشد. به بیان دیگر ثبات انتخابی بجای عملوند، حاوی آدرس عملوند می‌باشد. قبل از استفاده از این نوع دستورالعمل، برنامه‌نویس باید مطمئن شود که آدرس حافظه قبلاً در ثبات قرار گرفته است. مزیت این روش این است که میدان آدرس ثبات، بیت‌های کمتری را در مقایسه با روش آدرس دهی مستقیم در برنامه بکار می‌برد.

روش خود افزایش یا خودکاهشی: این روش مشابه روش قبلی است با این تفاوت که پس از هر بار استفاده از آدرس حافظه مقدار آن در ثبات بطور خودکار افزایش یا کاهش می‌یابد. وقتی که آدرس ذخیره شده در ثبات به جدولی از داده‌ها در حافظه ارجاع داده شود، لازم است تا ثبات پس از هر دستیابی یکبار افزایش یا کاهش یابد. این تصحیح با بکارگیری دستورات افزایش یا کاهش میسر است. با این وجود بعلت استفاده‌های مکرر، برخی کامپیوترها از امکان خاصی سود می‌برند که توسط آن پس از هر دستیابی محتوای ثبات را بطور خودکار افزایش یا کاهش می‌دهند.

میدان آدرس یک دستور توسط واحد کنترل استفاده می‌شود تا عملوند را از حافظه بدست آورد. گاهی اوقات مقدار داده شده در میدان آدرس، آدرس عملوند است و گاهی نیز آدرسی است که با استفاده از آن آدرس عملوند محاسبه می‌گردد. برای تفکیک انواع روشهای آدرس دهی لازم است تا بخش آدرس دستورالعمل مورد تفحص قرار گرفته و آدرس مؤثر که بوسیله کنترل در زمان اجرای دستور استفاده می‌شود معین می‌گردد. آدرس مؤثر[1] آدرس حافظه‌ای است که ازمحاسبه‌ای که روش آدرس دهی دیکته

1- Effective Address

می‌کند حاصل می‌شود. در دستورات محاسباتی آدرس مؤثر همان آدرس عملوند است. در دستورات انشعابی این آدرس، آدرسی است که کنترل به آن انشعاب می‌نماید. ما در فصل ۵ دو روش آدرس دهی را معرفی کردیم. در اینجا مجدداً آنها را بطور خلاصه یادآور می‌شویم.

روش آدرس دهی مستقیم[1]: در این روش آدرس موثر همان بخش آدرس دستورالعمل است. عملوند واقع در حافظه مستقیماً توسط این آدرس مشخص می‌شود. در دستورات نوع انشعاب، میدان آدرس، آدرس واقعی انشعاب را تعیین می‌نماید.

روش آدرس‌دهی غیرمستقیم:[2] در این روش میدان آدرس دستورالعمل مکان آدرس مؤثر را که در حافظه است مشخص می‌نماید. کنترل دستور را از حافظه برداشت و بخش آدرس آنرا برای دستیابی به حافظه و لذا یافتن آدرس مؤثر می‌خواند. روش آدرس دهی غیرمستقیم قبلاً در بخش ۱-۵ تشریح شد.

در چند روش آدرس دهی لازم است تا میدان آدرس دستور به محتوای ثبات خاصی در CPU اضافه شود. در این روشها آدرس موثر از طریق زیر حاصل می‌گردد.

آدرس موثر = بخش آدرس دستور + محتوای ثبات در CPU

ثبات بکار رفته در محاسبه فوق می‌تواند شمارنده برنامه، یک ثبات شاخص (یا اندیس) یا یک ثبات پایه باشد. در هر حالت ما روش آدرس دهی مختلفی خواهیم داشت که در کاربردهای مختلف مورد استفاده قرار گیرد.

روش آدرس دهی نسبی:[3] در این روش معمولاً شمارنده برنامه به بخش آدرس دستور اضافه می‌شود تا آدرس موثر بدست آید. بخش آدرس دستور معمولاً یک عدد علامت دار است (بفرم مکمل 2) که می‌تواند مثبت یا منفی باشد. وقتی که این عدد به محتوای شمارنده برنامه اضافه شود حاصل آدرس مؤثری است که مکانی را در حافظه نسبت به دستورالعمل بعدی معین می‌سازد. بعنوان مثال، فرض کنید که شمارنده برنامه دارای عدد 825 باشد و بخش آدرس دستور 24 در نظر گرفته شود. دستور واقع در مکان 825 در طول فاز برداشت ازمکان 825 خوانده شده و شمارنده سپس به 826 افزایش می‌یابد. آدرس موثر در این روش آدرس دهی برابرست 850 = 24 + 826. این مکان 24 خانه حافظه بعداز دستور بعدی است. آدرس دهی نسبی اغلب همراه دستورات نوع انشعابی و بهنگام قرار گرفتن آدرس انشعاب در حول و حوش خود دستورالعمل بکار می‌رود. این روش میدان آدرس کوچکتری را در ساختار دستورالعمل می‌طلبد زیرا آدرس نسبی به تعداد بیت‌های کمتری در مقایسه با کل آدرس حافظه نیاز دارد.

1- Direct Address Mode 2- Indirect Address Mode 3- Relative Address Mode

روش آدرس دهی شاخص دار:[1] در این روش محتوای یک ثبات شاخص[2] به بخش آدرس دستور اضافه می‌شود تا آدرس موثر بدست آید. ثبات شاخص، ثبات خاصی از CPU است که مقدار شاخص برای کمیت‌ها را در خود دارد. میدان آدرس دستورالعمل آدرس شروع یک آرایه داده را در حافظه تعریف می‌کند. هر عملوند در آرایه در حافظه‌ای نسبت به آدرس شروع ذخیره می‌گردد. فاصله بین آدرس شروع آرایه و آدرس عملوند، مقدار آدرس ذخیره شده در ثبات شاخص است. هر عملوندی در آرایه با یک دستور قابل دسترسی است بشرطی که ثبات شاخص دارای مقدار صحیح شاخص باشد. ثبات شاخص می‌تواند افزایش یابد تا عملوندهای متوالی را در دسترس قرار دهد. توجه داشته باشید که اگر این دسته دستورات دارای میدان آدرس در قالب دستورالعمل نباشند، دستور به آدرس دهی غیرمستقیم ثبات تبدیل می‌شود.

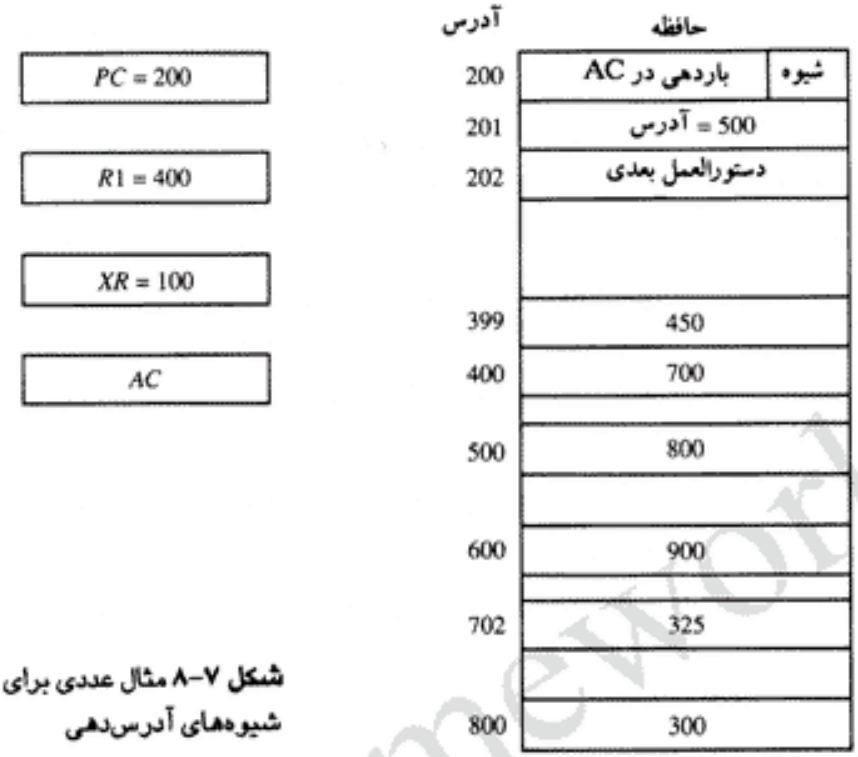
برخی از کامپیوترها یک ثبات بتنهایی برای شاخص اختصاص می‌دهند. این ثبات بهنگام استفاده از دستورات آدرس دهی شاخص دار در عملیات دخالت می‌نماید. در کامپیوترهایی که چندین ثبات پردازنده دارند هر یک از ثبات‌ها می‌توانند حاوی عدد شاخص باشند. در چنین حالتی ثبات را باید در داخل قالب دستور مشخص کرد.

روش آدرس دهی با ثبات پایه: در این روش محتوای یک ثبات پایه به بخش آدرس دستور اضافه می‌شود تا آدرس موثر بدست آید. این روش مشابه روش آدرس دهی شاخص دار است با این اختلاف که ثبات در اینجا ثبات پایه نام دارد. اختلاف بین دو روش بیشتر روش استفاده از آنهاست تا روش محاسبه آنها. فرض بر این است که ثبات شاخص، عدد شاخص را نسبت به بخش آدرس دستور درخود نگه می‌دارد. یک ثبات پایه آدرس مبنایی را نگه می‌دارد و میدان آدرس دستور جابجایی نسبت به این آدرس را معین می‌نماید. روش آدرس دهی ثبات پایه در کامپیوتر برای تسهیل در تغییر مکان برنامه در حافظه مورد استفاده است. وقتی که برنامه ها و داده‌ها از یک قسمت از حافظه به قسمت دیگری از حافظه جابجا می‌شوند، مثل سیستم‌های چند برنامه‌ای[3]، مقادیر آدرس دستورات باید این تغییر را انعکاس دهند. با یک ثبات پایه، مقادیر جابجایی دستورات نیاز به تغییر ندارند. و این فقط مقدار ثبات پایه است که نیاز به تصحیح دارد تا منعکس کننده ابتدای قطعه جدیدی از حافظه باشد.

مثال عددی

برای نشان دادن تفاوت‌های بین روشهای مختلف آدرس دهی ، ما تأثیر آدرس دهی را روی دستورات معرفی شده در شکل ۷-۸ نشان می‌دهیم. دستور دو کلمه‌ای واقع در آدرس‌های 200 و 201 یک دستور "بارکردن AC" با میدن آدرس 500 می‌باشد. اولین کلمه دستور کدعمل و روش آدرس دهی و دومین

1- Indexed Addressing Mode
2- Index Register
3- Multipragramming



شکل ۷-۸ مثال عددی برای شیوه‌های آدرس‌دهی

کلمه آدرس می‌باشد. PC دارای عدد 200 برای دریافت این دستور است. محتوای ثبات پردازشگر R_1 برابر 400 و محتوای ثبات اندیس XR نیز 100 است. AC عملوند را پس از اجرای دستور برداشت می‌نماید. در شکل مزبور چند آدرس مرتبط با مثال لیست شده و محتوای حافظه هر یک از آدرس‌های مذکور را نشان می‌دهد.

میدان روش[1] دستور می‌تواند هر یک از روش‌ها را مشخص کند. برای هر روش ممکن ما آدرس موثر را محاسبه خواهیم کرد و عملوندی را که باید در AC بار شود معین می‌نمائیم. در روش آدرس دهی مستقیم، آدرس موثر همان بخش آدرس دستور است و عملوندی که باید در AC بار شود 800 است. در روش بلافصل، بخش دوم دستور بجای اینکه آدرس باشد عملوند تلقی می‌شود. بنابراین 500 در AC بارمی‌گردد. (آدرس مؤثر در اینحالت 201 است). در روش غیرمستقیم آدرس مؤثر در آدرس 500 ذخیره شده است. بنابراین آدرس موثر 800 و عملوند 300 می‌باشد. در روش نسبی آدرس موثر برابر است با 702=202+500 و عملوند 325 می‌باشد. (توجه کنید که مقدار داخل PC بعداز فازبرداشت و در طول فاز اجرا 202 است) در روش آدرس دهی شاخص دار، آدرس موثر برابرست با 600=500+100=XR+500 و عملوند هم 900 است. در روش ثبات عملوند در R_1 و عدد 400 و برابر با محتوای R_1 است و عملوندی که در AC بار می‌شود 700 می‌باشد. روش خود افزایش مشابه روش ثباتی غیرمستقیم است با این اختلاف که پس

1- Mode field

جدول ۴-۸ جدول مقادیر مثال عددی

محتوای AC	آدرس موثر	آدرس دهی
800	500	آدرس مستقیم
500	201	عملوند بلافصل
300	800	آدرس غیرمستقیم
325	702	آدرس نسبی
900	600	آدرس شاخص‌دار
400	—	ثباتی
700	400	ثباتی غیرمستقیم
700	400	خودافزایش
450	399	خودکاهش

از اجرا، R_1 افزایش و به 401 تغییر یافته است. در روش خودکاهش، قبل از اجرای دستور، R_1 را به 399 تقلیل می‌دهد. عملوند بارشده در AC برابر 450 است. جدول ۴-۸ مقادیر آدرس موثر و عملوند بارشده در AC را برای 9 روش آدرس دهی لیست نموده است.

۸-۶ انتقال و دستکاری داده‌ها

کامپیوترها مجموعه دستورات قابل توجهی را برای ایجاد انعطاف پذیری اعمال محاسباتی در اختیار کاربر قرار می‌دهند. مجموعه دستورالعمل‌های کامپیوترها از نظر روش تعیین عملوندها با یکدیگر اختلاف دارند. ولی اعمال واقعی انجام شده بوسیله دستورات از یک کامپیوتر به کامپیوتر دیگر با هم تفاوت چندانی ندارند. اغلب دیده می‌شود کد دودویی تخصیص یافته به دستورات در کامپیوترها مختلفند و این حتی برای دستورات مشابه با هم نیز دیده می‌شود. همچنین ممکن است نام سمبلیک بکار رفته در زبان اسمبلی در کامپیوترهای مختلف متفاوت باشد. در هر صورت در بیشتر کامپیوترها مجموعه‌ای از دستورات دیده می‌شوند که اعمال ساده و پایه‌ای را انجام می‌دهند. مجموعه دستورات اصلی در نمونه‌ای از کامپیوترها موضوع مورد بحث در این بخش و بخش بعدی است.

اکثر دستورالعمل‌های کامپیوتر را می‌توان به سه دسته زیر تقسیم کرد:

۱- دستورالعمل‌های انتقال داده[1]

۲- دستورالعمل‌های دستکاری داده‌ها[2]

۳- دستورات کنترل برنامه

دستورالعمل‌های انتقال داده موجب انتقال داده‌ها از یک مکان به مکان دیگر بدون هر گونه تغییر در محتوای اطلاعات دودویی آنها می‌گردند. دستورالعمل‌های دستکاری داده‌ها دستوراتی هستند که اعمال حسابی، منطقی وشیفت را انجام می‌دهند. دستورالعمل‌های کنترل برنامه قابلیت‌های تصمیم‌گیری را

1- Data Transfer 2- Data Manipulation

فراهم می‌آورند و مسیر برنامه را در حین اجرای برنامه تغییر می‌دهند. مجموعه دستورالعمل‌های یک کامپیوتر بخصوص، اعمال انتقال ثبات و تصمیمات کنترلی موجود را برای استفاده کننده معین می‌نمایند.

دستورالعمل‌های انتقال داده

دستورالعمل‌های انتقال داده، انتقال داده‌ها را از یک مکان در کامپیوتر به مکانی دیگر بدون هر گونه تغییری در آنها انجام می‌دهد. متداولترین انتقال‌ها بین حافظه و ثبات‌های پردازشگر صورت می‌گیرد. جدول ۵-۸ لیست هشت دستورالعمل انتقال داده را که در بسیاری از کامپیوترها بکار می‌رود نشان می‌دهد. همراه باهر دستور یک سمبل اختصاری ارائه شده است. باید دانست که کامپیوترهای مختلف سمبل‌های اختصاری مختلفی را استفاده می‌کنند.

دستور LD به معنی بارکردن[1] اکثراً به هنگام انتقال داده از حافظه به یک ثبات پردازنده که معمولاً انباره است به کار می‌رود. دستور ST به معنی ذخیره کردن[2]، انتقال از یک ثبات پردازنده به حافظه را مشخص می‌کند. دستورالعمل MOV به معنی انتقال[3]، در کامپیوتر چند ثباتی برای انتقال داده از ثباتی به ثبات دیگر مورد استفاده قرار می‌گیرد. این دستور برای انتقال داده‌ها بین ثبات‌های CPU و حافظه و نیز بین دو مکان حافظه هم بکار می‌رود. دستور XCH به معنی تعویض[4]، اطلاعات را بین دو ثبات یا بین یک ثبات و یک حافظه عوض می‌کند. دستورات IN و OUT به معنی ورودی[5] و خروجی[6]، داده‌ها را بین ثبات‌ها و پایانه‌های ورودی یا خروجی جابجا می‌کنند. دستورات PUSH (پوش) و POP (پاپ) به ترتیب به معنی فشردن (نشاندن) و کشیدن (بازگرفتن)، داده‌ها را بین ثبات‌های پردازشگر و پشته‌ای در حافظه انتقال می‌دهند.

باید دانست که دستورات جدول ۵-۸ و نیز جداول بعدی در این بخش، اغلب با انواع روشهای آدرس دهی همراه‌اند. گاهی در زبان اسمبلی بطور قراردادی تغییری در سمبل‌های اختصاری داده می‌شود که منعکس کننده شیوه آدرس دهی می‌باشد. مثلاً سمبل اختصاری دستور load immediate به معنی

جدول ۵-۸ دستورالعمل‌های انتقال داده‌ها

سمبل اختصاری	نام
LD	بارکردن
ST	ذخیره
MOV	انتقال
XCH	تبادل
IN	ورودی
OUT	خروجی
PUSH	پوش (فشردن)
POP	پاپ (برداشتن)

1- Load
2- Store
3- Move
4- Exchange
5- Input
6- Output

بار کردن بلافصل بصورت LDI درمی‌آید. گاهی نیز قرار داد زبان اسمبلی این است که از کاراکتر خاصی برای مشخص کردن روش آدرس دهی استفاده شود. مثلاً روش بلافصل باقراردادن علامت # قبل از عملوند مشخص می‌شود. در هر حال نکته مهمی که باید به آن توجه کرد این است که هر دستور می‌تواند با انواع روش‌های آدرس دهی همراه باشد. مثلاً دستور بارکردن انباره رابا هشت روش آدرس دهی مختلف ملاحظه کنید. جدول ۶-۸ زبان اسمبلی پیشنهادی و انتقال واقعی داده راکه در هر مورد صورت می‌گیرد نشان می‌دهد. ADR به معنی آدرس، NBR بمعنی یک عدد یا عملوند، X یک ثبات شاخص، R1 یک ثبات پردازنده و AC ثبات انباره است. کاراکتر @ سمبل آدرس غیرمستقیم است. کاراکتر $ قبل از آدرس، آنرا به درون PC نسبت می‌دهد. کاراکتر # در دستورات بلافاصل قبل از عملوند قرارداده می‌شود. دستورات شاخص‌دار با ثباتی در درون پرانتز که پس از آدرس قرار می‌گیرد شناخته می‌شود. در آدرس‌دهی ثباتی نام ثبات حاوی عملوند جلو نام اختصاری دستور قرار می‌گیرد. در آدرس دهی ثباتی غیرمستقیم نام ثباتی که حاوی آدرس است داخل پرانتز قرار داده می‌شود. در روش خود افزایشی با قرار دادن علامت + بعدازپرانتز محصورکننده ثبات از روش ثباتی غیرمستقیم متمایز می‌شود. در روش خود کاهشی هم از علامت – استفاده می‌گردد. برای نوشتن برنامه‌هایی بزبان اسمبلی در کامپیوتر، لازم است که انواع دستورالعمل‌های موجود دانسته شوند و با شیوه‌ها و روش‌های آدرس‌دهی بکار رفته در آن آشنا باشیم.

دستورالعمل‌های دستکاری داده‌ها

دستورالعمل‌های دستکاری داده اعمال راروی داده‌ها انجام می‌دهند و قابلیت‌های محاسباتی کامپیوتر را فراهم می‌آورند. این دستورالعمل‌ها به سه نوع اساسی تقسیم می‌شوند.

۱- دستورالعمل‌های حسابی
۲- دستورالعمل‌های منطقی و دستکاری‌های بیتی
۳- دستورالعمل‌های شیفت

لیست دستورات دستکاری داده‌ها شبیه به لیست ریزعمل‌ها در فصل ۴ است. با این وجود باید دانست

جدول ۶-۸ هشت روش آدرس‌دهی برای دستورالعمل بارکردن

روش	قرارداد اسمبلی	انتقال ثباتی
آدرس مستقیم	LD ADR	$AC \leftarrow M[ADR]$
آدرس غیرمستقیم	LD @ADR	$AC \leftarrow M[M[ADR]]$
آدرس نسبی	LD $ADR	$AC \leftarrow M[PC + ADR]$
عملوند بلافصل	LD #NBR	$AC \leftarrow NBR$
آدرس‌دهی شاخص‌دار	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
ثباتی	LD R1	$AC \leftarrow R1$
ثباتی غیرمستقیم	LD (R1)	$AC \leftarrow M[R1]$
خودافزایش	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

که هر دستور ضمن اجرا باید وارد فاز برداشت شده و طی آن مقدار کد را از حافظه بخواند. همچنین طبق روش آدرس دهی، عملوندها نیز باید به ثبات‌های پردازشگر منتقل شوند. آخرین مرحله اجرای دستور در پردازشگر است و همین مرحله است که طبق بحث فصل ۴ توسط ریزعمل‌ها در ALU و یا در شیفت‌دهنده طبق شکل ۲-۸ اجرا شد. برخی دستورات حسابی هم نیاز به مدارات خاصی جهت پیاده‌سازی دارند.

دستورالعمل‌های حسابی

چهار عمل اصلی حساب عبارتند از جمع و تفریق، ضرب و تقسیم. بسیاری از کامپیوترها دستورالعمل‌های لازم را برای هر چهارعمل دارا هستند. برخی کامپیوترهای کوچک فقط جمع و احتمالاً دستورات تفریق را دارند. اعمال ضرب، تقسیم باید با نوشتن زیر روال‌های نرم افزاری تولید گردند. چهار عمل اصلی حسابی فوق برای حل فرمول‌ها و مسائل علمی پس از بیان آن‌ها براساس روش‌های محاسبات عددی کافی هستند.

لیستی از دستورات حسابی در جدول ۷-۸ دیده می‌شود. دستورافزایش، یک واحد به مقدار ذخیره شده در ثبات می‌افزاید. یکی از ویژگیهای دستور افزایش پس از اجرا این است که یک عدد دودویی متشکل از 1 در تمام بیت‌ها وقتی افزایش یابد به عددی تشکیل شده از تماماً 0 بدل می‌شود. بالعکس عددی با فقط 0 پس از کاهش به عددی تماماً 1 تبدیل می‌گردد.

دستورات جمع، تفریق، ضرب و تقسیم ممکن است برای انواعی از داده‌ها موجود باشند. فرض براین است که نوع داده در ثبات‌های پردازشگر در طول اجرای این دستورات حسابی در تعریف کد عمل گنجانده شده باشد. یک دستورالعمل حسابی ممکن است یک داده ممیز ثبات، ممیز شناور، دودویی و یا اعشاری، با دقت کم یا با دقت مضاعف باشد. انواع داده‌ها در فصل ۳ معرفی شدند.

جدول ۷-۸ دستورالعمل‌های حسابی

سمبل اختصاری	نام
INC	افزایش
DEC	کاهش
ADD	جمع
SUB	تفریق
MUL	ضرب
DIV	تقسیم
ADDC	جمع با نقلی
SUBB	تفریق با قرض
NEG	نفی (متمم 2)

معمولاً نمی‌توان کامپیوتری برای هر سه نوع دستور جمع پیدا کرد: یکی برای اعداد صحیح دودویی، یکی برای عملوندها با ممیز شناور و یکی هم برای اعداد دهدهی. علامت اختصاری[1] هر سه نوع دستور جمع که انواع مختلف داده‌ها را بکار می‌برند در زیر نشان داده شده است.

ADD I	جمع دو عدد صحیح دودویی
ADD F	جمع دوعدد ممیز شناور
ADD D	جمع دوعدد دهدهی بفرم BCD[2]

الگوریتم‌های اعداد صحیح، ممیز شناور و اعمال حسابی دهدهی در فصل ۱۰ ارائه شده‌اند.

تعداد بیت‌های هر ثبات محدود است بنابراین دقت نتایج اعمال حسابی محدودیت خواهد داشت. در برخی ازکامپیوترها، مدارات سخت افزاری، اعمال محاسباتی با دقت مضاعف را ممکن می‌سازند بطوری که طول عملوند تا دو کلمه حافظه می‌رسد. بعضی کامپیوترهای کوچک نیز نرم افزار خاصی را برای محاسبات با دقت مضاعف ارائه می‌نمایند. فلیپ فلاپ خاصی برای ذخیره کردن رقم نقلی یک عمل جمع در نظر گرفته می‌شود. دستور "جمع با رقم نقلی"[3] عمل جمع برروی دو عملوند بعلاوه رقم نقلی حاصل از عمل قبلی را انجام می‌دهد. مشابهاً دستور "تفریق باقرض"[4] دو عملوند را با توجه به رقم قرضی احتمالی از عمل قبل، از یکدیگر کم می‌کند. دستور نفی سازی[5]، مکمل 2 یک عدد رامشخص می‌نماید که عبارت است از معکوس کردن علامت یک عدد صحیح وقتی که بشکل مکمل 2 علامت دار در نظر گرفته می‌شود.

دستورالعمل‌های منطقی و دستکاری بیتی

دستورات منطقی اعمال دودویی را برروی رشته‌ای از بیت‌های ذخیره شده در ثبات‌ها انجام می‌دهند. این دستورات برای بیت‌های جدا از هم و یا گروهی از بیت‌ها که نمایانگر اطلاعات کد شده دودویی هستند مفیدند. دستورات منطقی هر بیت از عملوند منطقی را جداگانه در نظر گرفته و آنرا بصورت یک متغیر بول تصور می‌کند. باکاربرد صحیح دستورات منطقی امکان تغییر بیت‌ها، امکان پاک کردن گروهی از بیت‌ها یا وارد نمودن مقادیری دربیت‌های عملوند ذخیره شده در ثبات یا کلمات حافظه وجود دارد.

نمونه‌هایی از دستورات منطقی و دستکاری بیتی در جدول ۸-۸ آورده شده‌اند. دستورالعمل پاک‌کردن[6] سبب می‌شود عملوند خاصی با 0ها جایگزین شود. دستور مکمل سازی[7]، مکمل 1 عملوند را با معکوس کردن تمام بیت‌ها تولید می‌نماید. دستورات AND، OR و XOR اعمال منطقی مربوطه

1- Mnemonic
2- Binary Coded Decimal
3- Add With Carry
4- Subtract With Borrow
5- Negate
6- Clear Instruction
7- Complement Instruction

جدول ۸-۸ دستورالعمل‌های منطقی و دستکاری بیت‌ها

سمبل اختصاری	نام
CLR	پاک کردن
COM	متمم کردن
AND	AND
OR	OR
XOR	OR انحصاری
CLRC	صفر کردن نقلی
SETC	1 کردن نقلی
COMC	متمم‌کردن رقم نقلی
EI	فعال کردن وقفه
DI	غیرفعال کردن وقفه

را روی بیتهای عملوند اجرا می‌نمایند. هرچند این دستورالعمل‌ها اعمال بولی را انجام می‌دهند ولی به آنها بصورت دستکاری کننده بیت‌ها باید نگریست. سه نوع دستکاری بیتی امکان پذیراست. بیت‌هایی که موردنظر هستند می‌توانند 0 شوند ، 1 شوند و یا مکمل شوند. سه دستور منطقی فوق دقیقاً برای انجام همین اهداف مورد استفاده قرار می‌گیرند.

دستور AND برای پاک کردن یک بیت یا گزیده‌ای از بیت‌های عملوند یک دستورالعمل است. برای هر متغیر بولی x، رابطه‌های $x \wedge 0 = 0$ و $x \wedge 1 = x$ بیان می‌دارد که هر متغیری که با 0، AND شود، تولید 0 می‌نماید ولی اگر با 1، AND شود متغیر تغییری نمی‌یابد. بنابراین دستور AND می‌تواند برای پاک کردن بیت‌های یک عملوند، یا AND آن عملوند با عملوند دیگری که در مکان‌های مورد نظر 0هایی برای پاک نمودن بیت‌های متناظر در عملوند اول دارد بکار رود. دستور AND ماسک[1] (پوشش) هم خوانده می‌شود زیرا در بخش انتخاب شده‌ای از عملوند بیت‌ها را ماسک (صفر) می‌کند.

دستور OR برای نشاندن[2] (برقرارکردن) یک بیت یا گروهی ازبیت‌های عملوند مورد استفاده قرار می‌گیرد. برای هر متغیر بولی مانند x، رابطه $x + 1 = 1$ و $x + 0 = x$ بیان می‌دارد که هر متغیری با 1، OR شود کمیت 1 را تولید می‌کند ولی اگر با 0، OR شود تغییری در متغیر وجود نخواهد داشت. بنابراین دستور OR می‌تواند برروی گزیده‌ای از بیت‌های یک عملوند و با استفاده از عملوند دیگری که در بیت‌های موردنظری از آن 1 قرار گرفته عمل کرده و بیت‌های متناظر در عملوند اول را 1 نماید.

بطور مشابه، دستور XOR بر روی بیت‌های منتخبی ازعملوند مورد استفاده است. این دستور نیز با توجه به رابطه بولی $x \oplus 1 = x'$ و $x \oplus 0 = x$ عمل می‌نماید. بنابراین یک متغیر دودویی وقتی با 1، XOR شود مکمل می‌شود. مثالهای عددی که نشاندهنده سه عمل منطقی است در بخش (۵-۴) داده شد.

چند دستور دیگر دستکاری بیت‌ها در جدول ۸-۸ دیده می‌شوند. بیت‌های خاصی همچون بیت رقم نقلی می‌توانند با دستورات مناسبی، پاک، نشانده و یا مکمل شوند. مثالی دیگر فلیپ فلاپ کنترل

1- Mask 2- Set

وقفه است که توسط دستورات دستکاری بیت‌ها توانا (فعال) و یا ناتوان (غیرفعال) می‌شود.

دستورات شیفت

دستورات شیفت، یک عملوند از جمله دستورات بسیار مفید بوده و بفرم‌های متنوعی وجود دارند. شیفت عملی است که تحت آن بیت‌های یک کلمه به چپ و یا به راست حرکت داده می‌شوند. بیتی که از انتهای کلمه وارد می‌شود نوع شیفت بکار رفته را مشخص می‌کند. دستورات شیفت ممکن است منطقی، یا حسابی و یا چرخشی باشند. در هر صورت شیفت ممکن است به راست و یا به چپ انجام شود.

جدول ۹-۸ چهار نوع دستور شیفت را نشان می‌دهد. شیفت منطقی، 0 را وارد آخرین بیت می‌نماید. آخرین بیت برای شیفت به راست، سمت چپ‌ترین و برای شیفت به چپ، راست‌ترین بیت است. شیفت حسابی معمولاً با رعایت قواعد اعداد مکمل 2 صورت می‌گیرد. این قواعد در بخش ۶-۴ آورده شدند. دستور شیفت به راست حسابی باید بیت علامت در سمت چپ ترین مکان را حفظ کند. لذا بیت علامت همراه با بقیه عدد به سمت راست شیفت داده می‌شود، ولی خود بیت علامت تغییری نمی‌کند. این یک شیفت به راست با حفظ بیت علامت است. شیفت به چپ حسابی یک 0 را به آخرین مکان وارد می‌کند و در واقع با شیفت به چپ منطقی یکی هستند. بهمین دلیل بسیاری ازکامپیوترها دستورشیفت به چپ جداگانه‌ای به هنگام وجود نوع منطقی ارائه نمی‌نمایند.

دستورات چرخش یک شیفت چرخشی را تولید می‌کنند. بیتی که از یک انتها از کلمه خارج می‌شود مفقود نشده و مجدداً از انتهای دیگر وارد می‌شود. در چرخش از طریق بیت نقلی، آنرا به منزله بیت اضافی برای ثباتی که محتوای آن چرخش داده می‌شود در نظر می‌گیرد. بنابراین چرخش به چپ همراه با رقم نقلی، بیت نقلی را به سمت راست‌ترین بیت در ثبات، و سمت چپ‌ترین بیت ثبات را به رقم نقلی انتقال می‌دهد، و همزمان با آنها کل ثبات را به چپ منتقل می‌سازد.

برخی کامپیوترها دارای قالب چند میدانی برای دستورات شیفت هستند. یک میدان مربوط به کد عمل، و بقیه نوع و تعداد شیفت‌ها را معین می‌سازد. نمونه‌ای از این دستورات می‌تواند تا پنج میدان مطابق زیر

جدول ۹-۸ دستورات شیفت

سمبل	نام
SHR	شیفت منطقی به راست
SHL	شیفت منطقی به چپ
SHRA	شیفت حسابی به راست
SHLA	شیفت حسابی به چپ
ROR	چرخش به راست
ROL	چرخش به چپ
RORC	چرخش به چپ با نقلی
ROLC	چرخش به راست با نقلی

را داشته باشد.

OP REG TYPE RL COUNT

که در آن OP عبارتست از میدان کد عمل؛ REG آدرس ثباتی است که مکان عملوند را مشخص می‌کند؛ TYPE یک میدان دوبیتی برای معرفی یکی از چهارنوع شیفت است؛ RL یک میدان یک بیتی برای چپ و یاراست بودن شیفت می‌باشد و COUNT یک میدان k بیتی برای 2^k-1 مرتبه شیفت می‌باشد. با چنین قالبی، می‌توان نوع شیفت، جهت و تعداد شیفت را در یک دستور معین کرد.

۸-۷ کنترل برنامه

دستورالعمل‌ها همواره در مکانهای متوالی حافظه ذخیره می‌شوند. بهنگام پردازش در CPU، دستورات از حافظه‌ها متوالیاً برداشت و اجرا می‌گردند. هر بار که دستوری از حافظه برداشته شد، شمارنده برنامه افزایش می‌یابد بطوری که محتوای آن آدرس دستور بعدی خواهد بود. پس از اجرای یک عمل انتقال یا دستکاری، کنترل با کمک شمارنده برنامه که آدرس بعدی را داراست برای برداشت دستور بعدی به سیکل برداشت باز می‌گردد. از طرف دیگر، دستوراتی که از نوع کنترل برنامه هستند پس از اجرا ممکن است مقدار آدرس شمارنده برنامه را تعویض و جریان کنترل را عوض نمایند. به بیان دیگر، دستورات کنترل برنامه شرایط تعویض محتوای شمارنده برنامه را مشخص می‌نمایند، در حالیکه دستورات انتقال و دستکاری داده شرایطی برای پردازش داده را فراهم می‌کنند. تغییر در مقدار شمارنده برنامه به علت اجرای دستور کنترل برنامه بسبب قطع در توالی اجرای دستورات می‌گردد. این یک ویژگی بسیار مهم در کامپیوترهای دیجیتال است که آنرا قادر می‌سازد تا کنترلی بر اجرای برنامه و انشعاب به قسمت‌های مختلف برنامه وجود داشته باشد.

برخی از انواع دستورات کنترل برنامه در جدول ۸-۱۰ آورده شده‌اند. دستورات انشعاب[1] و پرش[2] عموماً قابل تعویض بوده و یک مفهوم دارند ولی گاهی از هر دو آنها به منظور تفکیک روشهای

جدول ۸-۱۰ دستورالعمل‌های کنترل برنامه

سمبل nic	نام
BR	انشعاب
JMP	پرش
SKP	گذر
CALL	فراخوانی
RET	بازگشت
CMP	مقایسه (با تفریق)
TST	آزمون (تست) با AND کردن

1- Branch 2- Jump

آدرس‌دهی استفاده می‌شود. انشعاب معمولاً دستور یک آدرسی است و در زبان اسمبلی شکل BR ADR نوشته می‌شود که در آن ADR نام سمبلیک برای آدرس است. وقتی این دستور اجرا شود ADR به شمارنده برنامه منتقل می‌گردد. چون شمارنده برنامه آدرس دستور بعدی را در خود دارد دستور بعدی از ADR برداشته خواهد شد.

دستورات انشعاب و پرش ممکن است شرطی و یا غیرشرطی باشند. دستور انشعاب غیرشرطی سبب انشعاب به آدرسی دیگر بدون هرگونه شرطی می‌گردد. دستورات شرطی انشعاب با شرط برقراری شرایطی مانند مثبت یاصفر بودن نتیجه عملی، انشعاب ایجاد می‌نمایند. اگر شرط برقرار بود، شمارنده برنامه با آدرس انشعاب بارشده و دستور بعدی از این آدرس برداشته می‌شود. اگر شرط برقرار نشد شمارنده برنامه تغییر نکرده و دستور بعدی از مکان بعدی برداشته می‌شود.

دستور گذر[1] نیازی به میدان آدرس ندارد و بنابراین یک دستور صفر آدرسه است. یک دستور شرطی گذر دستور بعدی را بشرط برقراری یک شرط گذر می‌کند. این عمل بدین ترتیب صورت می‌گیرد که شمارنده برنامه علاوه بر افزایش در فاز دریافت، در فاز اجرا هم افزایش می‌یابد. در صورتی که شرط برقرار نباشد کنترل دستور بعدی را اجرا خواهد کرد که برنامه نویس معمولاً یک دستور انشعاب غیرشرطی را در آنجا جای می‌دهد. بنابراین یک جفت دستور انشعاب-گذر بشرطی انشعاب انجام می‌دهد که شرط برقرار نباشد در حالیکه یک انشعاب شرطی، بشرطی خواهد کرد که شرط برقرار باشد.

دستورات فراخوانی[2] و بازگشت[3] بهنگام استفاده از زیرروال[4] بکار می‌روند. در مورد این دستورات بعداً در این بخش صحبت خواهد شد. دستورات مقایسه[5] و تست[6] مستقیماً توالی اجرا برنامه را عوض نمی‌کنند. این دستورات هم بعلت کاربردشان دربرقراری شرایط برای دستور بعدی انشعاب در جدول ۸-۱۰ آورده شده‌اند. دستور مقایسه تفریقی را بین دو عملوند انجام می‌دهد ولی نتیجه عمل نگهداری نمی‌گردد. با این وجود بیت‌های وضعیت معینی بعنوان نتیجه مقدار می‌گیرند. بطور مشابه دستور تست عمل منطقی AND دو عملوند را اجرا می‌نماید و تصحیح بیت‌های وضعیت را بدون نگهداری نتیجه یا تغییر عملوند انجام می‌دهد. بیت‌های وضعیت معمولاً بیت نقلی، علامت، صفر و سرریز می‌باشند. تولید این بیت‌های وضعیت ابتدا بحث شده و سپس چگونگی کاربرد آنها در دستورات انشعاب نشان داده خواهد شد.

بیت‌های وضعیتی[7]

گاهی اوقات مناسب‌تر است تا مدار ALU در CPU را با یک ثبات وضعیت همراه کرد تا حالات بیت‌های وضعیت برای تحلیل بعدی در آن ذخیره شود.بیت‌های وضعیت گاهی اوقات بیت‌های کد

1- Skip
2- Call
3- Return
4- Subroutine
5- Compare
6- Test
7- Status Bit

شرایط[1] یا بیت‌های پرچم[2] نیز خوانده می‌شوند. شکل ۸-۸ دیاگرام یک ALU هشت بیتی به همراه یک ثبات چهاربیتی وضعیت را نشان می‌دهد. این چهار بیت وضعیت با C، S، Z و V نشان داده شده‌اند. این بیت‌ها بعنوان نتیجه یک عمل انجام شده در ALU نشانده شده (1) و یا پاک می‌شوند (0).

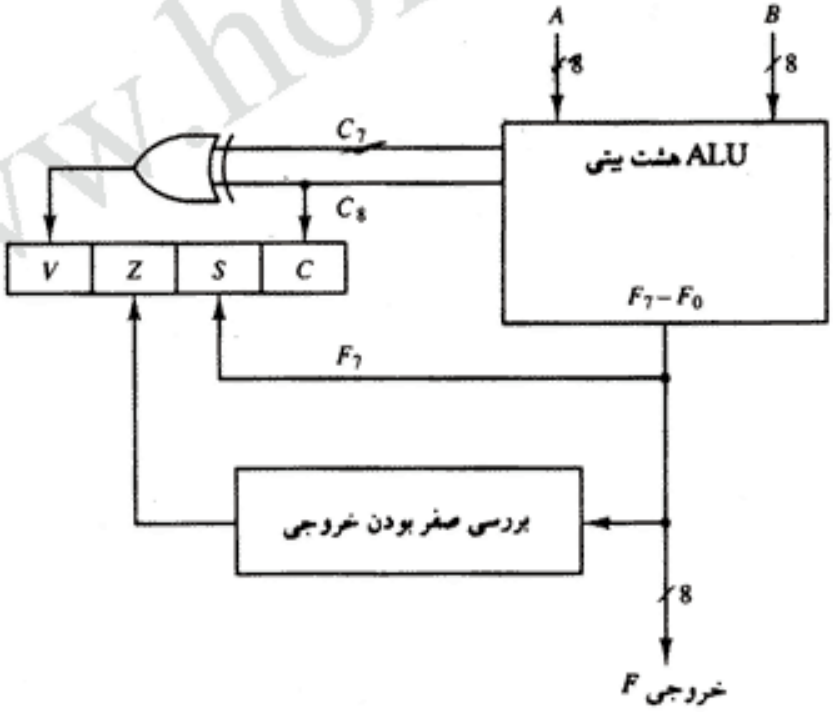
۱- بیت C (نقلی) اگر رقم نقلی انتهایی $C_8=1$ شود نشانده می‌شود (1). اگر $C_8=0$ شود این بیت 0 می‌گردد.

۲- بیت S (علامت) نشانده می‌شود بشرطی که بالا رتبه‌ترین بیت یعنی F_7 در ALU برابر 1 شود. اگر بیت مذکور 0 گردد S هم 0 شود.

۳- بیت Z[3] (صفر) هنگامی 1 می‌شود که تمام خروجی‌های ALU برابر 0 باشند. در غیر اینصورت این بیت پاک می‌شود. به بیان دیگر $Z=1$ است اگر خروجی صفر و $Z=0$ اگر خروجی غیرصفر باشد.

۴- بیت V (سرریز) نشانده می‌شود اگر، XOR دو رقم نقلی آخر برابر 1 شود ودر غیر اینصورت 0است. این شرط یک سرریز وقتی اعداد منفی بفرم مکمل 2 می‌باشند است. (بخش ۳-۳ را ببینید). برای یک ALU با هشت بیت اگر خروجی بیش از 127+ یا کمتر از 128− باشد، $V=1$ می‌گردد.

بیت‌های وضعیت پس از هر عمل در ALU می‌تواند چک شوند تا رابطه موجود بین مقادیر A و B



شکل ۸-۸ بیت‌های ثبات وضعیت

1- Condition Code Bit 2- Flag Bit 3- Zero Bit

معین گردد. اگر V پس از جمع دو عدد علامت دار نشانده شد، یک سرریز رخ داده است. اگر Z پس از یک عمل XOR نشانده شد یعنی A=B است. این بدلیل وجود رابطه $x \oplus x = 0$ است که بیان می‌دارد XOR دو عددمساوی 0 بوده و این نیز Z=1 را سبب می‌گردد. هر بیت در A نیز می‌تواند بوسیله پوشش دادن[1] سایر بیت‌ها چک شود و سپس بیت وضعیت Z ملاحظه گردد. مثلاً فرض کنید A=101x1100 باشد که در آن x بیت مورد آزمایش است. اگر x=0 باشد بیت وضعیت Z=1 می‌گردد، ولی اگر x=1 باشد Z=0 خواهد شد زیرا نتیجه غیرصفر است. عمل AND می‌تواند توسط دستور TEST، اگر حفاظت از محتویات اصلی A موردنظر باشد، صورت گیرد.

دستورات انشعاب شرطی

جدول ۱۱-۸ لیستی از دستورات انشعابی متداول را ارائه می‌دهد. هر سمبل اختصاری تشکیل شده است از حرف B (برای انشعاب) و یک مخفف از کلمه شرط. وقتی حالت مخالف بکار رود، حرف N

جدول ۱۱-۸ دستورالعمل‌های انشعاب شرطی

سمبل	شرط انشعاب	شرط مورد تست
BZ	انشعاب به شرط صفر	$Z = 1$
BNZ	انشعاب به شرط غیرصفر	$Z = 0$
BC	انشعاب به شرط نقلی	$C = 1$
BNC	انشعاب به شرط عدم نقلی	$C = 0$
BP	انشعاب به شرط مثبت	$S = 0$
BM	انشعاب به شرط منفی	$S = 1$
BV	انشعاب اگر سرریز	$V = 1$
BNV	انشعاب اگر عدم سرریز	$V = 0$
	شرایط مقایسه بی‌علامت (A−B)	
BHI	انشعاب اگر بالاتر	$A > B$
BHE	انشعاب اگر بالاتر یا مساوی	$A \geq B$
BLO	انشعاب اگر پائین‌تر	$A < B$
BLOE	انشعاب اگر پائین‌تر یا مساوی	$A \leq B$
BE	انشعاب اگر مساوی	$A = B$
BNE	انشعاب اگر نامساوی	$A \neq B$
	شرایط مقایسه علامت‌دار (A−B)	
BGT	انشعاب اگر بزرگتر	$A > B$
BGE	انشعاب اگر بزرگتر یا مساوی	$A \geq B$
BLT	انشعاب اگر کوچکتر	$A < B$
BLE	انشعاب اگر کوچکتر یا مساوی	$A \leq B$
BE	انشعاب اگر مساوی	$A = B$
BNE	انشعاب اگر نامساوی	$A \neq B$

1- Mask

(برای نه) وارد می‌شود تا وضعیت 0 را بیان نماید. بنابراین BC بمعنی انشعاب بشرط C=1 و BNC بشرط C=0 است. اگر شرط بیان شده برقرار باشد کنترل به آدرسی که توسط دستور مشخص شده منتقل می‌گردد. در غیراینصورت کنترل با دستوری که بدنبال آمده ادامه خواهد داد. دستورات شرطی می‌توانند بهمراه دستورات کنترل پرش[1]، گذر، فراخوانی و بازگشت بکار روند.

بیت وضعیت صفر (Z) برای تست نتیجه در ALU و صفربودن یا نبودن آن بکار می‌رود. بیت نقلی (C) برای وجود رقم نقلی از بالا رتبه‌ترین بیت در ALU استفاده می‌شود. این بیت به همراه دستور چرخش نیز مورد استفاده است تا بدین وسیله بیت شیفت داده شده از انتهای چرخش در آن مورد آزمایش قرار گیرد. بیت علامت (S) وضعیت بالا رتبه ترین بیت خروجی ALU را معین می‌نماید. S=0 به منزله علامت مثبت و S=1 به معنی علامت منفی است. بنابراین یک انشعاب بشرط مثبت‌بودن، بیت S را برای یافتن 0 چک می‌کند و انشعاب بشرط منفی بودن، S را برای یافتن 1 چک می‌نماید. با این وجود باید توجه داشت که این دو دستور بالارتبه ترین بیت را بدون توجه به بیت علامت بودن یا نبودن چک می‌کنند. بیت سرریز (V) به همراه اعمال حسابی انجام شده روی اعداد علامت‌دار مکمل 2 مورد استفاده قرار می‌گیرد.

همانطور که قبلاً بیان شد دستورات مقایسه، عمل تفریق دو عملوند را از هم انجام می‌دهند، یعنی A−B. ولی نتیجه عمل به ثبات مقصد منتقل نمی‌گردد. در عوض بیت‌های وضعیت تحت تأثیر قرار می‌گیرند. ثبات وضعیت اطلاعات نسبی را درباره اندازه A و B فراهم می‌نماید. برخی کامپیوترها دستورات شرطی انشعاب را که بلافاصله بعد از دستور مقایسه قابل استفاده‌اند در اختیار برنامه نویس می‌گذارند. شرایط مورد تست بستگی به علامت دار بودن یا نبودن اعداد A و B دارد. جدول ۱۱-۸ لیستی از دستورات انشعاب شرطی را نشان می‌دهد. دقت کنید که ما اصطلاحات بالاتر[2] و پائین‌تر[3] را برای اعداد بدون علامت و بزرگتر[4] و کوچکتر[5] را برای اعداد علامت دار استفاده خواهیم کرد. بنظر می‌رسد نسبت اندازه‌ها در زیر ستون سمت راست جدول برای اعداد علامت دار وبدون علامت یکی است. با این وجود این صحیح نیست زیرا هر یک باید بطور جداگانه طبق آنچه در زیر آمده است مورد بررسی قرار گیرد.

یک ALU هشت بیتی را مطابق شکل ۸-۸ در نظر بگیرید. بزرگترین عدد بدون علامتی که می‌تواند در آن جای گیرد 255 است. محدوده عدد علامت دار بین 127+ و 128− است. تفریق دو عدد چه علامت‌دار مکمل 2 و چه بدون علامت باشند یکی است (فصل ۳ ملاحظه‌شود). اجازه بدهید A=11110000 و B=00010100 باشند. در عمل A−B، ALU مکمل 2 کمیت B را با A جمع می‌کند.

```
A:        11110000
B̄ + 1:  +11101100
A − B:    11011100    C = 1    S = 1    V = 0    Z = 0
```

1- Jump
2- Higher
3- Lower
4- greater
5- Less Than

دستور مقایسه[1] بیت‌های وضعیت را طبق آنچه در فوق دیده می‌شود تصحیح می‌نماید. C=1 می‌شود زیرا یک رقم نقلی از بالاترین رتبه خارج می‌گردد. S=1 است زیرا سمت چپ‌ترین بیت 1 می‌باشد. V=0 است چون آخرین دو بیت نقلی هر دو 1 هستند و Z=0 می‌باشد چون نتیجه حاصل 0 نیست.

اگر ما اعداد را بدون علامت تصور کنیم، عدد اعشاری در A معادل 240 و B برابر 20 است. تفریق در مبنای ده 220=20−240 می‌باشد. نتیجه دودویی حاصل نیز معادل 220 است. چون 20<240 است، داریم A>B و A≠B. این دو رابطه از این واقعیت که C=1 و Z=0 است نیز استنتاج می‌گردد. دستوراتی که پس از این مقایسه‌ها می‌توانند انشعابی ایجاد نمایند، BHI (انشعاب اگر بزرگتر)، BHE (انشعاب اگر بزرگتر یا مساوی) و BNE (انشعاب اگر غیرمساوی) می‌باشند.

اگر ما اعداد را علامت دار تصور کنیم عدد موجود در A برابر 16− است. زیرا علامت A منفی و 11110000 مکمل 2 عدد 0000 0001 می‌باشد که معادل 16+ دهدهی است. عدد دهدهی B هم 20+ است. تفریق دهدهی دوعدد، (20+)− (16−) خواهد بود. نتیجه دودویی حاصل هم 11011100 (مکمل 2 عدد 00100100) می‌باشد که معادل دهدهی 36− است. چون (20+)>(16−) است داریم A<B و A≠B. این دو رابطه از S=1 (منفی) و V=0 (نبودن سرریز) و Z=0 قابل استنتاج است. دستوراتی که پس از مقایسه می‌توانند انشعابی را سبب شوند عبارتند از BLT (انشعاب اگر کوچکتر از)، BLE (انشعاب اگر کوچکتر یا مساوی) و BNE (انشعاب اگر غیرمساوی). دقت کنید که دستورات BNE و BNZ یکسانند. بطورمشابه BE و BZ نیز یکی هستند. هر یک از آنها بخاطر وضوح بیشتر مطلب سه بار در جدول ذکر شده‌اند.

با توجه به مثال فوق واضح است که نسبت اندازه‌های دوعدد بدون علامت می‌تواند از مقادیر C و Z (مسئله ۲۶-۸ ملاحظه شود) پس از دستور مقایسه معین شود. نسبت اندازه‌های اعداد علامت دار با توجه به مقادیر S و V و Z (مسئله ۲۷-۸) مشخص می‌گردد.

بعضی کامپیوترها بیت C را بیت قرض کردن پس ازتفریق A−B فرض می‌کنند. اگر A≥B باشد هیچ قرضی رخ نمی‌دهد ولی اگر A<B باشد از بیت بالا رتبه‌تر عدد قرض گرفته خواهد شد. شرط قرض کردن این است که پس از انجام تفریق با توجه به مکمل 2 عدد B، مکمل بیت نقلی بدست آید. به این دلیل کامپیوتری که پس از یک تفریق قرضی انجام می‌دهد، پس ازجمع مکمل 2 مفروق منه بیت C را مکمل نموده وآنرا قرض تلقی می‌نماید.

فراخوانی[2] وبازگشت[3] اززیرروال

یک زیرروال مجموعه‌ای مستقل از دستورات است که کار محاسباتی خاصی را انجام‌می‌دهد. در طول اجرای یک برنامه ممکن است چند مرتبه اجرای یک زیرروال تکرار گردد. هر بار که زیرروال احضار گردد یک انشعاب به شروع زیرروال انجام شده و مجموعه دستورات آن اجرا می‌گردد. پس از

1-Compare 2- Call 3- Return

اجرای زیرروال، انشعابی به برنامه اصلی صورت خواهد گرفت.

دستوری که انتقال کنترل برنامه را به زیرروال بعهده دارد با نام‌های مختلفی شناخته می‌شود. معمول‌ترین نام عبارتنداز: فراخوانی زیرروال، پرش به زیرروال، انشعاب به زیرروال، یا انشعاب و ذخیره آدرس می‌باشند. دستور فراخوانی زیرروال عبارتست از یک کد عمل به همراه آدرسی که شروع زیرروال را تعیین می‌نماید. دستور طی دو مرحله اجرا می‌شود: (۱) آدرس دستور بعدی موجود در شمارنده برنامه (آدرس برگشت) در مکان موقتی ذخیره می‌گردد و لذا کنترل محل بازگشت را خواهد دانست، و (۲) کنترل به شروع زیرروال منتقل می‌گردد.آخرین دستور هر زیرروال که بازگشت از زیرروال نام دارد، آدرس بازگشت را از مکان موقت به شمارنده برنامه باز می‌گرداند. نتیجه این عمل انتقال کنترل برنامه به آدرسی است که قبلاً در محل موقت ذخیره شده بود.

کامپیوترهای مختلف، مکان‌های موقت متفاوتی را برای ذخیره کردن آدرس برگشت بکار می‌برند. برخی آنرا در اولین مکان از حافظه مربوط به زیر برنامه ذخیره می‌کنند، و بعضی هم آنرا در مکان ثابتی نگه می‌دارند، دسته‌ای هم در یک ثبات پردازشگر آنرا حفظ می‌کنند و بالاخره عده‌ای هم آنرا در پشته ذخیره می‌نمایند، مزیت استفاده از پشته برای آدرس بازگشت این است که می‌توان زیرروال‌های متعددی را متوالیاً فراخواند که در این حالت آدرس‌های بازگشت می‌توانند بداخل پشته بطور متوالی ارسال گردند. دستور بازگشت از زیرروال سبب می‌شود تا محتوای حافظه فوقانی در پشته فراخوانی و به شمارنده برنامه منتقل گردد. بدین ترتیب، بازگشت همیشه به آخرین زیرروال خواهد بود. احضار یک زیرروال توسط دو ریزعمل زیر صورت می‌گیرد.

کاهش اشاره‌گر پشته	SP ← SP − 1
انتقال محتوای PC را به پشته (پوش)	M[SP] ← PC
انتقال کنترل به زیرروال	PC ← effective address

اگر زیرروال دیگری توسط زیرروال جاری احضار شود، آدرس بازگشت جدید بداخل پشته پوش داده می‌شود والی آخر. دستوری که بازگشت از آخرین زیرروال را انجام می‌دهد توسط ریزعمل‌های زیرپیاده‌سازی می‌شود.

انتقال آدرس از پشته به PC	PC ← M[SP]
افزایش اشاره‌گر پشته	SP ← SP + 1

با استفاده از پشته برای زیرروال، تمام آدرس‌های بازگشت بصورت اتوماتیک توسط سخت افزار در یک واحد ذخیره می‌شوند و برنامه نویس لزومی ندارد مکان آدرس‌های ذخیره شده را بخاطر بسپارد.

یک زیرروال بازگشتی [1] زیرروالی است که خودش را فرا می‌خواند. اگر تنها یک ثبات یا حافظه برای

1- Recursive

ذخیره کردن آدرس برگشت بکارورود، و زیرروال بازگشتی خودش را فراخوانی نماید، آدرس ذخیره شده قبلی را تخریب خواهد کرد. معمولاً این اتفاق مطلوب نیست. برای رفع این مشکل مکان‌های مختلفی برای هر بار مورد استفاده قرار گیرد هر آدرس بازگشتی می‌تواند در پشته ذخیره شود بدون آنکه مقادیر قبلی تخریب گردند. این انتخاب، مسئله زیرروال بازگشتی را نیز حل می‌کند زیرا اولین زیرروالی که ترک می‌شود آخرین زیرروالی است که احضار شده است.

وقفه برنامه

وقفه برنامه به منظور برخورد وحل مشکلات ناشی از مسائلی است که خارج از روال معمول برنامه ایجاد می‌شوند. منظور از وقفه برنامه انتقال کنترل برنامه از برنامه جاری به برنامه دیگری بنام سرویس‌دهی وقفه است که پس از یک تقاضای داخلی یا خارجی صورت می‌گیرد. پس از اجرای برنامه سرویس دهی وقفه، کنترل مجدداً به برنامه اصلی باز می‌گردد.

رویه[1] وقفه در اصل مشابه با یک فراخوانی زیرروال است، بجز در سه موردزیر: (۱) وقفه معمولاً توسط یک سیگنال داخلی یا خارجی رخ می‌دهد و نه با اجرای یک دستورالعمل (بجز برای وقفه نرم افزاری که بعداً توضیح داده شده است)؛ (۲) آدرس برنامه سرویس دهی وقفه توسط سخت افزار معین می‌شود و نه بوسیله میدان آدرس یک دستور؛ (۳) رویه وقفه معمولاً تمام اطلاعات لازم رابرای تعیین وضعیت CPU در آینده ذخیره می‌کند و تنها به ذخیره شمارنده برنامه اکتفا نمی‌نماید. در زیر مفاهیم سه‌گانه فوق را تشریح می‌نمائیم .

پس از هر وقفه در برنامه اصلی واجرای برنامه سرویس وقفه، CPU باید دقیقاً به مکانی که به بهنگام ایجاد وقفه بود بازگردد. تنها در این صورت است که اجرای برنامه متوقف شده قادر خواهد بود بگونه‌ای که هیچ اتفاقی نیفتاده ادامه یابد.وضعیت CPU در انتهای سیکل اجرا (پس از تشخیص تقاضای وقفه) از موارد زیر معین می‌گردد.

۱- محتوای شمارنده برنامه
۲- محتوای تمام ثبات‌های پردازشگر
۳- محتوای بعضی از بیت‌های وضعیت

مجموعه کلیه بیت‌های وضعیت در CPU گاهی اوقات کلمه وضعیت برنامه[2]، PSW خوانده می‌شود. PSW در یک ثبات سخت افزاری جداگانه ذخیره می‌شود و حاوی اطلاعات وضعیتی CPU است. معمولاً این اطلاعات شامل بیت‌های وضعیت از آخرین عمل در ALU، اجازه به وقوع یا عدم وقوع وقفه و اینکه آیا CPU در وضعیت ناظری (سوپروایزری) است و یا در اختیار کاربر[3]، می‌باشد. بسیاری از کامپیوترها دارای یک سیستم عامل مقیمی هستند که تمام برنامه‌های دیگر راکنترل و نظارت می‌کند.

1- Procedure 2- Program Status Word 3- User Mode

وقتی که CPU بخشی از برنامه سیستم عامل را اجرا می‌کند در وضعیت نظارت است. تنها دستورات معینی در این وضعیت ارجحیت داشته و قابل اجرا هستند. CPU معمولاً بهنگام اجرای برنامه کاربر در وضعیت کاربر است. وضعیت کاری CPU در هر لحظه از بیت‌های خاصی از PSW قابل تشخیص است.

دسته‌ای از کامپیوترها بهنگام پاسخ به وقفه شمارنده برنامه را ذخیره می‌کنند. در نتیجه برنامه سرویس به وقفه، قبل از استفاده از بیت‌های وضعیت و ثباتها باید توسط دستوراتی آنها را ذخیره کند. فقط چند کامپیوتر بهنگام پاسخ به وقفه هم شمارنده و هم تمام بیت‌های وضعیت و ثباتها را ذخیره می‌کنند. بسیاری از کامپیوترها نیز فقط شمارنده برنامه و PSW را ذخیره می‌نمایند. در بعضی موارد، برای هریک از دونوع عملکرد نظارت و کاربری مجموعه‌های جداگانه‌ای از ثبات وجود دارد. بااین ترتیب، وقتی که برنامه از وضعیت کاربری به نظارت سوئیچ می‌شود (یابالعکس)، نیازی به ذخیره کردن محتوای ثبات‌ها وجود نداشته و هر یک از مجموعه ثبات‌های خود استفاده می‌نمایند.

گفتیم که روش یا رویه سخت افزاری پردازش وقفه مشابه با فراخوانی یک زیرروال است. در هنگام وقوع وقفه وضعیت CPU به داخل حافظه پشته پوش می‌شود و آدرس شروع روال سرویس دهی به شمارنده برنامه منتقل می‌گردد. آدرس شروع روال سرویس بجای تعیین توسط میدان آدرس یک دستور، بطور سخت افزاری معین می‌گردد. برخی کامپیوترها بهنگام وقفه به آدرس ثابتی مراجعه می‌نمایند. در این صورت برنامه سرویس باید علت وقفه را مشخص و آنرا سرویس دهد. بعضی کامپیوترها هم برای وقفه یک مکان حافظه جداگانه را تخصیص می‌دهند. گاهی اوقات هم وقفه سخت افزاری آدرس خود را برای هدایت CPU در اختیار می‌گذارد. در هر حال، CPU بایستی نوعی رویه سخت‌افزاری را برای انتخاب آدرس انشعاب به منظور سرویس دهی به وقفه داشته باشد.

CPU تا فرارسیدن انتهای اجرای یک دستور به وقفه پاسخ نمی‌دهد. کنترل ، درست قبل از فاز دریافت بعدی، وجود سیگنال وقفه را چک می‌کند. اگر وقفه‌ای تقاضا شده باشد، کنترل وارد سیکل وقفه سخت فزاری می‌شود. در طول این سیکل، محتویات PC و PSW به داخل پشته پوش داده می‌شوند. سپس آدرس انشعاب مربوط به وقفه خاص موردنظر به PC منتقل می‌شود و PSW جدیدی در ثبات وضعیت بار می‌شود. بدین ترتیب، برنامه سرویس‌دهی می‌تواند با شروع از آدرس انشعاب و وضعیتی که PSW جدید مشخص می‌کند اجرا شود.

آخرین دستور در برنامه سرویس وقفه بازگشت از وقفه[1] می‌باشد. وقتی که این دستور اجرا شود، پشته پاپ می‌شود تا PSW قبلی و آدرس برگشت بازیابی گردند. PSW به ثبات وضعیت و آدرس برگشت به شمارنده برنامه انتقال می‌یابد. لذا وضعیت CPU به حالت اولیه بازگشت و اجرا برنامه اصلی ادامه می‌یابد.

انواع وقفه

سه نوع وقفه عمده موجود که موجب توقف اجرای عادی یک برنامه می‌شوند در زیر طبقه‌بندی

1- Return from Interrupt

شده‌اند.

۱- وقفه خارجی
۲- وقفه داخلی
۳- وقفه نرم افزاری

وقفه خارجی از وسایل ورودی - خروجی [1] (I/O)، از وسایل زمانبندی، از مدارهای کنترل کننده منابع تغذیه یا هر منبع خارجی دیگر تولید می‌شوند. مثالهایی از عوامل تولید وقفه خارجی عبارتند از وسایل I/O متقاضی انتقال داده، وسایل I/O اعلام کننده پایان انتقال داده، انقضای زمان یک رخداد، یا از کار افتادن منابع تغذیه است. وقفه‌های اعلان اتمام وقت [2] از قرار گرفتن احتمالی برنامه در یک حلقه بی‌پایان ناشی می‌شوند. در نوعی وقفه حاصل از وجود اشکال در منبع انرژی، روال سرویس دهی می‌باید ظرف چند میلی ثانیه وضعیت کامل CPU را به یک حافظه انتقال دهد.

وقفه‌های داخلی از کاربرد غیرمجاز و یا اشتباه یک دستور و یا داده ناشی می‌شود. این وقفه را تله [3] یا دام هم می‌خوانند. مثال‌هایی از این نوع وقفه‌ها عبارتند از، سرریز [4] ثبات ، تقسیم برصفر، کد عمل غیرمعتبر، سرریزپشته و نادیده گرفتن مقررات حفاظتی. این خطاها معمولاً بهنگام اجرای ناقص دستور رخ می‌دهد. برنامه سرویس وقفه که وقفه داخلی راپردازش می‌کند اقدامات اصلاحی راکه باید صورت گیرد تعیین می‌کند.

اختلاف بین وقفه‌های داخلی و خارجی این است که وقفه داخلی از بعضی وضعیت‌های استثنایی حاصل از خود برنامه ناشی می‌شود و نه از یک پدیده خارجی. وقفه‌های داخلی با برنامه همگام هستند در حالیکه وقفه‌های خارجی حالت غیرهمگام را با برنامه دارند. اگر برنامه مجدداً اجرا شود وقفه داخلی هربار در همان مکان تکرار می‌شود. وقفه‌های خارجی به شرایط خارجی بستگی داشته و مستقل از برنامه در حال اجراست.

وقفه‌های خارجی و داخلی توسط سیگنالهایی که در سخت افزار CPU بوجود می‌آیند اتفاق می‌افتند. وقفه نرم افزاری با اجرای یک دستور ایجاد می‌شود. این وقفه نوع خاصی از دستور فراخوانی است که بیشتر مانند یک وقفه عمل می‌کند و نه مانند فراخوانی زیرروال. این دستور می‌تواند توسط کاربر برای ایجاد یک وقفه نرم افزاری در هر نقطه مطلوب از برنامه بکار رود. متداول‌ترین کاربرد وقفه نرم افزاری به دستورالعمل فراخوانی ناظر مربوط می‌شود. اجرای این دستور CPU را از حالت کاربری به حالت ناظر منتقل می‌نماید. در کامپیوتر اعمال معینی مانند انتقال ورودی و خروجی‌های پیچیده ممکن است به وضعیت ناظری محول شوند. برنامه نوشته توسط کاربر باید در وضعیت کاربری اجرا شود. وقتی که یک انتقال ورودی یا خروجی لازم شود، وضعیت ناظری توسط یک دستور فراخوانی

1- Input / Output 2- Time Out 3- Trap
4- Overflow

ناظر تقاضا می‌گردد. این دستور یک وقفه نرم افزاری را تولید می‌کند و در نتیجه وضعیت فعلی CPU را ذخیره و PSW جدیدی راکه به وضعیت ناظر متعلق است بار می‌کند. برنامه فراخواننده باید اطلاعات را به سیستم عامل جهت انجام کار خاص تحویل نماید.

۸-۸ کامپیوتر کم دستور[1] RISC

یکی از اهداف مهم آرشیتکت یا معماری کامپیوتر، طراحی مجموعه دستورات برای پردازنده است. مجموعه دستورات انتخابی برای یک کامپیوتر خاص تعیین کننده ساختار برنامه‌ها در زبان ماشین آن است. کامپیوترهای اولیه مجموعه دستورات ساده و کوچکی را داشتند که این خود بیشتر بعلت اجبار در کاهش سخت افزار بود. به مرور که سخت افزار دیجیتال با ابداع مدارهای مجتمع ارزان شد، دستورات کامپیوتر از نظر پیچیدگی و تعداد رو به افزایش گذاشت. بسیاری از کامپیوترها در حال حاضر بیش 100 و حتی 200 دستورالعمل را دارا هستند. بعلاوه این کامپیوترها ازانواع داده‌ها و تعداد زیادی روشهای آدرس دهی سود می‌برند. سیرحرکت در پیچیدگی سخت افزار بعلت فاکتورهای مختلفی مانند ارتقای مدل‌های موجود برای افزایش زمینه‌های کاربردی، افزایش دستوراتی که تبدیل زبانهای سطح بالا را به زبان ماشین تسهیل نمایند و تلاش در طراحی ماشین‌هایی که توابع را از فرم نرم افزاری به سخت افزاری انتقال دهند، بود. یک کامپیوتر با تعداد زیادی دستور راکامپیوتر با مجموعه دستورات پیچیده[2] (CISC) یا پردستور می‌نامند.

در اوایل 1980، تعدادی از سازندگان، طراحی کامپیوتری با دستورات کمتر و ساختاری ساده‌تر را پیشنهاد نمودند بطوری که بتوانند سریع‌تر در CPU اجرا شوند و ضمناً به استفاده مکرر از حافظه هم نیازی نداشته باشند. این نوع کامپیوترها با نام کامپیوتر با مجموعه دستورات کاهش یافته (RISC) یا کم دستور طبقه‌بندی می‌شوند. در این بخش مشخصات مهم معماری CISC و RISC را معرفی و سپس مجموعه و قالب دستورات رانشان خواهیم داد.

مشخصه‌های CISC

در طراحی یک مجموعه دستورالعمل نه تنها ساختار زبان ماشین بلکه نیازهای زبان‌های سطح بالا را نیز باید در نظر گرفت. ترجمه برنامه بازبانهای سطح بالا به زبان ماشین ، توسط کامپایلر (یا مترجم) انجام می‌شود. یکی از دلایل روی آوردن سازندگان به مجموعه دستورات CISC ساده کردن کار ترجمه فوق و ارتقای عملکرد کلی کامپیوتر است. وظیفه کامپایلر تولید مجموعه دستورات زبان ماشین برای هر عبارت در زبان سطح بالاست. اگر دستوراتی بزبان ماشین برای تبدیل مستقیم وجود داشته باشند کار ساده‌تر خواهد بود. هدف اصلی معماری CISC تهیه یک دستورالعمل برای هر عبارتی است که در زبان سطح بالا نوشته

1- Redneed Instruction Set Computer

2- Complex Instruction Set Computer

شده باشد. مثالهایی از CISC عبارتند از کامپیوتر VAX متعلق به Digital Equipment Cooperation و کامپیوتر IBM 370 .

مشخصه دیگری از ساختار CISC استفاده از قالب دستورات با طول متغیر است. دستوراتی که به ثبات احتیاج دارند ممکن است دو بایتی باشند ولی دستوراتی که دو آدرس حافظه را لازم دارند می‌توانند برای تمام کد دستور تا پنج بایت را استفاده نمائید. اگر کلمات کامپیوتر ۳۲ بیتی (چهاربایت) باشند، اولین دستور نیمی از کلمه را اشغال می‌کند، در حالیکه دومین دستور به یک کلمه و یک بایت از کلمه بعدی احتیاج دارد. قرار دادن قالبهای متغیر دستورالعمل در کلمه‌های حافظه، با طول ثابت، نیاز به مدار دیکدر خاصی دارند که قادر است تعداد بایت را در کلمه شمرده و دستورات را برطبق تعداد بایت آنها دسته‌بندی نماید.

دستورالعمل‌های یک نمونه پردازشگر CISC امکان استفاده مستقیم از عملوندها را درحافظه فراهم می‌آورند. مثلاً، یک دستور ADD می‌تواند دارای یک عملوند در حافظه با روش آدرس‌دهی شاخص دار و یک عملوند در حافظه از طریق آدرس دهی مستقیم باشد. حافظه دیگری را هم می‌توان در دستور برای ذخیره کردن نتیجه منظور کرد. درنتیجه در طول اجرای دستور سه بار ارجاع به حافظه وجود خواهد داشت. هر چند پردازشگرهای CISC دارای دستوراتی هستند که فقط از ثبات‌ها استفاده می‌نمایند. وجود روش‌های آدرس دهی دیگر در عملیات، کامپایل کردن زبان‌های سطح بالا را ساده می‌نماید. با این وجود، هر چه تعداد دستورات و روش‌های آدرس دهی در کامپیوتر بیشتر شود، مدارهای سخت افزاری بیشتری برای پیاده‌کردن و پشتیابی آنها لازم است و این سبب می‌گردد تا سرعت محاسبات کاهش یابد. بطور خلاصه مهمترین مشخصه‌های یک معماری CISC عبارتند از:

۱- تعداد زیاد دستورالعمل‌ها - معمولاً بین ۱۰۰ تا ۲۵۰ دستور
۲- دستورالعمل‌هایی که کارهای خاصی انجام می‌دهند ولی بندرت بکار می‌روند.
۳- انواع متنوعی از روشهای آدرس دهی موجود است - معمولاً بین ۵ الی ۲۰ روش مختلف
۴- قالب دستورات با طول متغیر
۵- دستوراتی که عملوندها را در حافظه دستکاری می‌کند.

مشخصه‌های RISC

هدف از معماری RISC کوتاه کردن زمان اجرا، با کاهش مجموعه دستورات در کامپیوتر است. مهمترین مشخصه‌های پردازشگر RISC بقرار زیرند.

۱- دستورات نسبتاً کم
۲- روش‌های آدرس دهی نسبتاً کم
۳- دستیابی به حافظه منحصر است به دستورات بارکردن[1] و ذخیره سازی[2]
۴- تمام اعمال در داخل ثبات‌های CPU انجام می‌شوند.

1- LOAD 2- STORE

۵- دستورات باطول ثابت که بسادگی دیکد می‌شوند.
۶- اجرای دستورات در یک سیکل
۷- کنترل سخت افزاری بجای ریزبرنامه نویسی

مجموعه کوچک دستورات یک پردازشگر RISC عمدتاً از عملیات ثبات - ثبات تشکیل شده و برای دستیابی به حافظه نیز دستورات ساده بارکردن و ذخیره سازی نیز در نظر گرفته شده است. بنابراین هر عملوندی ابتدا توسط دستور بارکردن به یک ثبات داخلی منتقل می‌شود. تمام محاسبات روی داده‌های ذخیره شده در ثبات صورت می‌گیرد. نتایج هم بکمک دستور ذخیره سازی به حافظه بازگردانده می‌شود. این امکانات در معماری سبب ساده شدن مجموعه دستورات و موجب تشویق بهینه سازی عملیات دستکاری ثباتها می‌گردد. استفاده از تنها چند روش آدرس دهی بعلت بکارگیری آدرس دهی ثباتی برای تمام دستورالعمل‌هاست. سایر روشهای آدرس دهی مانند بلافصل یا نسبی نیز گاهاً بکار گرفته می‌شوند.

با استفاده از قالب نسبتاً ساده برای دستورات، طول دستور می‌تواند ثابت و برمحدوده کلمات منطبق گردد. یکی از خصوصیات مهم دستورات RISC ، سادگی در دیکدشدن است. بنابراین کنترل می‌تواند به کد عمل و میدان ثبات بطور همزمان دست یابد. با ساده‌کردن دستورات و قالب آن‌ها، مدار کنترل را می‌توان ساده کرد و برای بالابردن سرعت عملیات، یک کنترل سخت افزاری برنوع ریزبرنامه نویسی شده ارجحیت دارد. مثالی از کنترل برای کامپیوتر پایه در فصل ۵ آورده شد. مثالهایی از کنترل ریزبرنامه‌ای نیز در فصل ۷ ارائه گردید.

از مشخصه‌های بارز پردازشگرهای RISC توانایی اجرای هر دستور در یک پالس ساعت است. این عمل با همزمانی انجام فازهای برداشت، دیکد و اجرای دو یا سه دستورالعمل با استفاده از روشی موسوم به پردازش خط لوله[1] ایست. دستورات بارکردن و ذخیره ممکن است دو سیکل ساعت نیاز داشته باشند. زیرا دستیابی به حافظه نیاز به زمان بیشتری نسبت به اعمال ثباتی دارد. خط لوله کارا به همراه چند مشخصه دیگر اغلب به RISC منتسب‌اند، هرچند این ویژگی‌ها در معماری‌های غیر RISC نیز وجود دارند. دیگر مشخصه‌های مربوط به معماری RISC عبارتند از:

۱- تعداد قابل توجهی ثبات در واحد پردازشگر
۲- استفاده از دریچه‌های ثبات هم پوش[2] برای بالا بردن سرعت دستورات فراخوانی و بازگشت از رویه
۳- خط لوله کارا در ثبات‌ها
۴- پشتیانی کامپایلر برای انتقال کارآی برنامه‌های زبان های سطح بالا به برنامه‌های زبان ماشین

1- Pipeline 2- Overlapped Register Windows

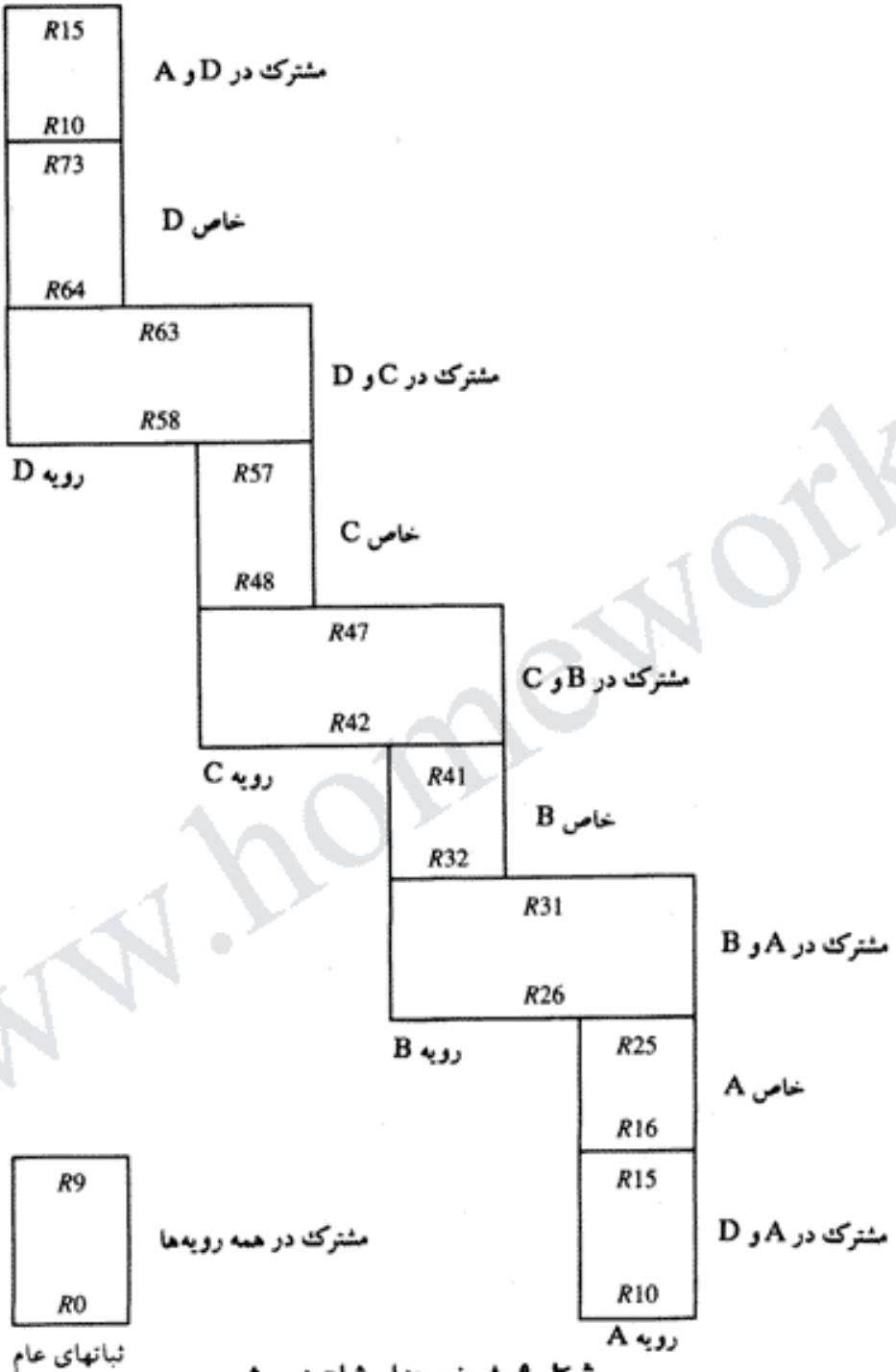
بالابردن تعداد ثبات‌ها برای ذخیره کردن نتایج میانی و کاهش تعداد ارجاعات به عملوندها مفید است. مزیت مکان‌های ذخیره از نوع ثبات در مقابل حافظه این است که عمل انتقال در ثبات‌ها بسیار سریعتر از حافظه‌هاست. بنابراین عمل انتقال ثبات به حافظه رابا نگهداری عملوند پرمصرف‌تر در ثبات می‌توان سرعت بخشید. مطالعات نشان می‌دهد که اصطلاحات در عملکرد معماری تواماً بعلت هر دو عامل ناشی از مجموعه دستورات کاهش یافته و نیز ناشی از تعداد ثبات‌ها بوده است. به این دلیل گاهی اوقات تعداد زیادی ثبات در واحد پردازش RISC در نظر گرفته می‌شوند. در ادامه این بخش، استفاده از دریچه‌های (پنجره) ثباتی هم پوش هنگام انتقال کنترل برنامه پس از یک فراخوانی رویه توضیح داده شده است. خط لوله دستورات در RISC در بخش ۵-۹ پس از تشریح مفهوم خط لوله توضیح داده خواهد شد.

دریچه‌های ثباتی همپوش

فراخوانی رویه‌ها و بازگشت، اغلب در زبانهای برنامه نویسی سطح بالا استفاده می‌شوند. وقتی که یک رویه بزبان ماشین ترجمه شود رشته‌ای از دستورات را تولید می‌کند که توسط آنها مقادیر ثبات‌ها ذخیره می‌شوند، پارامترهای لازم را برای رویه‌ها تحویل می‌نمایند و سپس زیرروالی را برای اجرای رویه فرامی‌خوانند. پس از بازگشت از رویه، برنامه مقادیر قبلی ثباتها را در مکان‌هایشان می‌نشاند، نتایج را به برنامه فرا خواننده تحویل و از زیرروال باز می‌گردد. ذخیره و بازیابی ثباتها و تحویل پارامترها و نتایج، عملیات وقتی گیری را بدنبال دارد. برخی کامپیوترها مجموعه‌هایی از ثبات‌ها را در اختیار دارند و لذا هر رویه مجموعه ثبات خاص خود را استفاده می‌نماید. این امکان سبب می‌شود تا از ذخیره و بازیابی مقادیر در ثبات‌ها خودداری شود. بعضی کامپیوترها هم از پشته حافظه‌ای‌برای ذخیره‌کردن پارامترهای لازم برای رویه‌ها استفاده می‌کنند ولی این خود نیاز به دستیابی حافظه در هر فراخوانی پشته دارد.

یکی ازویژگی‌های برخی از پردازشگرهای RISC استفاده از دریچه‌های ثباتی هم پوش برای تحویل پارامترها و اجتناب از ذخیره و بازیابی مقادیر ثبات‌هاست. فراخوانی هر رویه منجر به تخصیص یک دریچه جدید متشکل از مجموع ثبات‌ها از فایل ثبات‌ها برای استفاده رویه جدید می‌شود. هر فراخوانی رویه، دریچه ثبات جدید را با افزایش یک اشاره‌گر فعال می‌سازد. و بالعکس یک عبارت بازگشت اشاره‌گر راکاهش و سبب فعال شدن دریچه قبلی می‌گردد. دریچه‌ها برای رویه‌های مجاور دارای ثبات‌های هم پوشند تا برای تبادل پارامترها و نتایج مورد استفاده مشترک قرار گیرند.

مفهوم دریچه‌های ثباتی هم پوش در شکل ۹-۸ تشریح شده است. سیستم دارای 74 ثبات است. ثبات‌های R_0 تا R_9 ، از نوع عام هستند و پارامترهای مشترک بین همه رویه را نگه می‌دارند. بقیه 64 ثبات به چهار دریچه برای جای دهی به رویه‌های A، B و C و D تخصیص یافته‌اند. هر دریچه ثبات متشکل از ده ثبات محلی (درونی) و دو مجموعه شش ثباتی مشترک با دریچه‌های مجاور است. ثبات‌های محلی برای متغیرهای محلی استفاده می‌شوند. ثبات‌های مشترک برای تعویض پارامترها و



شکل ۹-۸ پنجره‌های ثبات همپوش

نتایج با رویه مجاور مورد استفاده قرار می‌گیرند. ثبات‌های مشترک هم پوش به پارامترها اجازه می‌دهند تا بدون حرکت واقعی داده‌ها جابجا شوند. در هر لحظه تنها یک دریچه ثبات فعال می‌شود و اشاره‌گر نیز دریچه فعال را نشان خواهد داد. هر فراخوانی رویه یک دریچه ثبات جدید را با افزایش اشاره‌گر فعال

می‌نماید. ثبات‌های فوقانی فراخواننده بر روی ثبات‌های تحتانی فراخوانده شده قرار دارند (هم پوشند) و لذا پارامترها بطور اتوماتیک از رویه فراخواننده به رویه فراخوانده منتقل می‌شوند.

بعنوان مثال، فرض کنید که رویه A، رویه B را فرا بخواند . ثبات‌های R_{26} تا R_{31} بین هر دو رویه مشترکند، و بنابراین رویه A پارامترها را برای رویه B در این ثبات‌ها ذخیره می‌کند. رویه B از ثبات‌های محلی R_{32} تا R_{42} برای ذخیره متغیرهای محلی استفاده می‌کند. اگر رویه B، رویه C را فرابخواند، پارامترها را از طریق R_{42} تا R_{47} به آن‌تحویل می‌نماید. وقتی که در انتهای محاسبات رویه B آماده بازگشت به رویه A باشد، برنامه نتایج محاسبات را درثبات‌های R_{26} تا R_{31} ذخیره کرده و به دریچه ثبات‌های رویه A بازمی‌گردد. توجه کنید که ثبات‌های R_{10} تا R_{15} بین رویه A و رویه D مشترک است زیرا چهار دریچه ساختار حلقوی دارند و لذا A با D مجاور است.

همانطور که قبلاً اشاره شد، ده ثبات عام R_0 تا R_9 مورد استفاده همه رویه‌ها هستند. هر رویه در صورت فعال بودن 32 ثبات در دسترس دارد. این تعداد شامل ده ثبات عام، ده محلی، و شش ثبات همپوش پائین رتبه و شش همپوش بالا رتبه می‌باشند. انواع دیگری از ساختارهای دریچه‌های ثباتی با ابعاد ثابت نیز امکان پذیر است، و هر یک ممکن است از نظر سایز دریچه‌ها و سایز کل فایل ثبات‌ها با دیگری اختلاف داشته باشند. بطور کلی سازمان دریچه‌های ثبات روابط زیر را دارا هستند.

۱- تعداد ثبات‌های عام G
۲- تعداد ثبات‌های محلی در هر دریچه L
۳- تعداد ثبات‌های مشترک بین دو دریچه C
۴- تعداد دریچه‌ها W

تعداد ثبات‌های قابل دسترسی توسط هر دریچه از رابطه زیر محاسبه می‌شود

$$\text{سایز دریچه} = L + 2C + G$$

تعداد ثبات‌های مورد نیاز در پردازشگر برابرست با

$$\text{فایل ثبات} = (L + C)\,W + G$$

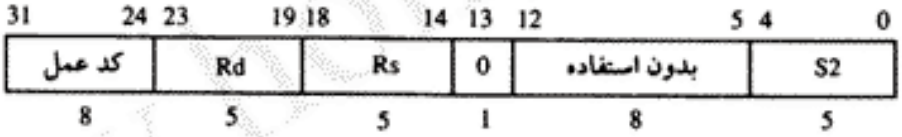
در مثال شکل ۹-۸ داریم $G = 10$، $L = 10$، $C = 6$ و $W = 4$. سایزدریچه $32=10+12+10$ و فایل ثبات متشکل از 74 ثبات خواهد بود زیرا $74=10+4\times(10+6)$.

Berkeley RISC I

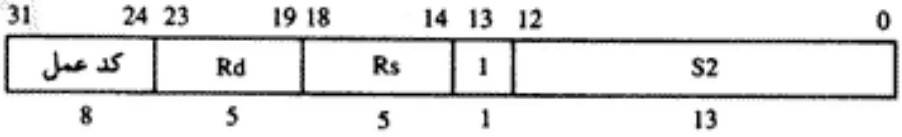
یکی از پروژه‌هایی که مزایای معماری RISC را نشان داد در دانشگاه برکلی در ایالت کالیفرنیا انجام شد. Berkeley RISCI یک CPU مدار مجتمع 32 بیتی است. این CPU دارای 32 بیت آدرس بوده و

داده‌های 8، 16 و 32 بیتی را پشتیبانی می‌کند. قالب دستورات نیز 32 بیتی است و کلاً 31 دستورالعمل دارد. سه روش آدرس دهی اساسی را داراست: آدرس دهی ثباتی، عملوندهای بلافصل و آدرس دهی نسبت به PC برای دستورات انشعاب. فایل ثبات آن دارای 138 ثبات دسته‌بندی شده به ده ثبات عام و هشت دریچه 32 ثباتی در هر کدام است. تعداد 32 ثبات در هر دریچه، سازمانی مشابه با شکل ۹-۸ دارد. چون در هر زمان معین فقط 32 ثبات در هر دریچه قابل دستیابی هستند، قالب دستور می‌تواند دارای پنج بیت برای میدان ثبات باشد.

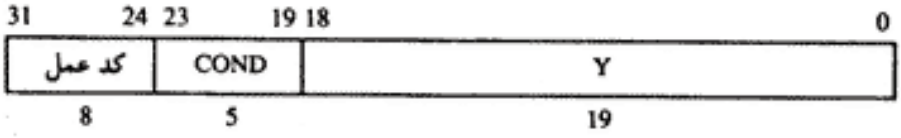
شکل ۱۰-۸ قالب دستورات بکار رفته برای دستورات ثبات - ثبات و دستیابی به حافظه را نشان می‌دهد. هفت بیت موجود کد عمل، یک عمل خاص را مشخص می‌کنند و بیت هشتم معین می‌کند که آیا بیت‌های وضعیت پس از عمل ALU باید بهنگام [1] شوند یا خیر. برای دستورات ثبات -ثبات میدان پنج بیتی Rd یکی از 32 ثبات را بعنوان مقصد [2] برای نتیجه عمل انتخاب می‌کند. عمل توسط داده‌های موجود در میدان‌های Rs و S2 معین می‌گردد. Rs یکی از ثبات های مبدأ [3] (منبع) است. اگر بیت 13 از دستورالعمل 0 باشد پنج بیت پائین رتبه S2 ثبات مبدأ دیگر را معین می‌کند. اگر بیت 13 از دستورالعمل 1 باشد، S2 مشخص کننده یک مقدار ثابت 13 بیتی با تعمیم علامت [4] است. بنابراین دستورالعمل دارای سه قالب آدرس است، ولی دومین مبدأ ممکن است یک ثبات و یا یک عملوند بلافصل باشد.



(الف) شیوه ثباتی: (S2 یک ثبات را مشخص می‌کند)



(ب) شیوه ثباتی بلافصل: (S2 یک عملوند را معین می‌نماید



(ج) شیوه نسبت به PC

شکل ۱۰-۸ قالب‌های دستورالعمل‌ها در Berkeley RISC I

1- Update
2- Destination
3- Source
4- Sign Extended

دستورات دستیابی به حافظه از Rs برای تعیین آدرس 32 بیتی در ثبات استفاده می‌کنند و S2 یک فاصله از مبدأ است. محتوای ثبات R0 تماماً 0 است بنابراین در هر حال بعنوان کمیت صفر می‌تواند مورد استفاده قرارگیرد. سومین قالب دستور سه میدان آخر را با هم ترکیب کرده ویک آدرس نسبی ۱۹ بیتی Y را تشکیل داده است و عمدتاً با دستورات فراخوانی و پرش بکار می‌رود. میدان COND که بجای میدان Rd در دستورالعمل‌ها آمده است برای تعیین یکی از 16 شرط انشعاب استفاده می‌شود.

31 دستور RISC I در جدول (۱۳-۸) لیست و به سه گروه تقسیم شده‌اند. دستورات دستکاری داده اعمال حسابی، منطقی و شیفت را انجام می‌دهند. سمبل زیر ستون کد عمل و عملوند بهنگام نوشتن برنامه‌هایی بزبان اسمبلی بکار می‌روند. ستون‌های انتقال ثبات و توضیحات، عمل دستورات را بصورت علائم ونیز تشریحی بیان می‌دارند. دقت کنید که تمام دستورات دارای سه عملوند هستند. مبدأ S2 می‌تواند یک ثبات و یا یک عملوند بلافصل باشد که با علامت # مشخص شده است. مثلاً دستور ADD را در نظر بگیرید و ملاحظه کنید که چگونه می‌توان انواع اعمال را توسط آن انجام داد.

ADD R22, R21, R23	R23 ← R22 + R21	
ADD R22, #150, R23	R23 ← R22 + 150	
ADD R0, R21, R22	R22 ← R21	(انتقال)
ADD R0, # 150, R22	R22 ← 150	باردهی بلافصل
ADD R22, # 1, R22	R22 ← R22+1	(افزایش)

با استفاده از ثبات R0، که همواره محتوای آن 0 است، می‌توان محتوای یک ثبات یا یک مقدار ثابت را به ثبات دیگری منتقل کرد. عمل افزایش هم باجمع ثابت 1 با یک ثبات امکان پذیر است.

دستورات انتقال داده متشکل از شش دستور بارکردن، سه دستور ذخیره کردن و دو دستور انتقال کلمه PSW می‌باشد. محتویات ثبات PSW وضعیت CPU را نشان می‌دهد که عبارتست از: شمارنده برنامه، بیت‌های وضعیت از ALU، اشاره‌گرهای بکار رفته با دریچه‌های ثباتی و سایر اطلاعاتی که وضعیت CPU را مشخص می‌نماید.

دستورات بارکردن و ذخیره سازی، داده‌ها را بین ثبات و حافظه حرکت می‌دهند. دستورات بارکردن داده‌های علامت دار یا بدون علامت هشت بیتی (بایت) و یا 16 بیتی (کلمه کوتاه) را داراهستند. دستورات کلمات بلند روی داده‌های 32 بیتی عمل می‌نمایند. هر چند بنظر می‌رسد که روش آدرس دهی ثباتی بعلاوه جابجایی در دستورات انتقال داده وجود دارد، روشهای آدرس دهی غیرمستقیم ثبات و مستقیم نیز امکان پذیر است. مثالهای زیر دستورات بارکردن با کلمات بلند را با روشهای مختلف نشان می‌دهد.

LDL (R22) # 150, R5	R5 ← M[R22] +150
LDL (R22) # 0, R5	R5 ← M[R22]
LDL (R0) # 500, R5	R5 ← M[500]

جدول ۱۲-۸ مجموعه دستورالعمل‌های Berkeley RISC I

کدعمل	عملوندها	انتقال ثبات	شرح
دستورالعمل‌های دستکاری داده‌ها			
ADD	Rs,S2,Rd	$Rd \leftarrow Rs + S2$	جمع صحیح
ADDC	Rs,S2,Rd	$Rd \leftarrow Rs + S2 +$ carry	جمع با نقلی
SUB	Rs,S2,Rd	$Rd \leftarrow Rs - S2$	تفریق صحیح
SUBC	Rs,S2,Rd	$Rd \leftarrow Rs - S2 -$ carry	تفریق با نقلی
SUBR	Rs,S2,Rd	$Rd \leftarrow S2 - Rs$	تفریق معکوس
SUBCR	Rs,S2,Rd	$Rd \leftarrow S2 - Rs -$ carry	تفریق با نقلی
AND	Rs,S2,Rd	$Rd \leftarrow Rs \wedge S2$	AND
OR	Rs,S2,Rd	$Rd \leftarrow Rs \vee S2$	OR
XOR	Rs,S2,Rd	$Rd \leftarrow Rs \oplus S2$	OR انحصاری
SLL	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by S	شیفت به چپ
SRL	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by S	شیفت منطقی به راست
SRA	Rs,S2,Rd	$Rd \leftarrow Rs$ shifted by S	شیفت حسابی به راست
دستورالعمل‌های انتقال داده			
LDL	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	بارکردن عملوند بلند
LDSU	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	بارکردن عملوند کوتاه بی‌علامت
LDSS	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	بارکردن عملوند کوتاه علامت‌دار
LDBU	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	بارکردن بایت بی‌علامت
LDBS	(Rs)S2,Rd	$Rd \leftarrow M[Rs + S2]$	بارکردن بایت علامت‌دار
LDHI	Rd,Y	$Rd \leftarrow Y$	بارکردن کمیت بلافصل در بایت بالا
STL	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	ذخیره عملوند بلند
STS	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	ذخیره عملوند کوتاه
STB	Rd,(Rs)S2	$M[Rs + S2] \leftarrow Rd$	ذخیره بایت
GETPSW	Rd	$Rd \leftarrow PSW$	بارکردن کلمه وضعیت
PUTPSW	Rd	$PSW \leftarrow Rd$	مقداردهی به کلمه وضعیت
دستورالعمل‌های برنامه			
JMP	COND, S2(Rs)	$PC \leftarrow Rs + S2$	پرش شرطی
JMPR	COND,Y	$PC \leftarrow PC + Y$	پرش نسبی
CALL	Rd,S2(Rs)	$Rd \leftarrow PC$ $PC \leftarrow Rs + S2$ $CWP \leftarrow CWP - 1$	فراخوانی زیرروال و تغییر پنجره
CALLR	Rd,Y	$Rd \leftarrow PC$ $PC \leftarrow PC + Y$ $CWP \leftarrow CWP - 1$	فراخوانی نسبی و تغییر پنجره
RET	Rd,S2	$PC \leftarrow Rd + S2$ $CWP - CWP + 1$	بازگشت و تغییر پنجره
CALLINT	Rd	$Rd \leftarrow PC$ $CWP \leftarrow CWP - 1$	غیرفعال کردن وقفه‌ها
RETINT	Rd,S2	$PC \leftarrow Rd + S2$ $CWP \leftarrow CWP + 1$	فعال کردن وقفه‌ها
GTLPC	Rd	$Rd \leftarrow PC$	دریافت آخرین وضعیت PC

آدرس موثر در اولین دستور با توجه به محتویات R22 بعلاوه یک مقدار جابجایی 150 محاسبه شده است. دستور دوم دارای فاصله از مبدأ 0 است و در نتیجه دستور تبدیل به یک دستور ثباتی غیرمستقیم می‌گردد. دستور سوم از صفرهای R0 استفاده می‌نماید و بدین‌ترتیب دستور از انواع آدرس مستقیم خواهد بود.

دستورات کنترل برنامه با شمارنده برنامه PC همکاری می‌نمایند تا رشته عملیات برنامه را کنترل کنند. دو دستور فراخوانی (CALL) و دو دستور پرش (JUMP) وجود دارد. یکی از آنها از روش آدرس دهی شاخص بعلاوه فاصله از مبداء استفاده می‌کند، و دومی با استفاده از 19 بیت مقدار Y بعنوان آدرس نسبی، روش نسبی نسبت به PC را بکار می‌گیرد. دستورات فراخوانی و بازگشت از یک ثبات 3 بیتی اشاره‌گر دریچه جاری (CWP)[1] استفاده کرده که به دریچه ثبات فعال جاری اشاره می‌کند. هرگاه برنامه یک رویه جدید را فراخوانی کند، CWP یک واحد کاهش یافته و به دریچه ثبات مرتبه پائین‌تر اشاره می‌نماید. پس از بازگشت، CWP یک واحد افزایش یافته و به دریچه ثبات قبلی بازمی‌گردد.

مسائل

۸-۱ یک CPU با سازمان گذرگاه مشابه با شکل ۸-۲ دارای 16 ثبات 32 بیتی، یک ALU و یک دیکدر مقصد یاب است.

الف - چند مولتی پلکسر در گذرگاه A وجود دارد و سایز هر مولتی پلکسر چقدر است.

ب - چند انتخاب کننده ورودی برای MUXA و MUXB لازم است.

پ - در دیکدر چند ورودی و چند خروجی وجود دارد.

ت - چند ورودی و خروجی داده، منجمله ورودی و خروجی رقم نقلی، در ALU وجود دارد.

ث - با فرض وجود 35 عمل در ALU، کلمه کنترل را برای سیستم فرموله کنید.

۸-۲ سیستم شکل ۸-۲ دارای تأخیرهای زمانی زیر است: 30 ns برای انتشار در MUX، 80ns برای عمل جمع در ALU، 20ns تأخیر در دیکدر مقصدیاب و 10ns برای ورود داده به ثبات مقصد. حداقل سیکل زمانی برای پالس ساعت چقدر است؟

۸-۳ کلمه کنترلی را که باید به پردازشگر شکل ۸-۲ اعمال شود تا اعمال زیر را انجام دهد مشخص کنید.

الف - R1 ← R2 + R3

ب - R4 ← R4

ج - R5 ← R5 − 1

د - R6 ← ShlR1

هـ - R7 ← input

1- Current Window Pointer

۸-۴ عملیات جزیی قابل اجرا در پردازشگر شکل ۲-۸ را وقتی که کلمات کنترل 14 بیتی زیر اعمال می‌شود معین کنید.

الف) 00101001100101

ب) 00000000000000

ج) 01001001001100

د) 00000100000010

هـ) 11110001110000

۸-۵ اجازه بدهید SP = 000000 در پشته شکل ۳-۸ باشد. چند مقدار در پشته وجود دارد اگر:

الف) FULL = 1 , EMTY =0

ب) FULL = 0 , EMTY =1

۸-۶ پشته طوری سازمان یافته است که همیشه SP به مکانی خالی در پشته اشاره می‌نماید. این بدان معنی است که مقدار اولیه SP در شکل ۴-۸ می‌تواند 4000 بوده و اولین کمیت در مکان 4000 از پشته ذخیره شود. برای اعمال پوش و پاپ عملیات جزیی را لیست نمائید.

۸-۷ عبارت محاسباتی را از میانوندی به RPN تبدیل کنید.

الف) A * B + C * D + E *F

ب) A * B + A * (B * D + C *E)

ج) A + B * [C * D + E * (F+G)]

د) $\frac{A * [B + C * (D + E)}{F * (G+H)}$

۸-۸ عبارات محاسباتی زیر را از RPN به میانوندی تبدیل کنید.

الف) A B C D E + * − /

ب) A B C D E * / − +

ج) A B C * / D − E F / +

د) A B C D E F G + * + * + *

۸-۹ عبارت محاسباتی عددی زیر را به RPN تبدیل و عملیات پشته را برای بدست آوردن نتیجه نشان دهید.

[3+4 [10(2+6)+8]

۸-۱۰ سازمان یک حافظه بشکل FIF0 است. نشان دهید چگونه یک حافظه FIF0 با سه شمارنده کار می‌کند. یک شمارنده نوشتن[1]، WC، آدرس نوشتن در حافظه را نگه می‌دارد. یک شمارنده

1- Write Counter

خواندن[1] RC آدرس خواندن در حافظه را حفظ می‌نماید. یک شمارنده فضای موجود در حافظه[2]، ASC، تعداد کلمات ذخیره شده در FIF0 را معین می‌نماید. ASC برای هر کلمه ذخیره شده یک واحد اضافه می‌شود و برعکس برای کلمات دریافت شده یک واحد کم می‌گردد.

۸-۱۱ برنامه‌ای بنویسید که عبارت ریاضی زیر را ارزیابی کند.

$$X = \frac{A - B + C * (D * E - F)}{G + H * k}$$

الف) با استفاده ازکامپیوتر با ثبات‌های عمومی و دستورات سه آدرسه

ب) با استفاده از کامپیوتر با ثبات‌های عمومی و دستورات دو آدرسه

ج) با استفاده از کامپیوتر نوع انباره‌ای با دستورات یک آدرسه

د) با استفاده از کامپیوتر با سازمان پشته و دستورات صفر آدرسه

۸-۱۳ واحد حافظه یک کامپیوتر 256k کلمه 32 بیتی دارد. کامپیوتر دارای قالب دستورات چهار میدانی است: میدان کد عملیات، میدان روش آدرس دهی برای هفت روش موجود، میدان آدرس ثبات برای انتخاب یکی از 60 ثبات پردازشگر ، و آدرس حافظه. قالب دستور و تعداد بیت‌ها در هر میدان را اگر دستور در یک کلمه حافظه باشد معین کنید.

۸-۱۴ یک دستور دو کلمه‌ای در حافظه و در آدرسی که با W مشخص شده ذخیره شده است. میدان آدرس دستور (ذخیره شده در W+1) با Y معین شده است. عملوند بکار رفته در هنگام اجرای دستور در آدرس Z می‌باشد. یک ثبات شاخص دارای مقدار X است. نشان دهید که چگونه Z از سایر آدرس‌ها محاسبه می‌شود اگر که روش آدرس دهی دستور برابر زیر باشد.

الف) مستقیم ب) غیرمستقیم

ج) نسبی د) شاخص دار

۸-۱۷ یک دستور انشعاب از نوع نسبی در آدرس دهدهی 750 از یک حافظه ذخیره شده است. انشعاب به آدرس 500 صورت گرفته است.

الف) مقدار میدان آدرس نسبی دستور (به اعشاری) چقدر است.

ب) مقدار میدان آدرس نسبی را به دودویی در 12 بیت نشان دهید (چرا باید عدد بفرم مکمل 2 باشد).

ج) مقدار دودویی در PC را پس از فاز برداشت (واکشی) بدست آورید و مقدار 500 را محاسبه کنید. سپس نشان دهید که مقدار دودویی در PC بعلاوه آدرس نسبی محاسبه شده در قسمت (ب) برابر عدد دودویی 500 است.

۸-۱۶ واحد کنترل وقتی که یک دستور با آدرس غیرمستقیم را برداشت و اجرا می‌کند چند بار به

1- Read Counter 2- Available

حافظه مراجعه می‌نماید اگر: (الف) دستور از نوع محاسباتی بوده و یک عملوند را از حافظه لازم داشته باشد. (ب) دستور از نوع انشعاب باشد.

۸-۱۷ میدان آدرس یک دستور با آدرس شاخص دار چه باشد تا مشابه یک دستور غیرمستقیم ثباتی گردد.

۸-۱۸ یک دستور در مکان 300 با میدانی در مکان 301 ذخیره شده است. میدان آدرس 400 است. یک ثبات پردازشگر R1 نیز حاوی عدد 200 است. آدرس موثر را اگر روش آدرس دهی دستور (الف) مستقیم؛ (ب) بلافصل (ج) نسبی؛ (د) غیرمستقیم ثباتی؛ (هـ) شاخص دار با R1 بعنوان ثبات شاخص باشد بدست آورید.

۸-۱۹ با فرض داشتن یک کامپیوتر 8 بیتی، یک جمع با دقت مضاعف برای اعداد بدون علامت 32 بیتی زیر با استفاده از دستور جمع با رقم نقلی را انجام دهید. هر بایت بصورت عدد دو رقمی مبنای 16 می‌باشد.

(6E C3 56 78) + (13 55 6B 8F)

۸-۲۰ برای دو رشته اعداد دودویی 10011100 و 10101010 اعمال AND، OR و XOR را انجام دهید.

۸-۲۱ عدد 16 بیتی 1001101011001101 مفروض است. چه عملی باید انجام داد تا:

الف) هشت بیت اول 0 شود

ب) هشت بیت آخر 1 شود

ج) هشت بیت وسط مکمل شود

۸-۲۲ یک ثبات 8 بیتی دارای مقدار 01111011 است و بیت نقلی نیز 1 است. هشت عمل شیفت موجود در جدول ۹-۸ را روی آنها پیاده کنید. هر بار مقدار اولیه، عدد فوق باشد.

۸-۲۳ اعداد علامت دار زیر را به دودویی بصورت 8 بیت نمایش دهید، 83-، 83+، 63-، 63+

الف- جمع (63+) + (83-) را به دودویی انجام و نتیجه را تفسیر کنید.

ب- تفریق (83+) - (68-) را به دودویی انجام و نشان دهید که سرریز رخ می‌دهد.

ج- عدد دودویی 68- را یکبار به راست شیفت دهید و مقدار شیفت داده شده را به دهدهی بدست آورید.

د - عدد دودویی 83- را یکبار به چپ شیفت دهید و ببینید آیا سرریز وجود دارد یا نه؟

۸-۲۴ نشان دهید مداری که خروجی 0 را در شکل ۸-۸ چک می‌کند یک گیت NOR است.

۸-۲۵ یک کامپیوتر 8 بیت دارای یک ثبات R است. مقادیر بیت‌های وضعیت C , S و Z و V (شکل ۸-۸) را پس از اجرای هر یک از دستورات زیر معین کنید. مقدار اولیه R در هر مورد 72 در مبنای شانزده است. اعداد زیر نیز در مبنای شانزده هستند.

الف) عملوند بلافصل C6 را با R جمع کنید.

ب) عملوند بلافصل 1E را با R جمع نمائید.

ج) عملوند بلافصل 9A را از R کم کنید.
د) R را با R، XOR نمائید.

۸-۲۶ دو عدد بدون علامت توسط رابطه A − B باهم مقایسه شده‌اند. بیت نقلی بعنوان بیت قرض پس از مقایسه در بیشتر کامپیوترهای تجاری در نظر گرفته می‌شود، بطوری که اگر A<B باشد C=1 است. نشان دهید که نسبت اندازه A نسبت به B از وضعیت C و Z طبق جدول مربوط به این مسئله و (جدول ۱۱-۸) قابل استنتاج است.

جدول مسئله ۸-۲۶

رابطه	حالت بیت‌های وضعیت
$A > B$	$C = 0$ and $Z = 0$
$A \geq B$	$C = 0$
$A < B$	$C = 1$
$A \leq B$	$C = 1$ or $Z = 1$
$A = B$	$Z = 1$
$A \neq B$	$Z = 0$

۸-۲۷ دو عدد علامت دار بشکل مکمل 2 با هم طبق رابطه A−B مقایسه شده‌اند. با توجه به نتیجه مقایسه بیت‌های وضعیت S و Z و V، 0 یا 1 خواهند شد. (دقت کنید که اگر سرریز رخ دهد تغییر علامتی وجود خواهد داشت). نشان دهید که اندازه نسبی A نسبت به B با توجه به بیت‌های وضعیت درجدول مربوط به این مسئله (وجدول ۱۱-۸) قابل استنتاج است.

جدول مسئله ۸-۲۷

رابطه	حالت بیت‌های وضعیت
$A > B$	$(S \oplus V) = 0$ and $Z = 0$
$A \geq B$	$(S \oplus V) = 0$
$A < B$	$(S \oplus V) = 1$
$A \leq B$	$(S \oplus V) = 1$ or $Z = 1$
$A = B$	$Z = 1$
$A \neq B$	$Z = 0$

۸-۲۸ قرار است یک سیستم دیجیتال با چهار ورودی C و S و Z و V با 10 خروجی طراحی شود، که هر یک مربوط به یک شرط انشعاب در مسئله ۸-۲۶ و ۸-۲۷ است. (حالات مساوی و نامساوی در هر دو جدول یکی است). دیاگرام منطقی مدار با استفاده از دو گیت OR، یک XOR و پنج معکوس کننده را رسم کنید.