اسمبلر و شبیه‌ساز سیستم IBM370

توسط

علیرضا فتحی

رساله ارائه شده به عنوان بخشی از ملزومات برای دریافت درجه

کارشناسی نرم‌افزار کامپیوتر

زیر نظر

دکتر حمید سربازی آزاد

دی‌ماه ۱۳۸۳

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

تهران

قدردانی

در این‌جا لازم است از آقای سربازی‌آزاد که که در تمام مراحل انجام این پروژه با مساعدت‌ها و راهنمایی‌های بی‌دریغ خود مرا یاری کردند، تشکر کنم. همچنین از آقای میرعمادی که جرقه‌ی اولیه‌ی نوشتن یک اسمبلر و شبیه‌ساز را ایشان در درس ساختار و زبان کامپیوتر در ذهن من زدند تشکرمی‌کنم.

این پایان‌نامه را به تمام دوستان خوبم تقدیم می‌کنم.

فهرست مندرجات

۱ مقدمه ۸

۱—۱ تاریخچه ۹

۱—۲ این پروژه و ساختار پایان‌نامه ۹

۲ معرفی ماشین IBM360/370 ۱۱

۲—۱ خصوصیات معماری ۱۱

۲—۲ اجرای دستورها ۱۳
۲—۲—۱ عملوندها ۱۳
۲—۲—۲ قالب دستورها ۱۴
۲—۲—۳ کلمه وضعیت برنامه (PSW) ۱۶

۲—۳ قالب زبان اسمبلی ۱۷
۲—۳—۱ فیلدهای زبان اسمبلی ۱۸
۲—۳—۲ خطاهای قالب زبان اسمبلی IBM370 . . . . . ۱۸

۳ اسمبلر ۲۰

۳—۱ ساختار کلی اسمبلر ۲۰

۳—۲ تشریح نحوه نوشتن برنامه ۲۱
۳—۲—۱ پیاده‌سازی درشت‌دستورالعمل‌ها ۲۱
۳—۲—۲ جایگزین‌کردن نمادها ۲۱
۳—۲—۳ جایگزین کردن عبارات ریاضی با مقدارشان ۲۲

۳—۳ چگونگی ورودی و خروجی ۲۲

۳—۴ نمونه‌ی خطاها ۲۴
۳—۴—۱ مرحله باز کردن فایل ورودی و خواندن آن ۲۴
۳—۴—۲ مرحله تشخیص نوع هر خط از برنامه ۲۵
۳—۴—۳ مرحله محاسبه آدرس هر خط ۲۵
۳—۴—۴ مرحله جایگزین‌کردن نمادها در عملوندها ۲۶
۳—۴—۵ مرحله محاسبه‌کردن مقدار عملوندها و تولید کد ماشین . . . ۲۶

۴ شبیه‌ساز ۲۸

۴—۱ ساختار کلی شبیه‌ساز ماشین ۳۷۰ . . . . . ۲۸

۴—۲ چگونگی ورودی و خروجی ۲۹

۴—۳ آشنایی با برنامه شبیه‌ساز ۲۹

۵ محیط کار با برنامه ۳۳

۵—۱ اسمبلر ۳۳

۵—۲ گزینه‌های دیگر منوی tools ۳۶

۶ خلاصه و نتیجه‌گیری ۳۷

A پیوست ۱ (دستورات ماشین IBM370) ۳۸

A—۱ دستورات عمومی ۳۸

A—۲ دستورات دهدهی ۵۵

A—۳ دستورات ممیزشناور ۵۸

B پیوست ۲ (برنامه کارت پانچ) ۶۳

لیست اشکال

۴—۱ محیط برنامه‌ی شبیه‌ساز IBM370 . . . . . ۳۰

۴—۲ دیالوگ نمایش محتوای حافظه در برنامه‌ی شبیه‌ساز IBM370 . . . . . ۳۲

۵—۱ محیط برنامه IBM370 . . . . . ۳۴

۵—۲ منوی tools در محیط برنامه‌ی IBM370 . . . . . ۳۵

۵—۳ دیالوگ اسمبلر IBM370 . . . . . ۳۶

B—۱ محیط برنامه کارت‌پانچ ۶۴

چکیده

یکی از دروس معمول در رشته‌ی مهندسی کامپیوتر چه گرایش نرم‌افزار و چه گرایش سخت‌افزار درس زبان‌ماشین می‌باشد. این درس در اکثر دانشگاه‌ها در قالب یک درس ۳ واحدی ارائه می‌شود که در نیمه‌ی اول درس به مقدماتی بر معماری‌کامپیوتر پرداخته‌می‌شود و در نیمه‌ی دوم درس زبان اسمبلی یک ماشین خاص آموزش‌داده‌می‌شود.

این پروژه به منظور آموزش زبان اسمبلی ماشین‌های IBM سری ۳۶۰ و ۳۷۰ انجام‌شده‌است. حاصل این پروژه یک ویرایشگر، اسمبلر و شبیه‌ساز برای ماشین IBM370 است که در این نوشته به شرح آن می‌پردازیم. نرم‌افزار نوشته‌شده در یک CD ضمیمه‌ی پایان‌نامه شده‌است. به منظور پشتیبانی از سیستم نوشته‌شده یک homepage به آدرس http://ibm370.ce.sharif.edu طراحی‌شده‌است که می‌توانید آخرین نسخه‌ی نرم‌افزار را از طریق این آدرس دریافت‌کنید.

واژه‌های کلیدی: زبان‌ماشین، مجموعه‌ی دستورها، اسمبلر دوگذر ، شبیه‌ساز، کارت‌پانچ

فصل ۱

مقدمه

خانواده سیستم IBM360/370 به‌عنوان شاهکار معماری در زمان خود و به‌عنوان یک مدل آموزشی بسیار مناسب در حال حاضر می‌باشد. سیستم IBM370 حاصل تجربیات کسب شده از سیستم های ۳۶۰ می باشد. سیستم ۳۷۰ راسازگار با سیستم ۳۶۰ طراحی کردند تا انتقال از ماشین‌های ۳۶۰ به ۳۷۰ ممکن باشد. کامپیوترهای ۳۷۰ علاوه بر قابلیت اطمینان و دسترسی پذیری بیشتر نسبت به ماشین‌های ۳۶۰ دارای یک سری خصوصیات جدید نسبت به سیستم های ۳۶۰ بودند از جمله :

ترجمه آدرس پویا[۱]، آدرس دهی غیر مستقیم کانال[۲]، چند پردازندگی[۳]، محاسبات ممیز شناور با دقت افزوده[۴] و پردازش همزمان چند کانال[۵].

سیستم IBM370 دارای دو حالت «عادی[۶]» و «سرپرست[۷]» می باشد. برنامه‌های کاربر هنگامی که سیستم در وضعیت عادی می‌باشد اجرا می‌شوند. وضعیت سرپرست برای سیستم عامل در نظر گرفته شده است و امکان دسترسی به ثبات ها و دستورات خاص را فراهم می‌کند. بدین ترتیب سیستم عامل می‌تواند اعمالی مانند اداره کردن وقفه ها ، ایجاد یک محیط چند کاربره ، بافرکردن ورودی و خروجی و اداره‌کردن منابع را انجام دهد.

---

[۱] Dynamic address translation

[۲] Channel indirect addressing

[۳] Multiprocessing

[۴] Extended-precision floating point

[۵] concurrent processing of multiple chanenls

[۶] Problem state

[۷] Supervisor state

۱—۱ تاریخچه

۸ آوریل ۱۹۶۴، شرکت IBM ماشین IBM360 را معرفی کرد. ماشین ۳۶۰ یک mainframe با تکنولوژی پیشتاز بود که باعث شد شرکت IBM به بزرگترین سازنده‌ی کامپیوتر تبدیل شود. قبل از IBM360 شرکت IBM بیشتر به خاطر ماشین تحریرِ *Selectric* معروف بود. شرکت IBM برای پروژه‌ی سیستم ۳۶۰ رقمی معادل پنج میلیارد دلار هزینه کرد که ارزش امروزی آن چیزی بالغ بر ۳۰ میلیارد دلار است. رئیس شرکت IBM در حالی این کار را کرد که سود سالیانه شرکت IBM، ۳/۲ میلیارد دلار بود. در آن زمان روزنامه‌ی *Fortune* این حرکت شرکت IBM را قمار پنج میلیارد دلاری نامید. از پنج میلیارد دلار ذکر شده ، یک میلیارد دلار صرف تحقیقات و چهار میلیارد دلار خرج ساختن کارخانه و ساختن سیستم شده بود. شرکت IBM در این قمار پیروز شد چرا که در فاصله سالهای ۱۹۶۴ تا ۱۹۷۰ سود سالیانه‌ی شرکت IBM بیش از ۷/۵ میلیارد دلار شد. از ۳۶۰ برای محاسبات در پیش‌بینی وضع هوا، فیزیک ذرات بنیادی، اکتشافات فضایی و نجوم استفاده شد. از مهمترین خریداران کامپیوترهای IBM370، NASA، بانک‌های آمریکا و Aetna(شرکت بیمه) بودند.

یک مقاله‌نویس PBS technology می‌نویسد :

> شرکت بوئینگ[۸] با ۷۰۷ برای همیشه حمل و نقل هوایی بازرگانی را تغییر داد. بانک آمریکا به وسیله BankAmericaCard که اکنون VISA نامیده می شود سرمایه‌ی فردی را برای همیشه تغییر داد. IBM به وسیله ۳۶۰ صنعت کامپیوتر آمریکا را از بقیه‌ی جهان ۲۰ سال جلو انداخت. پروژه ۳۶۰ شاید بزرگ‌ترین و ریسک‌پذیرترین پروژه در تاریخ تجارت آمریکا بوده است.

قبل از ۳۶۰ شرکت IBM دو خط تولید کامپیوتر داشت. یکی کامپیوترهای ارزان‌تر تجاری ۱۴۰۱ و دیگری کامپیوترهای سری ۷۰۰۰ برای استفاده های علمی. از لحاظ هزینه در دهه ۱۹۶۰ پروژه IBM360 گرانترین پروژه بعد از پروژه آپولو بود.

۱—۲ این پروژه و ساختار پایان‌نامه

معماری ماشین‌های سری ۳۶۰ و ۳۷۰ آغازی بر معماری نوین استفاده‌شده در کامپیوترهای بعدی بود. دلیل این که از معماری ماشین‌های سری ۳۶۰ در نسل‌های بعدی کامپیوترها استفاده‌شد،

---

[۸] BOEING

ساده بودن و در عین‌حال پیشرفتگی و ساختار بجای آن بود. یکی از مهمترین امکاناتی که ماشین‌های ۳۶۰ و ۳۷۰ داشتند راه‌اندازی هزاران کاربر در یک‌زمان بود. ماشین‌های ۳۶۰ و ۳۷۰ خاصیت سازگاری روبه‌بالا[۹] و روبه‌پایین[۱۰] زیادی داشتند. سازگاری روبه‌بالا به این معنی است که یک برنامه‌ی نوشته‌شده در یک ماشین ارزان‌تر قابل اجرا بر روی یک ماشین گران‌تر باشد و سازگاری روبه‌پایین معنی عکس آن را دارد. برای این منظور شرکت IBM در نسل‌های بعدی ماشین‌های خود نیز از جریان‌داده و مجموعه‌دستورالعمل‌های مشابه ماشین ۳۶۰ استفاده‌کرد. در نتیجه‌ی این سیاست شرکت IBM می‌توان کدی را که برای یک ماشین ۳۶۰ نوشته‌شده‌است بر روی ماشین‌های سری Z 900 امروزی اجراکرد. با توجه به موارد ذکرشده بسیاری از دانشگاه‌ها و مراکز آموزشی برای آموزش درس زبان‌ماشین از زبان اسمبلی IBM360/370 استفاده‌می‌کرده و می‌کنند. در سال ۱۹۷۰ در دانشگاه Penn State یک سیستم نرم‌افزاری به نام ASSIST[۱۱] برای ایجاد یک محیط اسمبلر و شبیه‌ساز برای ماشین ۳۷۰ ساخته‌شد. نرم‌افزار ASSIST تنها برای دانشگاه‌ها مقدور بود و بیش از ۱۰۰ نسخه از آن در دانشگاه‌های مختلف نصب شد و با استفاده از آن زبان ماشین در یک درس ۳ واحدی با نام Assembler language programming و باا ستفاده از کتابی با نام Assembler language with assist که ویرایش چهارم آن در حال حاضر موجود می‌باشد آموزش داده‌می‌شد. در سال ۱۹۹۸ دانشگاه Penn State اعلام‌کرد که دیگر این نرم‌افزار copyright ندارد[۱۲] و استفاده از آن برای عموم آزاد است. که البته این سیستم به دلیل قدیمی بودن و عدم ادامه حمایت سازنده بر روی کامپیوترهای شخصی امروزی قابل اجرا نیست.

در این نوشته در فصل دوم ابتدا به معرفی ماشین‌های IBM360/370 در چهارچوب معماری و زبان اسمبلی می‌پردازیم. پروژه انجام‌شده شامل ۳ بخش اصلی اسمبلر ، شبیه‌ساز و ویرایشگر(محیط برنامه) است که هر یک از اجزا را جداگانه در فصل‌های سوم تا پنجم بررسی می‌کنیم. دو ضمیمه بر این نوشته موجود است که ضمیمه‌ی اول دستورات ماشین و عملکردشان را لیست کرده‌است و در ضمیمه‌ی دوم به معرفی ابزار کارت‌پانچ که عمراه برنامه موجود است پرداخته‌شده‌است.

---

[۹] Upward Compatibility

[۱۰] Downward Compatibility

[۱۱] Assembler System for Student Teaching

[۱۲] http://mstack.cs.niu.edu/pub/ASSIST

فصل ۲

معرفی ماشین IBM360/370

در این فصل با معماری ماشین‌های IBM سری ۳۶۰ و ۳۷۰ و زبان اسمبلی آن‌ها بیشتر آشنا می‌شویم.

۲—۱ خصوصیات معماری

سیستم IBM370 از اجزای زیر تشکیل شده است :

- حافظه اصلی : در سیستم ۳۷۰ حافظه اصلی از واحدهای آدرس‌پذیر هشت بیتی تشکیل شده است. طول آدرس ۲۴ بیت است در نتیجه سیستم می‌تواند تا ۱۶۷۷۷۲۱۶ بایت را آدرس‌دهی کند. بایت هادر کنار هم نیم‌کلمه[۱] به طول دو بایت، کلمه[۲] به طول چهار بایت و کلمه مضاعف[۳] به طول هشت بایت را می‌سازند.

- یک یا چند CPU : سیستم ۳۷۰ قابلیت پردازش موازی با استفاده از چند CPU را دارا می‌باشد. در داخل CPU پنج واحد پردازش وجود دارد:

۱) حافظه‌ی داخل پردازنده : شامل سه دسته ثبات‌های کلیِ کنترلی، عمومی و ممیزشناور می‌شود. پردازنده دارای ۱۶ ثبات عمومی به طول ۳۲ بیت، ۴ ثبات ممیز شناور

[۱] Half-word
[۲] Word
[۳] Double-word

به طول ٦۴ بیت، ١٦ ثبات کنترلی به طول ۳۲ بیت و همچنین یک ثبات وضعیت برنامه به طول ٦۴ بیت می باشد.

۲) کنترل گذرگاه حافظه

۳) پردازنده انجام دستورات

۴) پردازنده محاسبات ممیز ثابت (پشتیبانی سخت افزار سیستم ۳۷۰ از اعداد دهدهی در حال حاضر تنها در معماری‌های پیشرفته کامپیوتر یافت می‌شود.)

۵) پردازنده محاسبات ممیز شناور

- کانال‌های سلکتور و مالتی‌پلکسر[۴] : کانال‌ها به CPU ، حافظه اصلی و دستگاه‌های ورودی/خروجی — معمولا بوسیله یک میانه ورودی/خروجی[۵]— وصل هستند. وظیفه کانال‌ها این است که از ارتباط مستقیم ورودی/خروجی با CPU جلوگیری کنند تا بار CPU کمتر شود. این کمک می‌کند تا پردازش داده به صورت موازی با عملیات ورودی/خروجی انجام‌شود. یک کانال ممکن است یک واحد مستقل دارای حافظه و قابلیت پردازش باشد یا از لحاظ فیزیکی در داخل خود CPU باشد. در هر دو حالت کانال‌ها از لحاظ عملکرد تفاوتی ندارند. سیستم ۳۷۰ سه نوع کانال دارد: کانال مالتی‌پلکسر ترتیبی[٦] ، کانال مالتی‌پلکسر بلاکی[۷] و کانال انتخابی[۸].

- دستگاه‌های ورودی/خروجی که بوسیله واحد کنترل به کانال‌ها وصل شده‌اند مانند کارت‌خوان ، دستگاه‌پانچ ، نوار مغناطیسی ، دیسک ، صفحه‌کلید ، دستگاه چاپگر و غیره.

دستوراتی که پردازنده اجرا می‌کند را می‌توان در پنج کلاس طبقه‌بندی کرد: دستورات کنترل سیستم ، دستورات عمومی ، دستورات دهدهی ، دستورات ممیز شناور و دستورات ورودی/خروجی. دستورات کنترل سیستم و دستورات ورودی/خروجی دستورات خاص هستند که تنها هنگامی که سیستم در وضعیت سرپرست می‌باشد قابل اجرا هستند. دستورات عمومی برای انجام اعمال ممیزثابت ، منطقی ، انشعابی و کنترلی استفاده‌می‌شوند. دستورات دهدهی بر روی اعداد دهدهی و دستورات ممیزشناور بر روی اعداد ممیزشناور اعمال می‌شوند.

---

[۴] Selector and multiplexer channels

[۵] I/O interface

[٦] Byte-multiplexer channel

[۷] block-multiplexer channel

[۸] Selector channel

۲—۲ اجرای دستورها

به صورت عادی اجرای اجرای یک برنامه نوشته‌شده توسط کاربر با اجرای ترتیبی دستورهایش صورت‌می‌پذیرد. این ترتیب توسط کلمه وضعیت برنامه (PSW) کنترل‌می‌شود که همواره آدرس دستور بعدی را که باید از حافظه خوانده و اجرا شود را در خود دارد. توسط انشعاب[۹] ، وقفه و یا دستور LOAD PSW می‌توان اجرای ترتیبی دستورها را تغییرداد. در سیستم واقعی IBM370 سیستم‌عامل برنامه‌ای است که به صورت یک حلقه‌ی بی‌نهایت در حال اجراشدن در ماشین است و هر برنامه‌ای که کاربر می‌خواهد اجراکند به صورت یک پروسس است که سیستم عامل به آن انشعاب‌می‌کند. با اتمام اجرای برنامه ، کنترل دوباره به سیستم عامل بازگردانده می‌شود. اما در شبیه‌سازی که نوشته‌شده‌است برنامه‌ی سیستم عامل در جایی از حافظه موجود نیست تا برنامه با دادن آدرس دستوری از آن به آن انشعاب‌کند تا کنترل به سیستم‌عامل بازگردانده‌شود. پس ناگزیر من یک دستور راهنما[۱۰] به اسم RET تعریف‌کردم که هر کجا کاربر برنامه‌نویس خواست برنامه را به پایان برساند از این دستورِراهنما استفاده‌می‌کند.

هر دستور از دو بخش اصلی تشکیل‌شده‌است: (۱) عملگر که عملی را که باید انجام‌بگیرد مشخص می‌کند و (۲) عملوندها.

۲—۲—۱ عملوندها

عملوندها را می‌توان به سه گروه کلی تقسیم‌کرد :

۱) عملوندهایی که نشان‌دهنده محتوای یک ثبات هستند.

۲) عملوندهای مستقیم[۱۱]

۳) عملوندهایی که نشان‌دهنده محتوای محلی در حافظه می‌باشند.

عملوندهای نوع اول ممکن‌است نشان دهنده یک ثبات عمومی ، ممیزشناور و یا کنترلی باشند که بسته به نوع دستور مشخص می‌شود. ثبات مورد نظر با ۴ بیت در کد زبان ماشین دستور مشخص می‌شود که به آن فیلد R می‌گویند. گاه پیش می‌آید که یک عملوند در یک ثبات

---

[۹] Branching

[۱۰] Directive

[۱۱] Direct

قراردارد که شماره ثبات در فیلد R آمده‌است. عملوندهای مستقیم به طور مستقیم در داخل کد ماشین قراردارند. عملوندهای مستقیم در کد ماشین داخل فیلد I به طول ۸ بیت قرارمی‌گیرند.

و بالاخره آخرین نوع عملوندها ، عملوندهای حافظه‌ای می‌باشند که ممکن است بسته به نوع دستور طول مشخص داشته‌باشند و یا طول آن‌ها در یک فیلد دیگر مانند M[۱۲] و یا فیلد L تعیین شده‌باشد. آدرس محلی از حافظه که عملوند به آن اشاره‌می‌کند توسط سه فیلد مبنا[۱۳] ، شاخص[۱۴] و جابجایی[۱۵] مشخص می‌شود. آدرس از جمع محتوای ثبات مبنا با محتوای ثبات شاخص و ۱۲ بیت جابجایی به دست می آید.

۲—۲—۲ قالب دستورها

یک دستور یک، دو و یا سه نیم‌کلمه طول دارد. هر دستور در یکی از شش قالب پایه RR ، RX ، RS ، SI ، SS و S می‌باشد. در پنج قالب اول یک بایت اول حاوی opcode می‌باشد و در قالب S ، opcode در دو بایت اول قرار گرفته‌است.

عملوندهای ثباتی

در قالب‌های RR ، RX و RS عملوند اول محتوای ثبات مشخص شده توسط فیلد R_1 می‌باشد. در قالب RR ، فیلد R_2 نشان‌دهنده عملوند دوم می‌باشد. بسته به دستور ممکن است فیلد R_3 نیز موجود باشد. فیلد R نشان‌دهنده یک ثبات عمومی در دستورهای عمومی است و نشان‌دهنده یک ثبات ممیزشناور در دستورات ممیزشناور می‌باشد.

عملوندهای مستقیم

در قالب SI ، محتوای ۸ بیتی فیلد I_2 نشان‌دهنده عملوند دوم است. B_1 و D_1 با هم عملوند اول را مشخص می‌کنند که آدرس محلی از حافظه می‌باشد.

---

[۱۲] Mask
[۱۳] Base
[۱۴] indeX
[۱۵] Displacement

عملوندهای حافظه

در قالب‌های SI و SS ، محتوای ثبات مشخص‌شده توسط فیلد B_1 با محتوای ۱۲ بیتی فیلد D_1 جمع می‌شود تا عملوند اول را شکل‌دهد. همچنین در قالب‌های S ، RS و SS محتوای ثبات مشخص‌شده توسط B_2 با D_2 جمع می‌شود تا عملوند دوم را شکل‌دهد. در قالب RX ، محتوای ثبات‌های B_2 و X_2 با محتوای ۱۲ بیتی فیلد D_2 جمع می‌شوند و حاصل عملوند دوم است که مشخص‌کننده آدرس محلی از حافظه می‌باشد. در قالب SS با دو فیلد ۴ بیتی L_1 و L_2 ، فیلد L_1 مشخص‌کننده طول عملوند اول منهای یک و فیلد L_2 مشخص‌کننده طول عملوند دوم منهای یک می‌باشد. در این نوع دستور طول هر یک از عملوندها می‌تواند بین ۱ تا ۱۶ باشد. در قالب SS با یک فیلد L ، فیلد L مشخص‌کننده طول عملوند اول منهای یک می‌باشد. در این نوع دستور طول عملوند اول ۱ تا ۲۵۶ متغیر می‌باشد.

محاسبه آدرس: آدرس استفاده‌شده برای نشانی‌دهی به یک محل از حافظه ، یا داخل یک ثبات می‌باشد یا به طریق زیر محاسبه‌می‌گردد:

آدرس مبنا یک عدد ۲۴ بیتی داخل یک ثبات عمومی مشخص‌شده توسط فیلد ۴ بیتی B در دستور می‌باشد. آدرس مبنا اصولا برای آدرس‌دهی مستقل به یک محل از حافظه استفاده می‌شود. به عنوان مثال در محاسباتی که از آرایه استفاده می‌شود می‌توان از آن برای مشخص‌کردن آدرس ابتدای آرایه استفاده‌نمود. با آدرس مبنا به تنهایی می‌توان تمام حافظه را آدرس‌دهی کرد.

شاخص یک عدد ۲۴ بیتی داخل یک ثبات عمومی است که توسط فیلد ۴ بیتی X در دستور مشخص می‌شود. فیلد X تنها در دستورهای با قالب RX وجوددارد. از شاخص برای شاخص‌دهی استفاده می‌شود به عنوان مثال در آرایه آدرس هر خانه‌ای را که به نحوی مد نظر است را می‌توان با آدرس شاخص معلوم‌کرد.

جابجایی یک عدد ۱۲ بیتی است که در فیلد D دستور قرار گرفته‌است. جابجایی آدرس نسبی تا ۴۰۹۵ بایت را ممکن می‌سازد. در مثال آرایه می‌توان از جابجایی برای نشانی‌دهی به هر یک از اجزای یک خانه استفاده‌کرد. مثلا در پردازش رکوردها می‌توان از جابجایی برای نشانی‌دهی به اول هر یک از فیلدهای یک رکورد استفاده‌نمود.

مجموع سه عدد ذکر شده در بالا آدرس کل را تشکیل می‌دهند که یک عدد ۲۴ بیتی مثبت می‌باشد. سه عدد بالا با هم جمع می‌شوند و از سرریز جمع آن‌ها نیز صرف نظر می‌شود. چنانچه محتوای هر یک از فیلدهای X و یا B صفر باشد به معنای صرف نظر کردن از آن

آدرس است و به معنای ثبات شماره صفر نمی‌باشد. در شکل زیر قالب‌های دستورها را مشاهده می‌کنید.

۲—۲—۳ کلمه وضعیت برنامه (PSW)

کلمه وضعیت برنامه یک ثبات داخل CPU می‌باشد با طول ۶۴ بیت که اطلاعات مورد نیاز را برای اجرای برنامه دارا می‌باشد. PSW شامل آدرس دستور بعدی ، کد وضعیت [۱۶] و فیلدهای دیگر می‌باشد. به طور کلی PSW برای کنترل اجرای ترتیبی برنامه و نگهداشتن وضعیت برنامه در حال اجرا می‌باشد. در ماشین‌های ۳۶۰ کلمه وضعیت برنامه تنها یک حالت دارد ولی در ماشین‌های ۳۷۰ کلمه وضعیت برنامه می‌تواند در یکی از دو وضعیت کنترلی کنترل پایه[۱۷] (BC) و یا کنترل گسترش‌یافته[۱۸] (EC) باشد. این که PSW در کدام وضعیت است توسط بیت شماره ۱۲ آن معلوم می‌یاشد. در حالت BC ، کلمه وضعیت برنامه قالب مشابه سیستم ۳۶۰ را دارد. در ادامه قالب کلمه وضعیت برنامه در حالت BC را شرح می‌دهیم :

چنانچه بیت ۱۲ PSW صفر باشد PSW در حالت BC می‌باشد.

- بیت‌های ۰—۵: کنترل می‌کنند آیا وقفه برای کانال‌های ۰ تا ۵ فعال می‌باشد یا نه. وقتی که بیت کانال مورد نظر یک باشد در شرایطی می‌تواند موجب وقفه ورودی/خروجی شود.
- بیت ۶: مشخص می‌کند که آیا کانال‌های ۶ به بعد موجب وقفه می‌شود یا خیر.
- بیت ۷: اتفاق افتادن و یا نیفتادن وقفه خارجی را کنترل می‌کند.
- بیت‌های ۸—۱۱: کلید محافظتی CPU هستند. این کلید با کلید ذخیره‌شده در حافظه هنگام ذخیره و یا خواندن اطلاعات تطبیق داده می‌شود.
- بیت ۱۲: همانطور که قبلا شرح داده شد چنانچه ۱۲ باشد CPU در حالت EC و چنانچه صفر باشد CPU در حالت BC می‌باشد.
- بیت ۱۳: کنترل می‌کند که وقفه وارسی ماشین[۱۹] اتفاق می‌افتد یا خیر.

---

[۱۶] Condition Code

[۱۷] Base Condition

[۱۸] Extended Condition

[۱۹] Machine check interruption

- بیت ۱۴: اگر این بیت یک باشد CPU در حالت انتظار[۲۰] است. در غیر این صورت در حالت اجرایی[۲۱] است.
- بیت ۱۵: وقتی این بیت یک باشد CPU در حالت عادی است و چنانچه صفر باشد در حالت سرپرست می‌باشد.
- بیت‌های ۱۶ تا ۳۱: کد وقفه را تشکیل می‌دهند.
- بیت‌های ۳۲ و ۳۳: طول آخرین دستور اجراشده را مشخص می‌کند.
- بیت‌های ۳۴ و ۳۵: کد وضعیت هستند.
- بیت‌های ۳۶ تا ۳۹: ماسک برنامه[۲۲] می‌باشند، و بالاخره
- بیت‌های ۴۰ تا ۶۳: آدرس دستور بعدی را که باید از حافظه آورده شود را نشان می‌دهد(که همان شمارنده برنامه می‌باشد).

۲—۳ قالب زبان اسمبلی

زبان اسمبلی IBM370 مانند تمام زبان‌های کامپیوتری دارای یک سری قواعد می‌باشد که به مجموعه این قواعد قالب زبان می‌گویند و برنامه‌نویس باید برنامه خود را در این قالب بنویسد و یا به عبارتی از این قواعد پیروی کند.

ماشین IBM370 متعلق به دهه ۷۰ میلادی می‌باشد و در آن زبان برنامه به صورت کارت‌های منگنه که ۸۰ ستون و ۱۲ سطر داشتند به ماشین داده‌می‌شد. هر کارت معمولا حاوی یک حکم زبان اسمبلی بود یا به عبارتی هر خط برگ برنامه‌نویسی نشانگر یک کارت پانچ بود. پس هر خط زبان حداکثر طول ۸۰ دارد منتها در بعضی موارد یک حکم در یک خط جا نمی‌شود. برای ادامه‌دادن یک حکم در خط بعد کافی‌است در ستون ۷۲ یک کاراکتر غیر از کاراکتر فاصله بگذاریم و در خط بعد دستور را از ستون ۱۶ آغاز کنیم. یک دستور در زبان اسمبلی ۳۷۰ حداکثر سه خط می‌باشد.

---

[۲۰] Wait state

[۲۱] Running state

[۲۲] Program mask

۲—۳—۱ فیلدهای زبان اسمبلی

هر دستور در زبان اسمبلی چهار فیلد دارد : فیلد اسم(برچسب)[۲۳] ، فیلد عملیات[۲۴] ، فیلد عملوندها[۲۵] و فیلد توضیحات[۲۶]. هر دو فیلد توسط حداقل یک فاصله خالی از همدیگر جدامی‌شوند.
فیلد اسم ، در صورت وجود ، اسم نمادی موجود در این فیلد را به مکان حافظه‌ای که دستور آن خط در آن قرار خواهدگرفت نسبت‌می‌دهد. اسمبلر یک جدولی از نمادها و مقادیرشان دارد که این برچسب‌ها وارد آن لیست می‌شوند و می‌توان در دستورات دیگر از این برچسب‌ها به عنوان عملوند آدرس استفاده‌کرد.

یک برچسب دارای حداکثر طول ۸ کاراکتر می‌باشد. اولین کاراکتر فیلد اسم باید یکی از کاراکترهای A-Z و یا یکی از کاراکترهای $ ، # و یا @ باشد. بقیه کاراکترها می‌توانند از کاراکترهای A-Z ، $ ، # ، @ و یا 0-9 باشند. اگر در دستوری فیلد اسم وجود داشته‌باشد باید از ستون اول آغاز شود در غیر این صورت ستون اول باید خالی باشد. در صورتی که در ستون اول کاراکتر '*' باشد تمام این خط توضیحات است. در صورت وجود فیلد اسم ، فیلد عملیات با یک فاصله بعد از آن آغازمی‌شود. فیلد عملیات می‌تواند یک opcode از زبان ماشین ، یک راهنما به اسمبلر ، نام یک درشت دستورالعمل[۲۷] تعریف‌شده و یا نوع تعریف یک ثابت باشد.

حداقل یک فاصله خالی فیلد عملیات را از فیلد عملوندها در صورت وجود جدا می‌کند. عملوندها معمولا آدرسی از حافظه و یا شماره یک ثبات می‌باشند. عملوندها در فیلد عملوند با ویرگول و بدون فاصله از یکدیگر جدامی‌شوند. فیلد چهارم فیلد توضیحات است که اجباری نمی‌باشد و در صورت وجود با یک فاصله بعد از فیلد عملوند می آید. فیلد توضیحات در ستون ۷۱ خاتمه‌می‌یابد و در صورت ادامه در خط بعدی ادامه پیدامی‌کند.

۲—۳—۲ خطاهای قالب زبان اسمبلی IBM370

شرح خطاهایی که در صورت وجود ایراد در قالب زبان برنامه نوشته‌شده بروزمی‌کند در زیر آمده‌است:

---

[۲۳] Label
[۲۴] Operation
[۲۵] Operands
[۲۶] Comment
[۲۷] Macro

[In line (line_number) the source should be shorter than 8] :

در صورت بیشتر از ۸ کاراکتر بودن طول فیلد اسم در یک دستور این خطا در برنامه بروزمی‌کند.

[The line number (line_number) has a length more than 72] :

در صورت بیشتر از ۸۰ بودن طول یک خط از برنامه این خطا بروزمی‌کند که البته کاربر در هنگام کار با برنامه مشاهده‌خواهدکرد که ویرایشگر برنامه خود از پیشامدن این امر جلوگیری می‌کند.

[The line number (line_number) should start from place 16] :

در صورتی که کاراکتر ۷۲ کاراکتر فاصله و یا خالی نباشد خط بعد باید از ستون ۱۶ آغازشود. چنانچه این قاعده رعایت‌نشده‌باشد این خطا بروزمی‌کند.

فصل ۳

اسمبلر

این اسمبلر به زبان C++ و در محیط NET. نوشته شده‌است و قابل اجرا در محیط Microsoft Windows می‌باشد. در ضمن همین اسمبلر با کمی تغییر در محیط LINUX هم ساخته‌شده‌است و قابل اجرا می‌باشد.

۳—۱ ساختار کلی اسمبلر

برای نوشتن اسمبلر از روش دو گذر [۱] استفاده‌شده‌است.

گذر اول شامل مراحل زیر می‌باشد:

۱) خواندن فایل ورودی و بررسی درستی برنامه نوشته‌شده از نظر شیوه‌نوشتن زبان ماشین.

۲) تعیین‌کردن نوع هر خط برنامه که شامل انواع دستورهای ماشین، تعریف متغیرها، راهنمایی به اسمبلر، درشت‌دستورالعمل‌ها و توضیحات می‌باشد.

۳) بسط درشت دستورالعمل‌ها و جایگزین‌کردن فراخوانی درشت‌دستورالعمل‌ها با تعریف آن.

و گذر دوم شامل مراحل زیر است:

۱) محاسبه آدرس هر خط از برنامه.

۲) محاسبه‌کردن مقادیر ثابت‌ها و تهیه جدول سمبل‌ها.

---

[۱] Two Path

۳) انجام عملیات مربوط به Literalها.

۴) جایگزین‌کردن سمبل‌ها در دستورات و ایجاد کد ماشین.

در محلی که برنامه نصب‌می شود دو فهرست‌راهنما[۲] وجود دارد، یکی برای نگهداری کتابخانه درشت‌دستورالعمل‌هاست و دیگری برای نگهداری فایل‌هایی که در طول اجرای برنامه ساخته می‌شوند.

۳—۲ تشریح نحوه نوشتن برنامه

در این بخش به شرح بیشتر پیاده‌سازی قسمت‌هایی قابل ذکر از برنامه می‌پردازیم.

۳—۲—۱ پیاده‌سازی درشت‌دستورالعمل‌ها

دو دسته درشت‌دستورالعمل در برنامه قابل استفاده هستند. دسته اول درشت‌دستورالعمل‌های موجود در کتابخانه‌ی درشت‌دستورالعمل‌ها می‌باشد. درشت‌دستورالعمل‌های متعلق به این دسته از پیش تعریف شده‌اند و در داخل فایل راهنمای « macro_list » در محلی که assembler نصب‌شده‌است موجود می‌باشند. کاربر می‌تواند درشت‌دستورالعمل‌های جدیدی را تعریف کرده در این مکان ذخیره‌کند. دسته دوم درشت‌دستورالعمل‌های داخل خود برنامه نوشته‌شده توسط کاربر می‌باشد.

assembler دارای یک آرایه از نام درشت‌دستورالعمل‌ها و محل ذخیره آن‌ها می‌باشد و در هر جای برنامه که با نام درشت‌دستورالعمل برخورد کرد ، آن را با کد زبان اسمبلی درشت‌دستورالعمل مورد نظر جایگزین‌می‌کند.

۳—۲—۲ جایگزین‌کردن نمادها

اسمبلر دارای یک آرایه از نماد[۳]ها می‌باشد که هر element از این آرایه از سه جزء تشکیل‌شده‌است : نام نماد ، طول نماد و مقدار نماد. در طول برنامه هر کجا که یک نماد تعریف شود ، اسمبلر آن را به لیست نمادها اضافه‌می‌کند و در طول برنامه چنانچه آن نماد استفاده شده‌باشد با توجه

---

[۲] Directory

[۳] Symbol

به طول و مقدار نماد مورد نظر ، جایگزینی مناسب را انجام می‌دهد. نمادها ممکن است در فیلد اسم یک دستور یا دستورات راهنمای EQU ، DC و یا DS تعریف شوند.

۳–۲–۳ جایگزین کردن عبارات ریاضی با مقدارشان

اسمبلر نوشته‌شده این توانایی را دارد که هر کجا به جای اجزای صریح یک عدد به آن یک عبارت ریاضی شامل ضرب ، تقسیم ، جمع ، تفریق و نقیض[۴] داده‌شود. پیاده‌سازی این قسمت توسط Stack و برنامه‌ی بازگشتی و با استفاده از مفاهیم زبان‌های مستقل از متن[۵] می‌باشد.

۳–۳ چگونگی ورودی و خروجی

ورودی اسمبلر یک برنامه نوشته‌شده به زبان اسمبلی می‌باشد که در یک فایل که به اسمبلر به عنوان ورودی داده‌می‌شود موجود است. چنانچه نام فایل ورودی داده‌نشود اسمبلر به صورت خودکار فایل با نام «in.dat» موجود در محلی که اسمبلر نصب‌شده‌است را بازمی‌کند. نام فایل خروجی نیز به اسمبلر داده‌می‌شود. در غیر این صورت اسمبلر خروجی خود را در فایل با نام «out.dat» در محل نصب‌شدن اسمبلر می‌ریزد. چنانچه برنامه‌ی ورودی مشکل داشته‌باشد و دارای خطای برنامه‌نویسی باشد اسمبلر در فایل خروجی دلیل خطاها و محل آن‌ها را می‌نویسد در غیر این صورت با توجه به پرچم‌هایی که به اسمبلر داده‌شده‌است نوع خاصی از خروجی را ایجاد می‌کند.
انواع خروجی عبارتند از:

۱) لیست اسمبلی برنامه نوشته‌شده : لیست اسمبلی شامل شش ستون می‌باشد. ستون اول شماره‌ی خط در لیست اسمبلی است. ستون دوم آدرس آغاز کد ماشین ایجادشده برای هر خط از برنامه‌ی اسمبلی نوشته‌شده را نشان می‌دهد. ستون سوم کد ماشین ایجادشده برای هر خط از برنامه‌ی نوشته‌شده‌است. ستون چهارم شماره‌ی خط واقعی بعد از تبدیل و جایگزینی درشت‌دستورالعمل‌ها با برنامه‌ی اسمبلی معادل آن‌ها را نشان‌می‌دهد. چنانچه در خطی از برنامه نمادی تعریف‌شده‌باشد آن نماد در ستون پنجم نمایش‌داده‌می‌شود. و بالاخره ستون ششم که برنامه‌ی نوشته‌شده در هر خط را نشان می‌دهد.

---

[۴] Not

[۵] Context-free languages

۲) دامپ[۶] از حافظه : این خروجی شامل لیست اسمبلی به علاوه ۶۴kb از محتوای حافظه تبدیل‌شده به hex از ابتدای محل تولید کد می‌باشد.

۳) خروجی به صورت Object برای استفاده شبیه‌ساز[۷] : این خروجی شامل اطلاعات مورد نیاز برای شبیه‌سازی می‌باشد تا با خواندن این فایل بتواند حافظه را به‌روز کند و شروع به اجرای برنامه کند.

در ادامه به گزینه‌های مختلف اجرای فایل اجرایی اسمبلر می‌پردازیم. برای اجرای فایل exe گزینه‌های مختلفی وجوددارد شامل موارد زیر:

۱) گزینه « assembler » : از فایل in.dat در محلی که فایل اجرایی وجود دارد برنامه را می‌خوند و چنانچه برنامه نوشته‌شده مشکلی نداشته باشد لیست اسمبلی را در فایل out.dat در همان محل می‌نویسد. در صورتی که برنامه مشکل داشته‌باشد مشکل را در همان فایل out.dat می‌نویسد.

۲) گزینه « assembler -help » : گزینه‌های اجرا کردن فایل assembler.exe را نمایش می‌دهد.

۳) گزینه « assembler file_in_path file_out_path » : فایل ورودی و خروجی را می‌گیرد.

۴) گزینه « assembler file_in_path » : فایل ورودی را می‌گیرد.

۵) گزینه « assembler -in file_in_path » : فایل ورودی را می‌گیرد.

۶) گزینه « assembler -out file_out_path » : فایل خروجی را می‌گیرد.

۷) گزینه « assembler -dump » : علاوه بر لیست اسمبلی محتوای حافظه از آدرس شروع برنامه تا 0X000FFFF0 بایت بعد از آن را نیز در فایل خروجی می‌نویسد.

۸) گزینه « assembler -dump file_in_path file_out_path » : مانند گزینه قبل با این تفاوت که فایل ورودی و خروجی را نیز دریافت‌می کند.

۹) گزینه « assembler -obj » : به عنوان فایل خروجی ، فایلی حاوی کد ماشین برنامه نوشته‌شده تولیدمی‌کند که به عنوان ورودی به شبیه‌ساز داده‌می‌شود و شبیه‌ساز توسط آن دستورات را خط به خط اجرا می‌کند.

---

[۶] Dump

[۷] Simulator

۱۰) گزینه « assembler -obj file_in_path file_out_path » : مانند گزینه قبل با این تفاوت که فایل ورودی و خروجی را نیز دریافت‌می‌کند.

۳—۴ نمونه‌ی خطاها

اسمبلر در مراحل مختلفی در طی گذرهایش برنامه نوشته‌شده را مورد بررسی قرار می‌دهد و چنانچه در محلی مشکلی را مشاهده‌کند آن را در قالب یک پیغام خطا اعلام‌می‌کند. چک کردن اسمبلر در طی پنج مرحله انجام می‌شود که در ادامه مراحل و خطاهای هر کدام را توضیح می‌دهیم:

۳—۴—۱ مرحله باز کردن فایل ورودی و خواندن آن

در این گام اسمبلر فایل ورودی که برنامه در آن است و همچنین فایل‌های کتابخانه درشت‌دستورالعمل‌ها را باز کرده و اطلاعات موجود در آن‌ها را می‌خواند. شرح خطاها به صورتی که در ادامه می‌آید می‌باشد:

[file could not be opened] :

در صورتی که فایل ورودی پیدا نشود و یا مشکلی در بازکردن آن باشد این پیغام داده‌می‌شود.

[list file of macros could not be opened] :

در کتابخانه درشت‌دستورالعمل‌ها فایلی وجود دارد که لیست درشت‌دستورالعمل‌های موجود در کتابخانه در این فایل می‌باشد. ابتدا اسمبلر سعی می‌کند این فایل را load کند تا با توجه به اطلاعات آن درشت‌دستورالعمل‌ها را از فایل‌های کتابخانه فراخوانی کند. چنانچه این فایل را پیدا نکند و یا نتواند ان را باز کند این پیغام را می‌دهد. ایجاد تغییر توسط برنامه‌ای غیر از اسمبلر ممکن است موجب ایجاد مشکلاتی اساسی در اجرای برنامه شود.

۳–۴–۲ مرحله تشخیص نوع هر خط از برنامه

در این مرحله اسمبلر دو عمل مهم را انجام می‌دهد : (۱) تعیین نوع هر خط از برنامه و (۲) جایگزین‌کردن فراخوانی درشت‌دستورالعمل‌ها با تعریف آن‌ها خواه در متن برنامه خواه در کتابخانه درشت‌دستورالعمل‌ه.ا خطاهایی که در این مرحله ممکن است اعلام‌شوند به صورت زیر می‌باشند:

[problem in operands in line (line_number)] :

چنانچه در تعریف یک درشت‌دستورالعمل در برنامه پارامترهایش دچار مشکل باشند این خطا بروزمی‌کند.

[problem in macro call in line (line_number)] :

در صورتی که فراخونی یک درشت‌دستورالعمل صحیح نباشد مثلا تعداد پارامترهای آن از تعداد پارامترهایی که تعریف درشت‌دستورالعمل می‌گیرد کمتر باشد این خطا بروزمی‌کند.

[Warning : ignored arguments of macro call in line (line_number)] :

چنانچه تعداد پارامترهای فراخوانی درشت‌دستورالعمل از تعداد پارامترهای تعریف آن بیشتر باشد پارامترهای اضافی نادیده گرفته‌می‌شوند و این اخطار داده می‌شود.

۳–۴–۳ مرحله محاسبه آدرس هر خط

در این مرحله علاوه بر محاسبه آدرس خط‌های برنامه عملیات مهم دیگری نیز شامل (۱) انجام عملیات لازم در نقاطی که از دستورهای راهنمای EQU ، DS و یا DC استفاده‌شده‌است و (۲) همچنین جایگزین‌کردن علامت * در نقاطی که به عنوان عملوند آدرس استفاده‌شده‌است با آدرس آن خط نیز انجام‌می‌شود. خطاهایی که در این مرحله امکان رخ‌دادن دارند عبارتند از:

[unknown expression in line number (line_number)] :

این اسمبلر دارای توانایی محاسبه عبارت‌های ریاضی استفاده‌شده در هر کجای برنامه می‌باشد. به این معنی که برنامه‌نویس در هر کجای برنامه می‌تواند به جای یک مقدار از یک عبارت

ریاضی استفاده‌کند. مثلا در عملوندهای آدرس کاربر می‌تواند یک عبارت ریاضی بنویسد تا آدرس از روی آن محاسبه‌شود. چنانچه یک عبارت ریاضی نوشته‌شده توسط کاربر دارای مشکل باشد این پیغام داده‌می‌شود. مشکلات ممکن در یک عبارت ریاضی می‌تواند استفاده نادرست از عملوندهای جمع ، تفریق ، ضرب و تقسیم و یا پرانتزگذاری نادرست باشد.

: [problem in constant definition in line (line_number)]

چنانچه در تعریف یک ثابت در دستورات راهنمای DC و DS مشکلی وجود داشته‌باشد این پیغام داده‌می‌شود. از خطاهای معمول می‌توان به استفاده از کلمه‌های DC و یا DS بدون عملوند اشاره‌کرد.

۳—۴—۴ مرحله جایگزین‌کردن نمادها در عملوندها

در طول برنامه نمادهایی توسط کاربر تعریف می‌شوند. هر نماد شامل سه داده نام نماد ، طول نماد و مقدار نماد می‌باشد. اسمبلر در هر کجای برنامه که نمادی تعریف می‌شود آن را به جدول نمادها اضافه‌می‌کند و چنانچه در جایی از برنامه از آن نماد استفاده‌شده‌باشد مقدار آن را جایگزین فراخوانی‌اش می‌کند. خطاهای زیر می‌تواند در این مرحله به صورت زیر گزارش شود:

: [Problem in instruction definition in line (line_number)]

اگر یک دستور زبان اسمبلی نادرست استفاده‌شده‌باشد این خطا روی‌می‌دهد. به عنوان مثال تعداد یا نوع عملوندهای یک دستور اشتباه‌باشد.

۳—۴—۵ مرحله محاسبه‌کردن مقدار عملوندها و تولید کد ماشین

در این مرحله با محاسبه‌کردن مقدار عملوندها توسط اسمبلر، همه چیز برای ایجاد کد ماشین آماده‌است. خطاهای قابل گزارش در این مرحله عبارتند از:

: [Problem in instruction definition in line (line_number)]

به این خطا در مرحله قبل نیز اشاره‌کردیم. این خطا معمول‌ترین خطا در این مرحله است.

[unknown opcode in line number (line_number)] : این خطا هنگامی بروز می‌کند که در یک خط از برنامه دستوری استفاده‌شده‌باشد که در زبان اسمبلی IBM370 تعریف‌شده‌نباشد.

[problem in definition of opernad 1 in line (line_number)] : چنانچه عملوند اول از یک دستور دچار مشکل باشد این پیغام داده‌می شود.

[problem in definition of opernad 2 in line (line_number)] : مانند خطای بالا با این تفاوت که در مورد عملوند دوم است.

[problem in definition of opernad 3 in line (line_number)] : مانند دو خطای بالا و برای دستوراتی که سه عملوند دارند در صورت مشکل‌دار بودن عملوند سوم روی‌می‌دهد.

[problem in definition of opernad in line (line_number)] : این خطا تنها در صورت وجود مشکل در تنها عملوند دستورهای نوع S روی‌می‌دهد.

فصل ۴

شبیه‌ساز

شبیه‌ساز ماشین ۳۷۰ به زبان C++ و در محیط NET. نوشته‌شده‌است. به منظور شبیه‌سازی عملکرد ماشین ، ساختار ماشین در چند سطح به صورت شیء گرا پیاده‌سازی شده‌است. در بخش زیر ساختار برنامه نوشته‌شده را بررسی می‌کنیم.

۴—۱ ساختار کلی شبیه‌ساز ماشین ۳۷۰

در پایین‌ترین سطح ، سخت‌افزار ماشین قرار دارد که به کمک یک کلاس شبیه‌سازی شده‌است. این کلاس در فیلدهایش یک آرایه‌ی ۱۶ تایی برای ثبات‌های عمومی ، یک آرایه‌ی ۱۶ تایی برای ثبات‌های کنترلی و یک آرایه‌ی ۸ تایی برای ثبات‌های ممیزشناور می‌باشد. حافظه نیز به صورت یک آرایه به طول ۱۶۷۷۷۲۱۶ پیاده‌سازی شده‌است. از دیگر فیلدهای مهم این کلاس PSWهای قدیمی و جدید و PSWی فعلی است.

در سطح بعدی دستورات ماشین قرار دارند. برای پیاده‌سازی هر دستور از یک تابع استفاده‌شده‌است. هر کدام از توابعی که دستوری از سیستم را شبیه‌سازی می‌کنند به عنوان ورودی حالت فعلی ماشین شامل وضعیت ثبات‌ها ، حافظه و کلمه‌ی وضعیت برنامه را دریافت‌می‌کنند. سپس با توجه به عملکردی که دارند روی وضعیت سیستم تأثیر می‌گذارند و آن را به حالت جدیدی می‌برند.

سطح سوم که بالاترین سطح نیز می‌باشد سطح شبیه‌سازی است. در این بخش برنامه فایل object تولیدشده توسط اسمبلر را می‌خواند و بر اساس آن حالت اولیه‌ی سیستم و محلی که اجرای دستورات باید از آنجا آغاز شود را تعیین می‌کند. با شروع شبیه‌سازی برنامه قابلیت

این را دارد که دستورات را به صورت تک‌پله‌ای[1] و یا پشت سر هم[2] اجرا کند. برای اجرای دستورات شبیه‌ساز یک بایت از حافظه در محلی که PSW نشان می‌دهد می‌خواند و تشخیص می‌دهد که این بایت opcodeِ چه دستوری‌است. با فهمیدن نوع دستور بقیه‌ی دستور را با توجه به طول معلوم آن از حافظه واکشی می‌کند. سپس این دستور را اجرا می‌کند و دستور بعدی را از محلی که در مقدار جدیدِ PSW به آن اشاره می‌شود واکشی می‌کند. با توجه به این که وقتی نوع دستور از روی opcodeِ آن فهمیده‌شد فیلدهای آن نیز مشخص است ، شبیه‌ساز می‌تواند مقدار عملوندها را تشخیص‌داده و تابع مربوط به آن دستور را فراخوانی‌کند.

کاربر با استفاده از محیط برنامه می‌تواند انتخاب‌کند که دستورات به دنبال هم یا تک‌تک اجراشوند.

۴—۲ چگونگی ورودی و خروجی

فایل ورودی اسمبلر یک فایلِ object است که توسط اسمبلر تولید شده‌است. شبیه‌ساز از روی این فایل خانه‌های حافظه را مقداردهی می‌کند. همچنین محلی از حافظه که اجرای برنامه از آنجا شروع می‌شود در این فایل ذکرشده‌است.

شبیه‌ساز خروجی تولید نمی‌کند بلکه با اجرای هر دستور کاربر با استفاده از محیط برنامه می‌تواند تغییرات صورت‌گرفته در ثبات‌ها و حافظه را مشاهده‌کند.

۴—۳ آشنایی با برنامه شبیه‌ساز

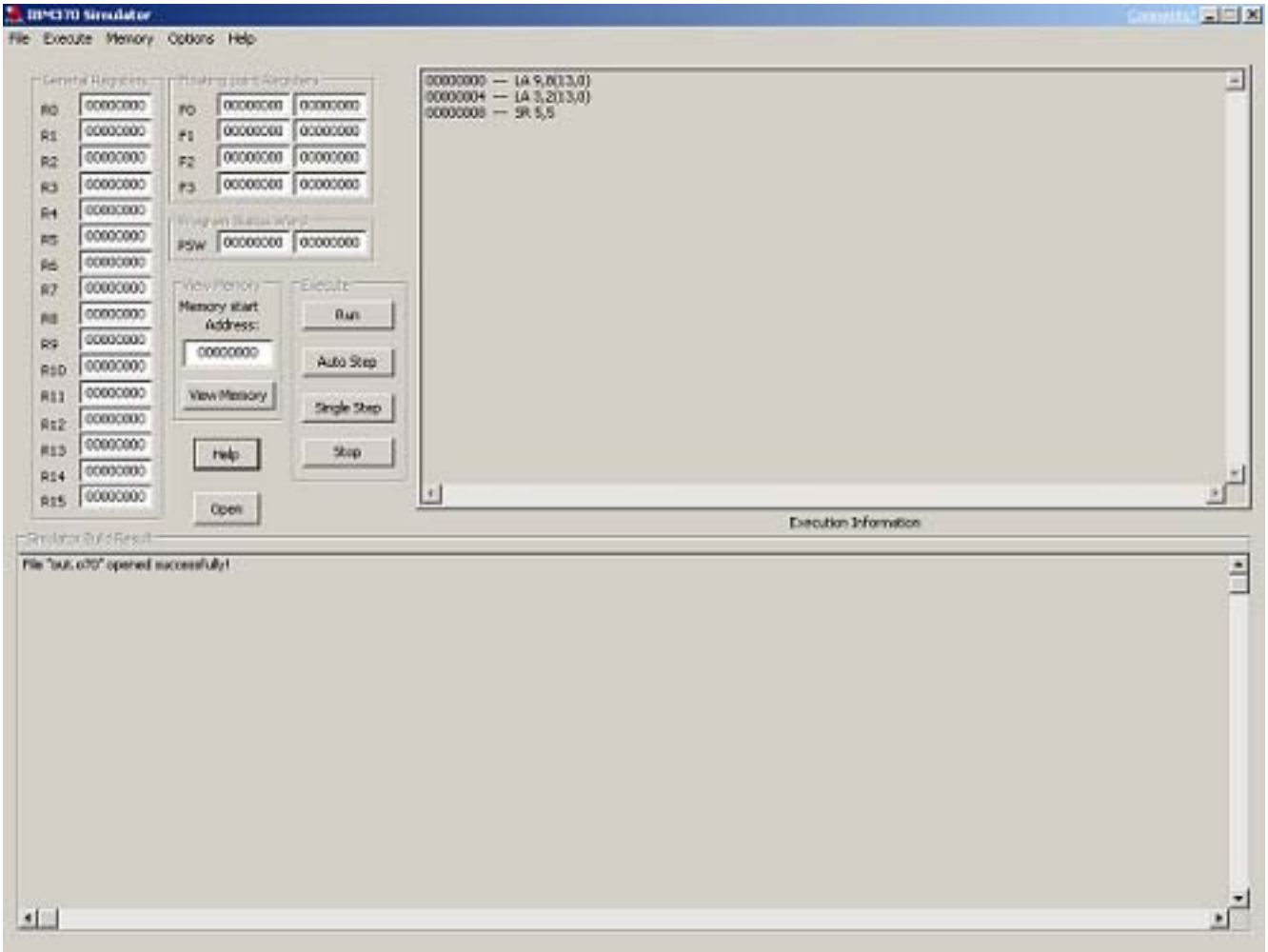
در این بخش خواننده با طرز کار برنامه شبیه‌ساز و امکانات آن آشنا می‌شود. ابتدا به شرح اجزای موجود در دیالوگ برنامه می‌پردازیم.

همانطور که در ۴—۱ مشاهده می‌کنید در دیالوگ شبیه‌ساز در سمت چپ ۱۶ خانه که نمایش‌دهنده ثبات‌های عمومی هستند دیده‌می‌شود. در کنار آن‌ها خانه‌های نمایش دهنده ثبات‌های ممیزشناور و ثبات وضعیت‌برنامه دیده‌می‌شود. در پایین‌تر دو امکان شبیه‌ساز قرارگرفته‌است. در بخش اجرا چهار دکمه وجود دارد که در زیر عملکرد هر کدام را شرح‌می‌دهیم:

---

[1] Single step

[2] Auto step



شکل ۴—۱: محیط برنامه‌ی شبیه‌ساز IBM370

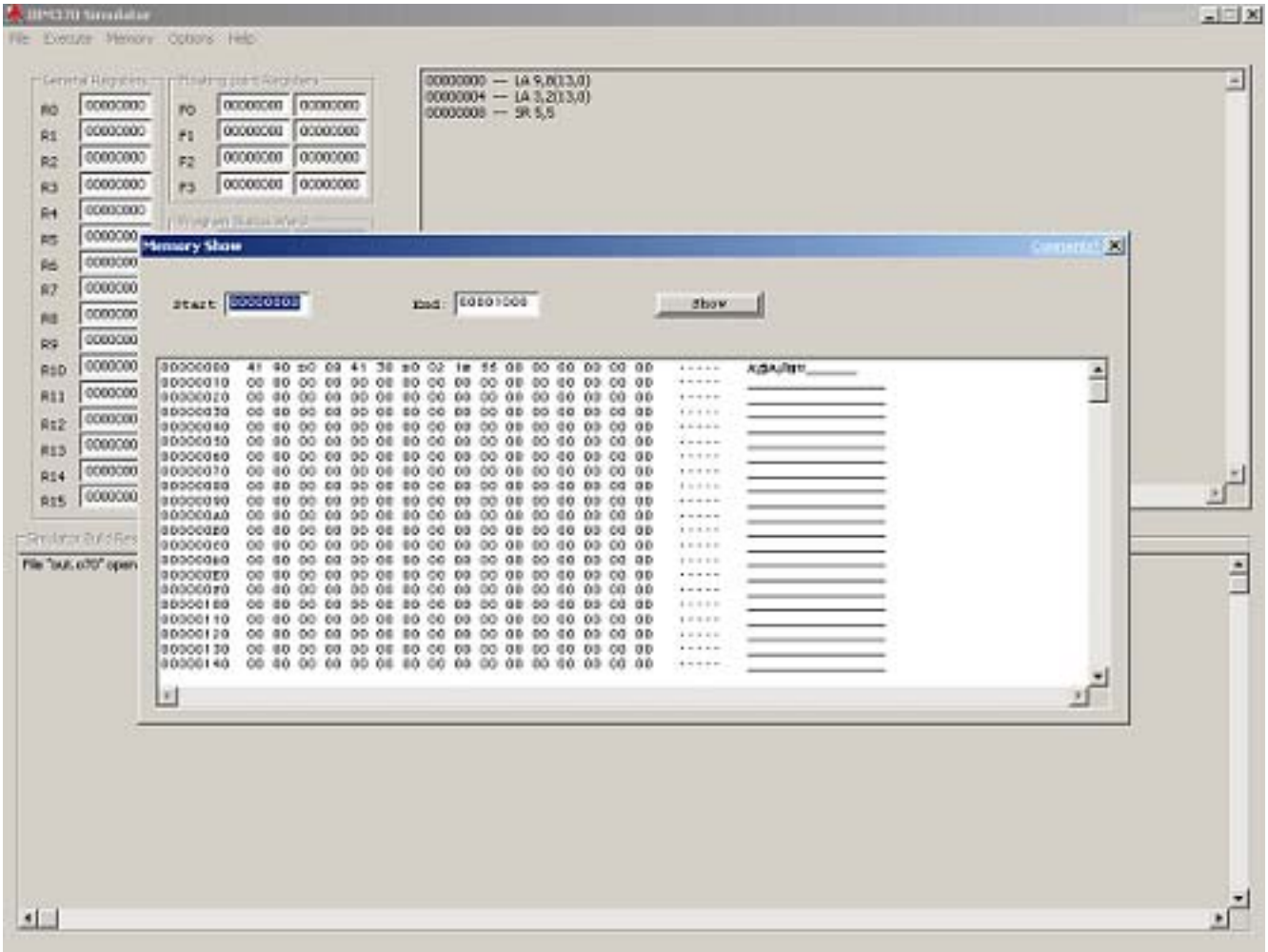
- اجرا بدون توقف(RUN) : با زدن این دکمه برنامه از ابتدا تا انتها اگر اشکالی پیش نیاید بدون توقف اجرامی‌شود.
- اجرای خودکار(Auto Step): با کلیک‌کردن بر روی این دکمه اجرای برنامه به صورت خودکار آغازمی‌شود و هر ۵ ثانیه یک دستور از برنامه اجرا می‌شود. در صورتی که کاربر بخواهد می‌تواند زمان بازه‌ی بین اجرای دو دستورالعمل را تغییردهد. برای عوض کردن بازه‌ی بین اجرای دو دستورالعمل باید از منوی برنامه گزینه‌ی Options و سپس Execution Options انتخاب‌شود.
- اجرای تک‌پله‌ای(Single Step): با کلیک‌کردن بر روی این دکمه یک دستور اجرا می‌شود.
- توقف اجرای خودکار(Stop): در صورتی که کاربر بخواهد در میان اجرای خودکار برنامه اجرای آن را متوقف‌کند بر روی این دکمه کلیک‌می‌کند.

بخش دیگر مربوط به مشاهده خانه‌های حافظه‌می‌باشد. در جعبه‌ی ویرایشی که مشخص‌کننده‌ی آدرس شروع نمایش حافظه می‌باشد کاربر می‌تواند آدرس مورد نظر خود را بنویسد. با کلیک‌کردن بر روی دکمه‌ی show memory پنجره‌ای باز می‌شود که خانه‌های حافظه را نشان می‌دهد. در شکل ۴—۲ مشاهده‌می‌کنید که در دیالوگ جدیدی که باز شده محتوای خانه‌های حافظه نشان‌داده‌شده‌اند.

در صورتی که کاربر بخواهد می‌تواند فایل object دیگری را توسط کلیک‌کردن بر روی دکمه‌ی open بازکرده و اجراکند.

قابل ذکر است که توانایی تغییر دادن هر یک از ثبات‌ها یا خانه‌های حافظه در حین اجرا به کاربر داده‌شده است. برای تغییر محتوای ثبات‌ها که می‌توان مستقیما در جعبه‌ی ویرایش هر کدام مقدارش را عوض‌کرد ولی برای تغییردادن محتوای خانه‌های حافظه از منوی برنامه گزینه‌ی memory و سپس modify memory را انتخاب کنید.

پنجره‌ی سمت راست در صفحه‌ی اصلی شبیه‌ساز برای نمایش آدرس و کد دستورات برنامه‌است. در پنجره‌ی پایین نیز وضعیت فعلی و یا خطاها نشان‌داده‌می‌شوند.



شکل ۴—۲: دیالوگ نمایش محتوای حافظه در برنامه‌ی شبیه‌ساز IBM370

فصل ۵

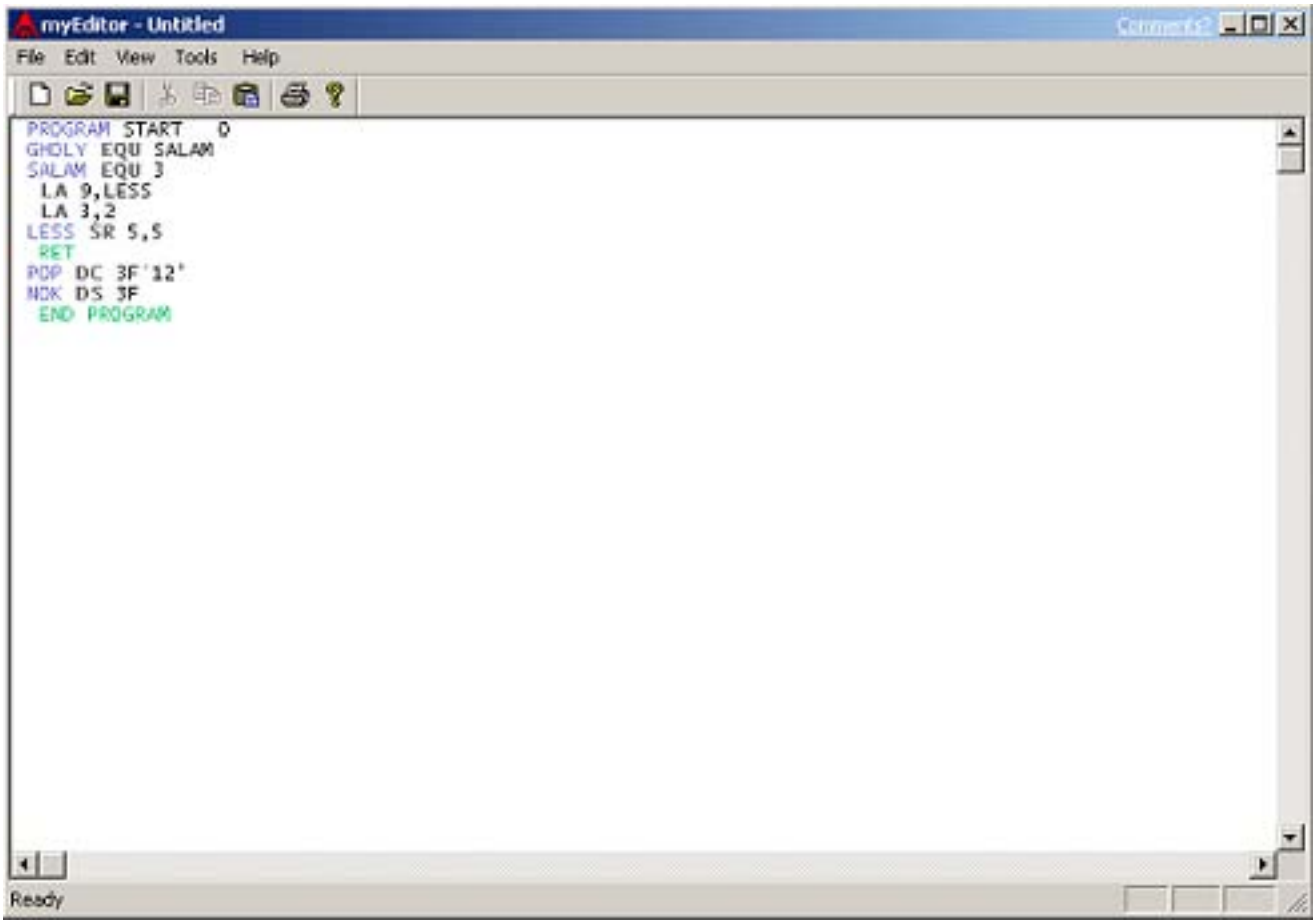
محیط کار با برنامه

با اجراکردن فایل اجرایی اصلی برنامه با نام IBM370.exe در محلی که برنامه نصب‌شده‌است وارد محیط برنامه می شوید. محیط برنامه در شکل ۵—۱ نشان‌داده‌شده‌است.

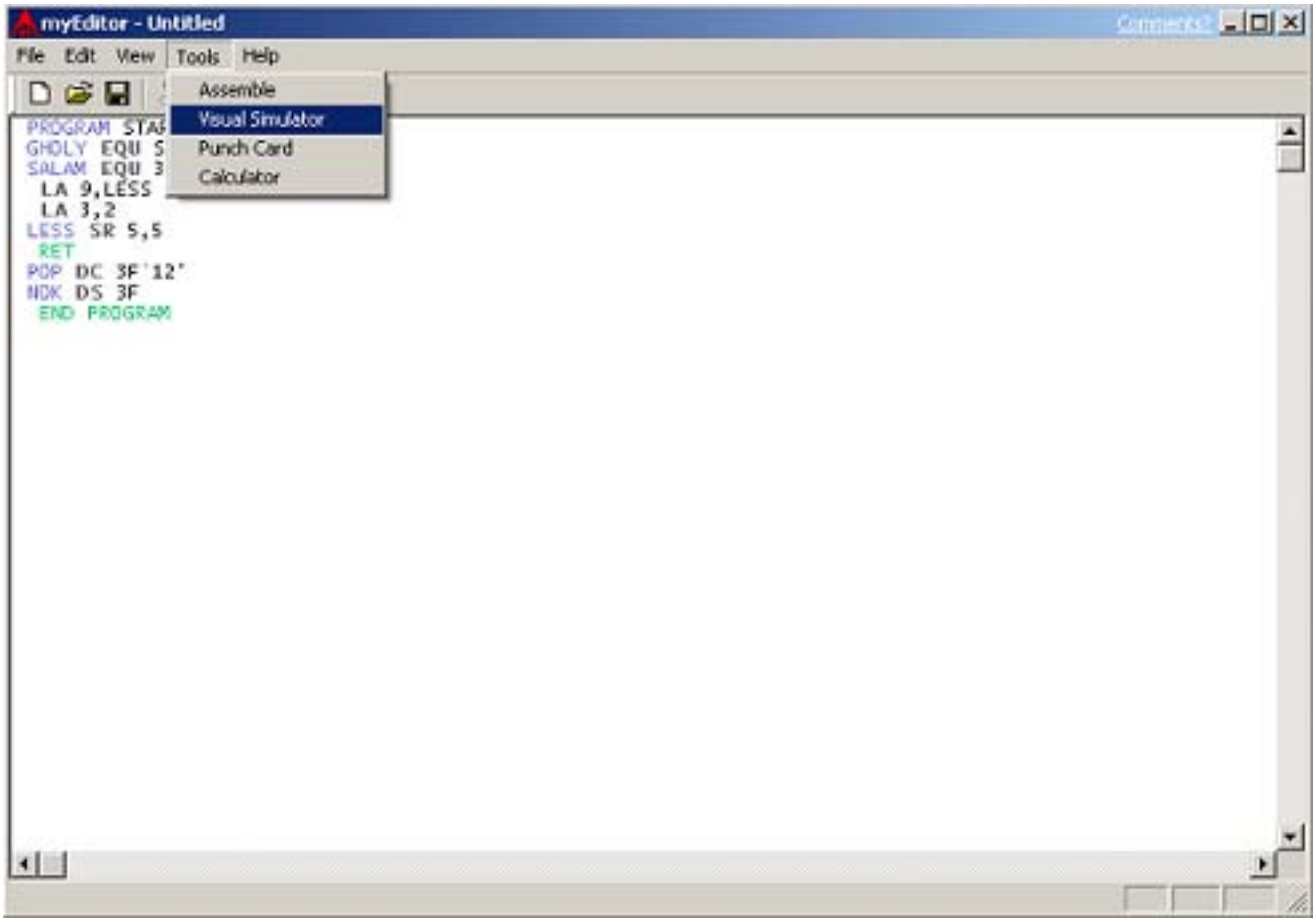
در داخل جعبه‌ی ویرایش که در واقع ویرایشگر برنامه می‌باشد کاربر می‌تواند برنامه‌ی خود را بنویسد و یا یک برنامه‌ی ذخیره‌شده در یک فایل را بازکند. فایل‌های ورودی و خروجی ویرایشگر پسوند 370 دارند. همانطور که در شکل نیز مشاهده‌می‌کنید هر فیلد از برنامه با رنگی متفاوت نشان‌داده‌می‌شود. تغییر رنگ فیلدها را هود برنامه با تشخیص نوع فیلدها انجام می‌دهد. فیلد اسم رنگ آبی ، فیلد عملوند رنگ خاکستری و فیلد توضیحات رنگ سبز دارد. فیلد عملوند با فونت درشت نوشته‌می شود. عملگرها به رنگ سیاه هستند و پرانتزها در آن‌ها به رنگ بنفش نمایش‌داده‌می‌شوند. چنانچه طول یک خط از برنامه از ۷۲ بیشتر باشد ویرایشگر به صورت خودکار ادامه‌ی آن را از ستون ١٦ خط بعد ادامه‌می‌دهد. حداکثر یک دستور برنامه می‌تواند سه خط طول داشته‌باشد ، چنانچه خطی از برنامه طول بیشتر از سه خط داشت ادامه‌ی آن دستوری جدید فرض‌می‌شود.

۵—۱ اسمبلر

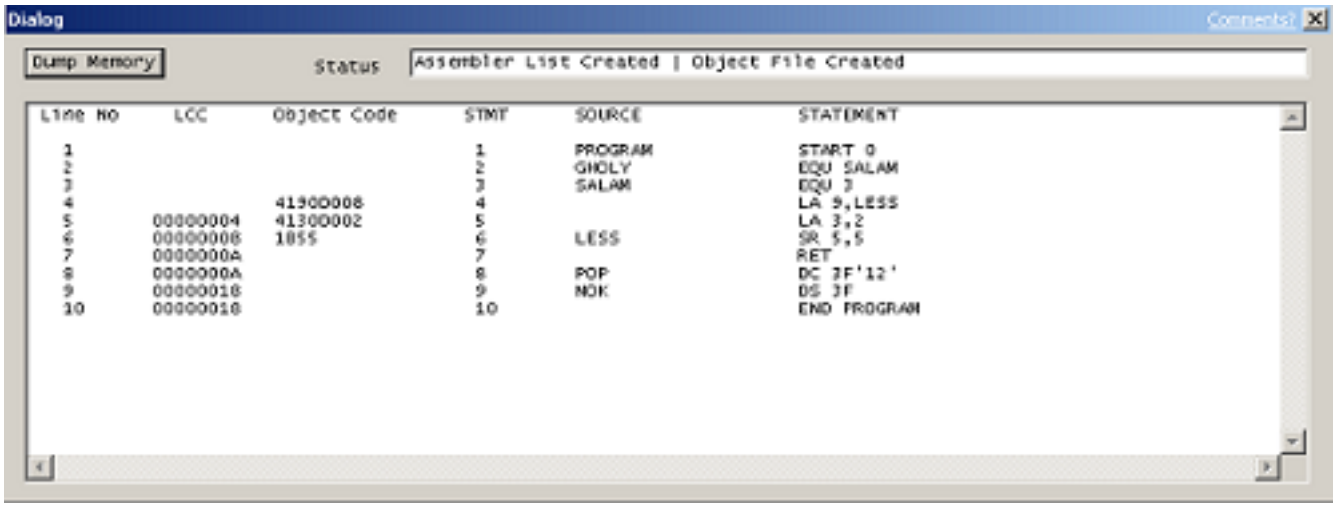
بعد از این که برنامه‌ای توسط کاربر نوشته‌شد حال آماده‌ی اسمبل‌شدن است. همانطور که در شکل ۵—۲ مشاهده می‌کنید در منوی tools چهار گزینه وجوددارد که گزینه‌ی اول assemble است.



شکل ۵—۱: محیط برنامه IBM370



شکل ۵—۲: منوی tools در محیط برنامه‌ی IBM370



شکل ۵—۳: دیالوگ اسمبلر IBM370

با انتخاب این گزینه برنامه توسط اسمبلر IBM370 اسمبل می‌شود و یک دیالوگ جدید باز می‌شود که در شکل ۵—۳ مشاهده‌می‌کنید.

در جعبه‌ی ویرایش status وضعیت فعلی نمایش‌داده‌می‌شود و در جعبه‌ی ویرایش پایین در صورتی که خطایی در برنامه وجودداشته‌باشد نوشته‌می‌شود. چنانچه برنامه بدون مشکل اسمبل‌شود در جعبه‌ی ویرایش پایین لیست اسمبلی برنامه نمایش‌داده‌می‌شود. چنانچه کاربر بخواهد محتوای حافظه را در محلی که برنامه‌ی نوشته‌شده قرارگرفته‌است ببیند بر روی دکمه‌ی Dump کلیک‌می‌کند.

۵—۲ گزینه‌های دیگر منوی tools

چنانچه برنامه‌ی نوشته‌شده توسط اسمبلر با موفقیت اسمبل‌شود یک فایلِ object با پسوندِ o70 ایجادمی‌شود که ورودی شبیه‌ساز است. حال کاربر می‌تواند از منوی tools گزینه‌ی Visual Simulator را انتخاب‌کند تا اجرای برنامه را شبیه‌سازی کند. گزینه‌های دیگر موجود در منوی tools ماشین‌حساب سیستم عاملِ Windows و برنامه‌ی کارت‌پانچ هستند. در پیوست دوم راجع به برنامه‌ی کارت‌پانچ توضیح‌داده‌شده‌است.

فصل ٦

خلاصه و نتیجه‌گیری

در این پایان‌نامه ابتدا نگاهی کوتاه به تاریخچه‌ی ساخت ماشین‌های IBM360/370 به منظور درک اهمیت و تأثیر این ماشین‌ها در پیشرفت تکنولوژی معماری‌کامپیوتر کردیم. به منظور آشنایی خواننده با معماری و مجموعه‌ی دستورهای ماشین‌های IBM360/370 در فصل دوم به معرفی این ماشین‌ها پرداختیم. معرفی ماشین در سه بخش اصلی خصوصیات معماری، آشنایی با ساختار دستورها و آشنایی با قالب زبان اسمبلی ارائه‌شده‌است. در قسمت بعد به اسمبلرنوشته‌شده پرداخته‌شد که به روش دوگذر نوشته‌شده‌است. ابتدا به نحوه‌ی نوشته‌شدن برنامه اشاره‌ای شده، سپس نحوه‌ی ورودی و خروجی و در آخر نمونه‌های خطاها با دلایل بروزشان آورده‌شده‌است. پس از اسمبلر نوبت به محیط‌شبیه‌ساز می‌رسد که در این قسمت سعی کردیم کاربر با محیط‌شبیه‌ساز آشناشود تا بتواند از برنامه‌ی موجود استفاده‌کند. در انتها نیز به محیط کلی برنامه و ویرایشگر پرداخته‌شد.

پیوست ۱ (دستورات ماشین IBM370)

در این پیوست لیست کامل دستورات ماشین ۳۷۰ در سه گروه دستورات عمومی ، دستورات دهدهی و دستورات ممیزشناور عرضه‌شده‌است.

A—۱ دستورات عمومی

دستورات عمومی ماشین ۳۷۰ از قرار زیرند:

Add شامل دو دستور A و AR می‌باشد. عملوند دوم به عملوند اول اضافه می‌شود و حاصل در عملوند اول جایگزین می‌شود.

$AR \quad R_1, R_2$

$A \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل جمع صفر باشد.
- ۱ چنانچه حاصل جمع از صفر کمتر باشد.
- ۲ اگر حاصل جمع از صفر بیشتر باشد.
- ۳ چنانچه سرریز رخ دهد.

Half Add شامل دستور AH می‌باشد که عملوند اول و دوم را جمع می‌کند و حاصل را در عملوند اول می‌ریزد.

$AR \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل جمع صفر باشد.
- ۱ چنانچه حاصل جمع از صفر کمتر باشد.
- ۲ اگر حاصل جمع از صفر بیشتر باشد.
- ۳ چنانچه سرریز رخ دهد

Logical Add شامل دو دستور ALR و AL می‌باشد. به مانند دستورات جمع دیگر دو عملوند را جمع می‌کند و حاصل را در عملوند اول می‌ریزد. تفاوت این جمع با جمع‌های حسابی در کد وضعیت حاصل است.

$ALR \quad R_1, R_2$

$AL \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل جمع صفر باشد و رقم نَقلی نداشته‌باشیم.
- ۱ چنانچه حاصل جمع صفر نباشد و رقم نَقلی نداشته‌باشیم.
- ۲ اگر حاصل جمع صفر باشد همراه با رقم نَقلی.
- ۳ در صورتی که حاصل جمع صفر نباشد همراه با رقم نَقلی.

AND شامل دستورات NR ، N ، NI و NC می‌باشد. دو عملوند اول و دوم را AND می‌کند و حاصل را در عملوند اول جایگزین می‌کند.

$NR \quad R_1, R_2$

$N \quad R_1, D_2(X_2, B_2)$

$NI \quad D_1(B_1), I_2$

$NC \quad D_1(L, B_1), D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ چنانچه حاصل صفر نباشد.

Branch and Link شامل دستورات BALR و BAL می‌باشد. اطلاعات PSW در حال حاضر به عملوند اول ریخته‌می‌شود.آدرس PSW ، آدرس موجود در عملوند دوم می‌شود و از آن برای انشعاب استفاده‌می‌شود.

$BALR \quad R_1, R_2$

$BAL \quad R_1, D_2(X_2, B_2)$

کد وضعیت تغییر نمی‌کند.

Branch on Condition شامل دو دستور BCR و BC است.

$BCR \quad M_1, R_2$

$BC \quad M_1, D_2(X_2, B_2)$

کد وضعیت تغییر نمی‌کند.

Branch on Count شامل دو دستور BCTR و BCT می‌باشد. مقدار فعلی R_1 منهای یک می‌شود ، اگر مقدار حاصل صفر باشد اجرای ترتیبی دستورها صورت می‌گیرد در غیر این صورت آدرس PSW با آدرس انشعاب جایگزین می‌شود.

$BCTR \quad R_1, R_2$

$BCTL \quad R_1, D_2(X_2, B_2)$

کد وضعیت تغییر نمی‌کند.

Branch on Index High شامل دستور BXH می‌باشد. عملوند اول را به علاوه‌ی یک می‌کند. چه انشعاب صورت بگیرد یا نه در هر حال عملوند اول را با یک جمع‌می‌کند. عملوند دوم به عنوان آدرس انشعاب استفاده‌می‌شود. چنانچه مقدار جدید عملوند اول از مقدار عملوند سوم بیشتر باشد انشعاب صورت‌می‌گیرد و در صورت کوچکتر یا مساوی بودن اجرای طبیعی دستورات به دنبال هم ادامه‌می‌یابد.

$BXH \quad R_1, R_3, D_2(B_2)$

کد وضعیت تغییر نمی‌کند.

Branch on Index Low or Equal شامل دستور BXLE می‌باشد. مانند دستور BXH با این تفاوت که موقعی که مقدار جدید عملوند اول از عملوند سوم بیشتر باشد دستورات به صورت ترتیبی اجرا می‌شوند و در غیر این صورت انشعاب صورت می‌گیرد

$BXLE \quad R_1, R_3, D_2(B_2)$

کد وضعیت تغییر نمی‌کند.

Compare شامل دستورهای CR و C می‌باشد. عملوندهای اول و دوم را با هم مقایسه می‌کند و بر اساس آن کد وضعیت را تعیین می‌کند.

$CR \quad R_1, R_2$

$C \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه عملوندها مساوی باشند.
- ۱ چنانچه عملوند اول کوچکتر باشد.
- ۲ اگر عملوند اول بیشتر باشد.

Compare and Swap شامل دستور CS است. عملوند اول و دوم با هم مقایسه می‌شوند چنانچه مساوی باشند ، عملوند سوم در محل عملوند دوم ذخیره می‌شود. چنانچه مساوی نباشند عملوند دوم در محل عملوند اول ذخیره می‌شود.

$CS \quad R_1, R_3, D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه عملوندها مساوی باشند.
- ۱ چنانچه عملوندها مساوی نباشند.

Compare Double ans Swap شامل دستور CDS می‌باشد. مانند دستور CS می‌باشد با این تفاوت که در اینجا عملوندها ٦۴ بیتی هستند.

$CDS \quad R_1, R_3, D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه عملوندها مساوی باشند.
- ۱ چنانچه عملوندها مساوی نباشند.

Compare halfword شامل دستور CH می‌باشد. عملوند ١٦ بیتی اول را با عملوند ١٦ بیتی دوم مقایسه می‌کند و بر اساس آن کد وضعیت را تعیین می‌کند.

$CH \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه عملوندها مساوی باشند.
- ۱ چنانچه عملوند اول کوچکتر باشد.
- ۲ اگر عملوند اول بیشتر باشد.

Compare Logical شامل دستورات CLR ، CL ، CLI و CLC می‌باشد. دو عملوند اول و دوم را مقایسه می‌کند و بر اساس آن‌ها کد وضعیت را تعیین می‌کند. عملوندها را به صورت بدون علامت درنظر می‌گیرد.

$CLR \quad R_1, R_2$

$CL \quad R_1, D_2(X_2, B_2)$

$CLI \quad D_1(B_1), I_2$

$CLC \quad D_1(L, B_1), D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه عملوندها مساوی باشند.
- ۱ چنانچه عملوند اول کوچکتر باشد.
- ۲ اگر عملوند اول بیشتر باشد.

Compare Logical Character Under Mask شامل دستور CLM است. عملوندهای اول و دوم تحت کنترل ماسک با هم مقایسه‌می‌شوند.

$CLM \quad R_1, M_3, D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه بایت‌های انتخاب‌شده مساوی باشند و یا ماسک صفر باشد.
- ۱ چنانچه بایت‌های انتخاب‌شده عملوند اول کوچکتر باشد.
- ۲ اگر بایت‌های انتخاب‌شده عملوند اول بزرگتر باشد.

Compare Logical Long شامل دستور CLCL است. عملوند اول با عملوند دوم مقایسه‌می‌شود و نتیجه درکد وضعیت نشان داده می‌شود. عملوند کوتاهتر گسترش‌یافته به وسیله کاراکتر padding در نظر گرفته‌می‌شود. ثبات‌های R_1 و R_2 هر کدام نمایانگر یک زوج ثبات هستند و باید عدد زوجی باشند ، در غیر این صورت حالت استثنا رخ می‌دهد.

$CLCL \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه عملوندها مساوی باشند و یا هر دو فیلد دارای طول صفر باشند.
- ۱ چنانچه عملوند اول کوچکتر باشد.
- ۲ اگر عملوند اول بزرگتر باشد.

Convert to Binary شامل دستور CVB است. مبنای عملوند دوم از ۱۰ به ۲ تغییر داده می‌شود و حاصل در عملوند اول جایگزین می‌شود. عدد چه قبل و چه بعد از تغییر به صورت یک integer علامت‌دار که از right-aligned در نظر گرفته می‌شود.

$CVB \quad R_1, D_2(X_2, B_2)$

کد حالت تغییر نمی‌کند.

Divide شامل دو دستور DR و D می‌باشد. عملوند اول (مقسوم) تقسیم بر عملوند دوم (مقسوم‌علیه) می‌شود و عملوند اول با خارج‌قسمت و باقیمانده جایگزین می‌شود. عملوند اول ۶۴ بیت و عملوند دوم ۳۲ بیت می‌باشد. در ۳۲ بیت سمت چپ عملوند اول باقیمانده و در ۳۲ بیت راست آن خارج‌قسمت قرارمی‌گیرد.

$DR \quad R_1, R_2$

$D \quad R_1, D_2(X_2, B_2)$

کد حالت تغییر نمی‌کند.

Exclusive OR شامل دستورات XR ، X ، XI و XC می‌باشد. عملوند اول و دوم را XOR می‌کند و حاصل را در عملوند اول جایگزین می‌کند.

$XR \quad R_1, R_2$

$X \quad R_1, D_2(X_2, B_2)$

$XI \quad D_1(B_1), I_2$

$XC \quad D_1(L_1, B_1), D_2(L_1, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر شود.
- ۱ چنانچه حاصل صفر نشود.

Insert Character شامل دستور IC است. بایت موجود در محل علوند دوم در بیتهای ۲۴ تا ۳۱ عملوند اول درج می‌شود.

$IC \quad R_1, D_2(X_2, B_2)$

کد حالت تغییر نمی‌کند.

Insert Character Under Mask شامل دستور ICM است. بایت‌های تعیین شده توسط ماسک از محل عملوند دوم به دنبال هم در عملوند اول درج می‌شوند.

$ICM \quad R_1, M_3, D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه تمام بیت‌های درج‌شده صفر باشند یا ماسک صفر باشد.

- ۱ اگر بیت اول فیلد درج‌شده یک باشد.
- ۲ در صورتی که بیت اول فیلد درج‌شده صفر باشد و همه بیت‌های دیگر آن صفر نباشند.

Load شامل دو دستور LR و L می‌باشد. عملوند دوم بدون تغییر جایگزین عملوند اول می‌شود.

$LR \quad R_1, R_2$

$L \quad R_1, D_2(X_2, B_2)$

کد حالت تغییر نمی‌کند.

Load Address شامل دستور LA است. آدرس مشخص‌شده توسط D_2 ، X_2 و B_2 جایگزین عملوند اول می‌شود.

$LA \quad R_1, D_2(X_2, B_2)$

کد حالت تغییر نمی‌کند.

Load and Test شامل دستور LTR است. عملوند دوم بدون تغییر جایگزین عملوند اول می‌شود و از مقدار و علامت آن کد وضعیت تعیین می‌شود.

$LTR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل کوچکتر از ثفر باشد.
- ۲ در صورتی که حاصل بیشتر از صفر باشد.

Load Complement شامل دستور LCR می‌باشد. مکمل ۲ عملوند دوم را جایگزین عملوند اول می‌کند.

$LCR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.

- ۱ اگر حاصل کوچکتر از ثفر باشد.
- ۲ در صورتی که حاصل بیشتر از صفر باشد.
- ۳ چنانچه سرریز رخ‌دهد.

Load Halfword شامل دستور LH است. عملوند ۱۶ بیتی دوم در ۱۶ بیت سمت راست عملوند اول قرارداده‌می‌شود و علامت آن تا ۳۲ بیت سمت چپ گسترش داده‌می شود.

$LH \quad R_1, D_2(X_2, B_2)$

کد حالت تغییر نمی‌کند.

Load Multiple شامل دستور LM می‌باشد. ثبات‌های بین دو ثبات مشخص‌شده توسط عملوندهای اول و سوم توسط محل مشخص‌شده توسط عملوند دوم مقداردهی می‌شوند.

$LM \quad R_1, R_3, D_2(B_2)$

کد حالت تغییر نمی‌کند.

Load Negative شامل دستور LNR است. منفی عملوند دوم به صورت مکمل ۲ جایگزین عملوند اول می‌شود.

$LNR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل کوچکتر از صفر باشد.

Load Positive شامل دستور LPR است. قدر مطلق عملوند دوم جایگزین عملوند اول می‌شود.

$LPR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۲ اگر حاصل بزرگتر از صفر باشد.

- ۳ در صورتی که سرریز رخ‌دهد.

Move شامل دستورات MVI و MVC می‌باشد. عملوند دوم در محل مشخص‌شده توسط عملوند اول جایگزین می‌شود.

$MVI \quad D_1(B_1), I_2$

$L \quad D_1(L, B_1), D_2(B_2)$

کد حالت تغییر نمی‌کند.

Move Long شامل دستور MVCL است. عملوند ٦۴ بیتی دوم جایگزین عملوند ٦۴ بیتی اول می‌شود.

$MVCL \quad R_1, R_2$

کد وضعیت حاصل:

Move Numeric شامل دستور MVN است. ۴ بیت کم‌ارزش هر بایت در عملوند دوم جایگزین ۴بیت کم‌ارزش بایت متقارن آن از عملوند اول می‌شود. ۴ بیت باارزش عملوند اول بدون تغییر می‌ماند.

$MVN \quad D_1(L, B_1), D_2(B_2)$

کد وضعیت بدون تغییر می‌ماند.

Move With offset شامل دستور MVO می‌باشد.

$MVO \quad D_1(L_1, B_1), D_2(L_2, B_2)$

کد حالت را تغییر نمی‌دهد.

Move Zones شامل دستور MVZ است. ۴ بیت پرارزش هر بایت در عملوند دوم جایگزین ۴بیت پرارزش بایت متقارن آن از عملوند اول می‌شود. ۴ بیت کم‌ارزش عملوند اول بدون تغییر می‌ماند.

$MVZ \quad D_1(L, B_1), D_2(B_2)$

کد حالت تغییر نمی‌کند.

Multiply شامل دو دستور MR و M است. حاصلضرب عملوند اول و دوم در عملوند اول ریخته‌می‌شود.

$MR \quad R_1, R_2$

$M \quad R_1, D_2(X_2, B_2)$

کد حالت عوض نمی‌شود.

Multiply Halfword شامل دستور MH است. حاصلضرب عملوند اول و دوم جایگزین عملوند اول می‌شود. طول عملوند دوم ۱۶ بایت است و به صورت یک integer ِعلامت‌دار در نظر گرفته‌می‌شود.

$MH \quad R_1, D_2(X_2, B_2)$

کد حالت عوض نمی‌شود.

OR شامل دستورات OR ، O ، OI و OC است. عملوند اول و دوم را OR می‌کند و حاصل را در عملوند اول می‌ریزد.

$OR \quad R_1, R_2$

$O \quad R_1, D_2(X_2, B_2)$

$OI \quad D_1(B_1), I_2$

$OC \quad D_1(L, B_1), D_2(B_2)$

کد حالت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل صفر نباشد.

Pack شامل دستور PACK می‌باشد. فرمت عملوند دوم از فرمت دهدهی ناحیه‌ای[1] به فرمت دهدهی فشرده[2] تبدیل می‌شود و جایگزین عملوند اول می‌شود.

$PACK \quad D_1(L_1, B_1), D_2(L_2, B_2)$

کد وضعیت تغییر نمی‌کند.

---

[1] Zoned Decimal

[2] Packed Decimal

Set Program Mask شامل دستور SPM می‌باشد. بیت‌های ۲ تا ۷ از عملوند اول جایگزین کد وضعیت و ماسک در PSW می‌شوند. بیت‌های ۱۲ تا ۱۵ دستور فراموش می‌شوند.

$SPM \quad R_1$

کد وضعیت با بیت‌های ۲ و ۳ از عملوند R_1 جایگزین می‌شود.

Shift left Double شامل دستور SLDA است. عملوند اول ٦۴ بیتی است. عملوند اول به صورت یک عدد ٦۳ بیتی علامت‌دار در نظر گرفته می‌شود. عملوند اول به تعداد مقدار عملوند دوم به چپ شیفت داده می‌شود بدون اینکه علامتش عوض‌شود.

$SLDA \quad R_1, D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل کمتر از صفر باشد.
- ۲ چنانچه حاصل بیشتر از صفر باشد.
- ۳ در صورتی که سرریز رخ‌دهد.

Shift Left Double logical شامل دستور LSDL می‌باشد. عملوند ٦۴ بیتی اول به تعداد مقدار عملوند دوم به چپ شیفت داده می‌شود. هر ٦۴ بیت عملوند اول در شیفت شرکت می‌کنند.

$SLDL \quad R_1, D_2(B_2)$

کد وضعیت تغییر نمی‌کند.

Shift Left Single شامل دستور SLA است. مانند دستور SLDA می‌باشد با این تفاوت که در این دستور طول عملوند اول ۳۲ بیت است.

$SLA \quad R_1, D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل کمتر از صفر باشد.

- ۲ چنانچه حاصل بیشتر از صفر باشد.
- ۳ در صورتی که سرریز رخ‌دهد.

Shift Left Single Logical شامل دستور SLL است. مانند دستور SLDL با این تفاوت که در این دستور طول عملوند اول ۳۲ بیت است.

$SLL \quad R_1, D_2(B_2)$

کد وضعیت تغییر نمی‌کند.

Shift Right Double شامل دستور SRDA می‌باشد. عدد ٦٣ بیتی علامت‌دار عملوند اول به تعداد مقدار عملوند دوم به راست شیفت داده می‌شود. علامت عملوند اول عوض نمی‌شود.

$SRDA \quad R_1, D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل کمتر از صفر باشد.
- ۲ چنانچه حاصل بیشتر از صفر باشد.

Shift Right Double Logical شامل دستور SRDL است. ٦۴ بیت عملوند اول به تعداد مقدار عملوند دوم به راست شیفت داده می‌شوند.

$SRDL \quad R_1, D_2(B_2)$

کد حالت عوض نمی‌شود.

Shift Right Single شامل دستور SRA است. مانند دستور SRDA است با این تفاوت که عملوند اول در این دستور ۳۲ بیت است.

$SRA \quad R_1, D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل کمتر از صفر باشد.

- ۲ چنانچه حاصل بیشتر از صفر باشد.

Shift Right Single Logical شامل دستور SRL است. مانند دستور SRDL است با این تفاوت که طول عملوند اول در این دستور ۳۲ بیت است.

$SRL \quad R_1, D_2(B_2)$

کد وضعیت تغییر نمی‌کند.

Store شامل دستور ST است. عملوند اول در محل عملوند دوم ذخیره می‌شود.

$ST \quad R_1, D_2(X_2, B_2)$

کد وضعیت تغییر نمی‌کند.

Store Character شامل دستور STC می‌باشد. بیت‌های ۲۴ تا ۳۱ از عملوند اول در محل عملوند دوم ذخیره می‌شود.

$STC \quad R_1, D_2(X_2, B_2)$

کد وضعیت تغییر نمی‌کند.

Store Character Under Mask شامل دستور STCM است. بایت‌های انتخاب‌شده تحت کنترل ماسک در بایت‌های پشت سر هم در محلی که عملوند دوم به آن اشاره می‌کند جایگزین می‌شوند.

$STCM \quad R_1, M_3, D_2(B_2)$

کد وضعیت تغییر نمی‌کند.

Store Clock شامل دستور STCK است. زمان فعلی «ساعت زمان روز[۳]» در فیلد ۸ بایتی مشخص‌شده توسط عملوند دوم ذخیره می‌شود.

$STCK \quad D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه ساعت در حالت تنظیم[۴] باشد.

---

[۳] Time-of-day clock

[۴] Set state

- ۱ اگر ساعت در حالت تنظیم نشده[۵] باشد.
- ۲ چنانچه ساعت در حالت خطا[۶] باشد.
- ۳ در صورتی که ساعت در حالت ایست[۷] و یا غیرکاری[۸] باشد.

Store Hlafword شامل دستور STH است. بیت‌های ۱۶ تا ۳۱ از عملوند اول در محل عملوند دوم ذخیره می‌شود.

$STH \quad R_1, D_2(X_2, B_2)$

کد حالت تغییر نمی‌کند.

Store Multiple شامل دستور STM می‌باشد. محتوای ثبات‌های از فیلد R_1 تا R_3 در محل مشخص‌شده توسط عملوند دوم ذخیره می‌شود.

$STM \quad R_1, R_3, D_2(B_2)$

کد حالت تغییر نمی‌کند.

Subtract شامل دو دستور SR و S است. عملوند دوم از عملوند اول کم می‌شود و حاصل در عملوند اول جایگزین می‌شود.

$SR \quad R_1, R_2$

$S \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل تفریق صفر باشد.
- ۱ اگر حاصل از صفر کمتر باشد.
- ۲ چنانچه حاصل از صفر بیشتر باشد.
- ۳ در صورتی که سرریز رخ‌دهد.

---

[۵] Not-set state
[۶] Error state
[۷] Stopped state
[۸] Not-operational state

Subtract Halfword شامل دستور SH است. عملوند دوم از عملوند اول کم می‌شود و حاصل در عملوند اول نوشته‌می‌شود. عملوند دوم ۱۶ بیت است که به صورت یک عدد ۳۲ بیتی علامت‌دار در نظر گرفته‌می‌شود.

$SH \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل تفریق صفر باشد.
- ۱ اگر حاصل از صفر کمتر باشد.
- ۲ چنانچه حاصل از صفر بیشتر باشد.
- ۳ در صورتی که سرریز رخ‌دهد.

Subtract Logical شامل دستورهای SLR و SL می‌باشد. تفریق را انجام می‌دهد و کد وضعیت را بر اساس رقم نَقلی تعیین می‌کند.

$SLR \quad R_1, R_2$

$SL \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- ۱ چنانچه حاصل تفریق صفر نیست و رقم نَقلی نداریم.
- ۲ اگر حاصل صفر است و رقم نَقلی داریم.
- ۳ چنانچه حاصل صفر نیست و رقم نَقلی داریم.

Translate شامل دستور TR است. بایت‌های عملوند اول برای ارجاع‌دادن به لیست مشخص‌شده توسط عملوند دوم استفاده می‌شوند. هر بایت انتخاب‌شده از لیست جایگزین بایت مورد نظر می‌شود.

$TR \quad D_1(L, B_1), D_2(B_2)$

کد وضعیت تغییر نمی‌کند.

Translate and Test شامل دستور TRT است. بایت‌های عملوند اول برای ارجاع‌دادن به لیست مشخص‌شده توسط عملوند دوم استفاده می‌شوند. هر بار بایت انتخاب‌شده از لیست تعیین می‌کند که آیا عمل ترجمه ادامه می‌یابد یا خیر. چنانچه بایت انتخاب‌شده از لیست صفر باشد عملیات با خواندن و ترجمه بایت بعدی ادامه می‌یابد. چنانچه بایت ترجمه صفر نباشد عملیات با درج بایتی که در حال ترجمه بود به ثبات R_1 و درج بایت ترجمه به ثبات R_2 کامل می‌شود.

$TRT \quad D_1(L, B_1), D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه همه بایت‌های ترجمه صفر باشند.
- ۱ اگر پیش از به پایان رسیدن عملوند اول به بایت ترجمه غیر صفر برخورد کنیم.
- ۲ در صورتی که آخرین بایت ترجمه غیر صفر باشد.

Unpack شامل دستور UNPK است. فرمت عملوند دوم را از دهدهی فشرده‌شده به دهدهی ناحیه‌ای تبدیل می‌کند.

$UNPK \quad D_1(L_1, B_1), D_2(L_2, B_2)$

کد حالت تغییر نمی‌کند.

A—۲ دستورات دهدهی

دستورات دهدهی اعمال حسابی ، منطقی ، شیفت و ویرایش را روی داده‌های دهدهی انجام‌می‌دهند. دستورات دهدهی ماشین ۳۷۰ عبارتند از:

Add Decimal شامل دستور AP است. عملوند اول با عملوند دوم جمع می‌شود و حاصل در عملوند اول جایگزین می‌شود.

$AP \quad D_1(L_1, B_1), D_2(L_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل از صفر کمتر باشد.
- ۲ اگر حاصل از صفر بیشتر باشد.
- ۳ در صورتی که سرریز رخ‌دهد.

Compare Decimal شامل دستور CP است. عملوند اول با عملوند دوم مقایسه می‌شود و بسته به نتیجه کد حالت تعیین می‌شود.

$CP \quad D_1(L_1, B_1), D_2(L_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه عملوندها مساوی باشند.
- ۱ اگر عملوند اول کوچکتر باشد.
- ۲ اگر عملوند اول بزرگتر باشد.

Divide Decimal شامل دستور DP است. عملوند اول به عملوند دوم تقسیم می‌شود و با خارج قسمت و باقیمانده جایگزین می‌شود.

$DP \quad D_1(L_1, B_1), D_2(L_2, B_2)$

کد حالت تغییر نمی‌کند.

Edit شامل دستور ED است. فرمت عملوند دوم که دهدهی فشرده‌شده می‌باشد به دهدهی ناحیه‌ای تبدیل می‌شود سپس تحت کنترل عملوند اول ویرایش می‌شود. حاصل ویرایش شده جایگزین عملوند اول می‌شود. برای اطلاعات کامل به منبع [۳] مراجعه کنید.

$ED \qquad D_1(L, B_1), D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه فیلد آخر صفر باشد.
- ۱ اگر فیلد آخر کوچکتر از صفر باشد.
- ۲ اگر فیلد آخر بزرگتر از صفر باشد.

Edit and Mark شامل دستور EDMK می‌باشد. مانند دستور ED می‌باشد علاوه بر اینکه آدرس هر نتیجه ارزش‌اول[۹] در ثبات R_1 ذخیره می‌شود.

$EDMK \qquad D_1(L, B_1), D_2(B_2)$

کد وضعیت حاصل:

- صفر چنانچه فیلد آخر صفر باشد.
- ۱ اگر فیلد آخر کوچکتر از صفر باشد.
- ۲ اگر فیلد آخر بزرگتر از صفر باشد.

Multiply Decimal شامل دستور MP است. حاصلضرب عملوند اول و دوم جایگزین عملوند اول می‌شود.

$MP \qquad D_1(L_1, B_1), D_2(L_2, B_2)$

کد حالت تغییر نمی‌کند.

Shift and Round Decimal شامل دستور SRP است. عملوند اول به تعداد قدر مطلق عملوند دوم به راست یا چپ شیفت داده می‌شود. مقدار منفی برای عملوند دوم نشانگر شیفت به راست و مقدار مثبت نشان‌دهنده شیفت به چپ می‌باشد. عملوند دوم برای آدرس‌دهی استفاده نمی‌شود بلکه از ۶ بایت سمت راست آن به عنوان مقدار شیفت استفاده می‌شود.

[۹] First Significant result

$SRP \qquad D_1(L_1, B_1), D_2(B_2), I_3$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل از صفر کمتر باشد.
- ۲ اگر حاصل از صفر بیشتر باشد.
- ۳ در صورتی که سرریز رخ‌دهد.

Subtract Decimal شامل دستور SP است. عملوند دوم از عملوند اول کم می‌شود و حاصل در عملوند اول جایگزین می‌شود.

$SP \qquad D_1(L_1, B_1), D_2(L_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل از صفر کمتر باشد.
- ۲ اگر حاصل از صفر بیشتر باشد.
- ۳ در صورتی که سرریز رخ‌دهد.

Zero and Add شامل دستور ZAP می‌باشد. عملوند دوم جایگزین عملوند اول می‌شود. عملکرد این دستور همانند جمع با صفر می‌باشد.

$ZAP \qquad D_1(L_1, B_1), D_2(L_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه حاصل صفر باشد.
- ۱ اگر حاصل از صفر کمتر باشد.
- ۲ اگر حاصل از صفر بیشتر باشد.
- ۳ در صورتی که سرریز رخ‌دهد.

A—۳ دستورات ممیزشناور

دستورات ممیزشناور برای محاسبات اعشاری و همچنین محاسبات در دامنه‌ی بزرگی از مقدار استفاده‌می‌شوند. دستورات ممیز شناور ماشین ۳۷۰ عبارتند از:

Add Normalized شامل دستورات AER ، AE ، ADR ، AD و AXR می‌باشد. عملوند اول را با عملوند دوم جمع می‌کند و حاصل نرمال‌شده را در عملوند اول جایگزین می‌کند.

$AER \quad R_1, R_2$

$AE \quad R_1, D_2(X_2, B_2)$

$ADR \quad R_1, R_2$

$AD \quad R_1, D_2(X_2, B_2)$

$AXR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه قسمت غیر نمایی نتیجه صفر باشد.
- ۱ اگر حاصل از صفر کمتر باشد.
- ۲ اگر حاصل از صفر بیشتر باشد.

Add Unnormalized شامل دستورات AUR ، AU ، AWR و AW می‌باشد. عملوند اول را با عملوند دوم جمع می‌کند و حاصل نرمال‌نشده را در عملوند اول جایگزین می‌کند.

$AUR \quad R_1, R_2$

$AU \quad R_1, D_2(X_2, B_2)$

$AWR \quad R_1, R_2$

$AW \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه قسمت غیر نمایی نتیجه صفر باشد.
- ۱ اگر حاصل از صفر کمتر باشد.
- ۲ اگر حاصل از صفر بیشتر باشد.

Compare شامل دستورات CER ، CE ، CDR و CD است. عملوند اول و دوم را مقایسه می‌کند و بر اساس آن کد وضعیت را تعیین می‌کند.

$CER \quad R_1, R_2$

$CE \quad R_1, D_2(X_2, B_2)$

$CDR \quad R_1, R_2$

$CD \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه عملوندها مساوی باشند.
- ۱ اگر عملوند اول کوچکتر باشد.
- ۲ اگر عملوند اول بزرگتر باشد.

Divide شامل دستورات DER ، DE ، DDR و DD است. عملوند اول تقسیم بر عملوند دوم می‌شود و با خارج‌قسمت و باقیمانده جایگزین می‌شود.

$DER \quad R_1, R_2$

$DE \quad R_1, D_2(X_2, B_2)$

$DDR \quad R_1, R_2$

$DD \quad R_1, D_2(X_2, B_2)$

کد وضعیت بدون تغییر می‌ماند.

Load شامل دستورات LER ، LE ، LDR و LD است. عملوند دوم بدون تغییر در محل عملوند اول جایگزین می‌شود.

$LER \quad R_1, R_2$

$LE \quad R_1, D_2(X_2, B_2)$

$LDR \quad R_1, R_2$

$LD \quad R_1, D_2(X_2, B_2)$

کد وضعیت بدون تغییر می‌ماند.

Load and Test شامل دستورات LTER ، LTDR است. عملوند دوم بدون تغییر در محل عملوند اول جایگزین می‌شود و از علامت و مقدار آن برای تعیین کد وضعیت استفاده می‌شود.

$LTER \quad R_1, R_2$

$LTDR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه قسمت غیر نمایی نتیجه صفر باشد.
- ۱ اگر نتیجه کوچکتر از صفر باشد.
- ۲ اگر نتیجه بزرگتر از صفر باشد.

Load Complementg شامل دستورهای LCER و LCDR است. منفی عملوند دوم جایگزین عملوند اول می‌شود.

$LCER \quad R_1, R_2$

$LCDR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه قسمت غیر نمایی نتیجه صفر باشد.
- ۱ اگر نتیجه کوچکتر از صفر باشد.
- ۲ اگر نتیجه بزرگتر از صفر باشد.

Load Negative شامل دو دستور LNER و LNDR می‌باشد. منفی قدر مطلق عملوند دوم را جایگزین عملوند اول می‌کند.

$LNER \quad R_1, R_2$

$LNDR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه قسمت غیر نمایی نتیجه صفر باشد.
- ۱ اگر نتیجه کوچکتر از صفر باشد.

Load Positive شامل دو دستور LPER و LPDR است. قدر مطلق عملوند دوم را جایگزین عملوند اول می‌کند.

$LPER \quad R_1, R_2$

$LPDR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه قسمت غیر نمایی نتیجه صفر باشد.
- ۲ اگر نتیجه بزرگتر از صفر باشد.

Load Rounded شامل دو دستور LRER و LRDR است. عملوند دوم را به بزرگترین عدد صحیح کوچکتر از آن گرد می‌کند و حاصل را در عملوند اول می‌ریزد.

$LRER \quad R_1, R_2$

$LRDR \quad R_1, R_2$

کد وضعیت را تغییر نمی‌دهد.

Multiply شامل دستورات MER ، ME ، MDR ، MD ، MXDR ، MXD و MXR است. عملوند اول و دوم را ضرب می کند و حاصلضرب نرمال‌شده را جایگزین عملوند اول می‌کند.

$MER \quad R_1, R_2$

$ME \quad R_1, D_2(X_2, B_2)$

$MDR \quad R_1, R_2$

$MD \quad R_1, D_2(X_2, B_2)$

$MXDR \quad R_1, R_2$

$MXD \quad R_1, D_2(X_2, B_2)$

$MXR \quad R_1, R_2$

کد وضعیت را تغییر نمی‌دهد.

Store شامل دستورات STE و STD است. عملوند اول بدون تغییر در محل عملوند دوم جایگزین می‌شود.

$STE \quad R_1, D_2(X_2, B_2)$

$STD \quad R_1, D_2(X_2, B_2)$

کد حالت عوض نمی‌شود.

Subtract Normalized شامل دستورهای SER ، SE ، SDR ، SD و SXR است. عملوند دوم را از عملوند اول کم می‌کند و حاصل نرمال‌شده را در عملوند اول جایگزین می‌کند.

$SER \quad R_1, R_2$

$SE \quad R_1, D_2(X_2, B_2)$

$SDR \quad R_1, R_2$

$SD \quad R_1, D_2(X_2, B_2)$

$SXR \quad R_1, R_2$

کد وضعیت حاصل:

- صفر چنانچه قسمت غیر نمایی نتیجه صفر باشد.
- ۱ اگر نتیجه کوچکتر از صفر باشد.
- ۲ اگر نتیجه بزرگتر از صفر باشد.

Subtract Unnormalized شامل دستورهای SUR ، SU ، SWR و SW است. عملوند دوم را از عملوند اول کم می‌کند و حاصل نرمال‌نشده را در عملوند اول جایگزین می‌کند.

$SUR \quad R_1, R_2$

$SU \quad R_1, D_2(X_2, B_2)$

$SWR \quad R_1, R_2$

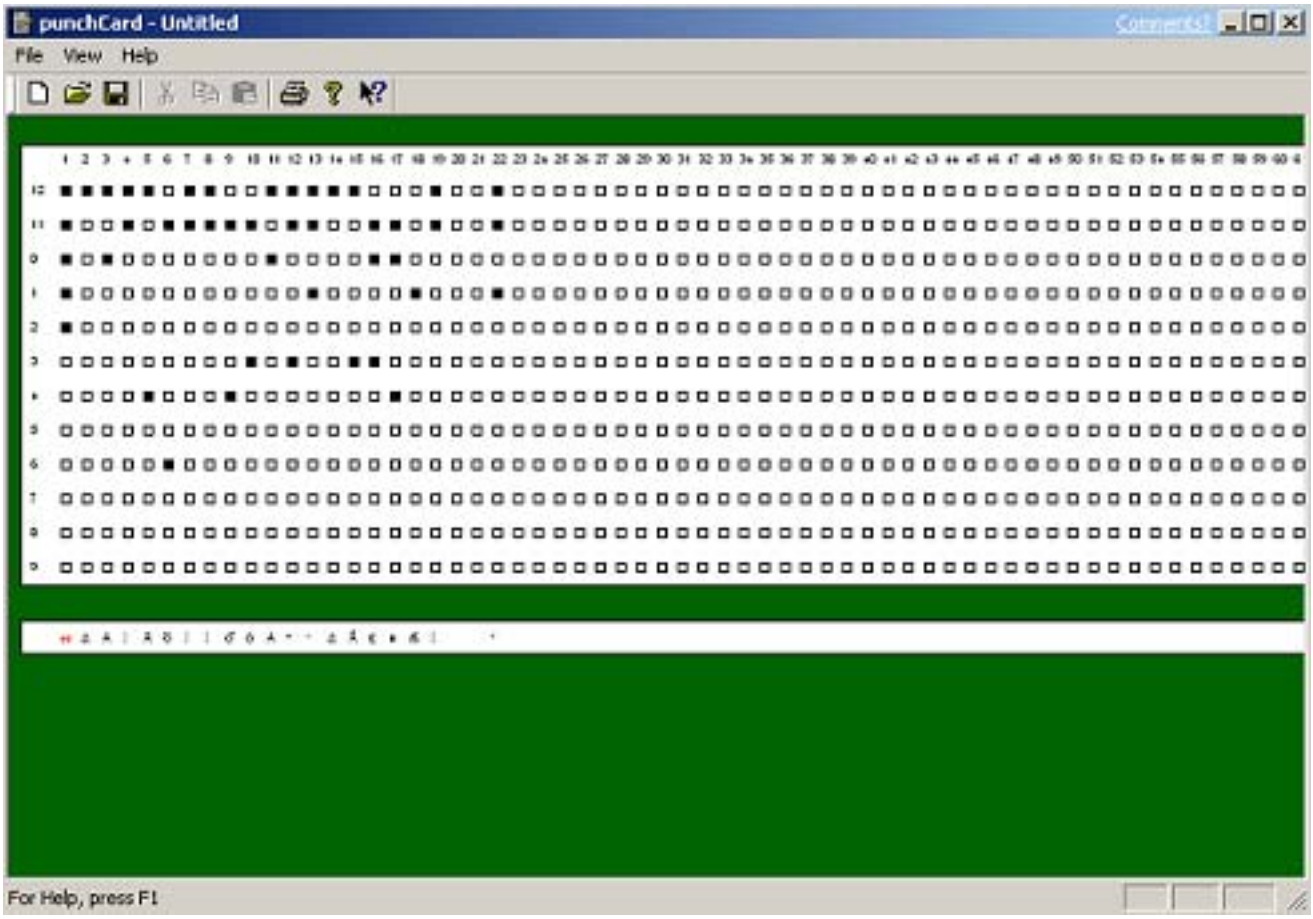
$SW \quad R_1, D_2(X_2, B_2)$

کد وضعیت حاصل:

- صفر چنانچه قسمت غیر نمایی نتیجه صفر باشد.
- ۱ اگر نتیجه کوچکتر از صفر باشد.
- ۲ اگر نتیجه بزرگتر از صفر باشد.

پیوست ۲ (برنامه کارت پانچ)

این برنامه در راستای شبیه‌سازی بخشی از توانایی‌های ماشین IBM370 نوشته‌شده‌است. به کمک این برنامه می‌توانیم یک کارت پانچ ایجاد و نقاط مورد نظر از آن را سوراخ‌کنیم. با انتخاب گزینه save از منوی فایل می‌توان محتویات کارت ساخته‌شده را در یک فایل ذخیره‌نمود. همچنین با انتخاب open از منوی فایل می‌توانیم یک فایل را که قبلا تهیه شده، باز کرده اطلاعات آن را بخوانیم و یا آن را ویرایش کنیم. در پایین هر ستون کارت کاراکتر کد EBCDIC معادل آن نوشته‌می شود. در صورتی که ساختار یک ستون اشتباه‌باشد در پایین با رنگ قرمز کلمه er نوشته‌می‌شود. در شکل B—۱ نمونه‌ای از یک کارت‌پانچ ساخته‌شده توسط این برنامه دیده‌می‌شود.



شکل B—۱: محیط برنامه کارت‌پانچ

فهرست الفبایی

آدرس
جابجایی، ۱۳
شاخص، ۱۲
مبنا، ۱۲
انشعاب، ۱۰

پردازنده، ۹
پردازنده انجام دستورات، ۹
پردازنده محاسبات ممیز ثابت، ۹
پردازنده محاسبات ممیز شناور، ۹

ثبات عمومی، ۹
ثبات کنترلی، ۹
ثبات ممیز شناور، ۹
ثبات وضعیت برنامه، ۹

حافظه اصلی، ۹
حافظه‌ی داخل پردازنده، ۹

دامپ، ۱۹
درشت‌دستورالعمل، ۱۸

عملوند، ۱۱
عملوند ثباتی، ۱۱
عملوند حافظه، ۱۱
عملوند مستقیم، ۱۱

فیلد زبان اسمبلی، ۱۵
فیلد اسم، ۱۵
فیلد توضیحات، ۱۵
فیلد عملگر، ۱۵
فیلد عملوند، ۱۵

قالب دستور، ۱۱
قالب زبان اسمبلی، ۱۴

کانال، ۹
کانال سلکتور، ۱۰
کانال مالتی‌پلکسر، ۱۰
کلمه، ۹

کلمه مضاعف، ۹
کلمه وضعیت برنامه، ۱۰، ۱۳
کنترل گذرگاه حافظه، ۹
محاسبه آدرس، ۱۲
نماد، ۱۸
نیم‌کلمه، ۹
وضعیت سرپرست، ۷
وضعیت عادی، ۷
وقفه، ۱۰

A، ۲۸
AD، ۴۸
ADR، ۴۸
AE، ۴۸
AER، ۴۸
AH، ۲۸
AL، ۲۹
ALR، ۲۹
AP، ۴۵
AR، ۲۸
AU، ۴۸
AUR، ۴۸
AW، ۴۸
AWR، ۴۸
AXR، ۴۸

BAL، ۲۹
BALR، ۲۹
BC، ۳۰
BCR، ۳۰
BCT، ۳۰
BCTR، ۳۰
BXH، ۳۰
BXLE، ۳۰

C، ۳۰
CD، ۴۹
CDR، ۴۹
CDS، ۳۱
CE، ۴۹
CER، ۴۹
CH، ۳۱
CL، ۳۲
CLC، ۳۲

CLCL، ۳۳
CLI، ۳۲
CLM، ۳۲
CLR، ۳۲
CP، ۴۵
CR، ۳۰
CS، ۳۱
CVB، ۳۳

D، ۳۳
DD، ۴۹
DDR، ۴۹
DE، ۴۹
DER، ۴۹
DP، ۴۵
DR، ۳۳

ED، ۴٦
EDMK، ۴٦

IC، ۳۴
ICM، ۳۴

L، ۳۴
LA، ۳۵
LCDR، ۵۰
LCER، ۵۰
LCR، ۳۵
LD، ۴۹
LDR، ۴۹
LE، ۴۹
LER، ۴۹
LH، ۳۵
LM، ۳٦
LNDR، ۵۰
LNER، ۵۰
LNR، ۳٦
LPDR، ۵۱
LPER، ۵۱
LPR، ۳٦
LR، ۳۴
LRDR، ۵۱
LRER، ۵۱
LSDL، ۳۹
LTDR، ۵۰

LTER، ۵۰
LTR، ۳۵

M، ۳۷
MD، ۵۱
MDR، ۵۱
ME، ۵۱
MER، ۵۱
MH، ۳۷
MP، ۴٦
MR، ۳۷
MVC، ۳٦
MVCL، ۳۷
MVI، ۳٦
MVN، ۳۷
MVO، ۳۷
MVZ، ۳۷
MXD، ۵۱
MXDR، ۵۱
MXR، ۵۱

N، ۲۹
NC، ۲۹
NI، ۲۹
NR، ۲۹

O، ۳۸
OC، ۳۸
OI، ۳۸
OR، ۳۸

PACK، ۳۸

S، ۴۲
SD، ۵۲
SDR، ۵۲
SE، ۵۲
SER، ۵۲
SH، ۴۲
SL، ۴۳
SLA، ۳۹
SLDA، ۳۸
SLL، ۳۹
SLR، ۴۳
SP، ۴۷
SPM، ۳۸

SR، ۴۲
SRA، ۴۰
SRDA، ۴۰
SRDL، ۴۰
SRL، ۴۰
SRP، ۴٦
ST، ۴۱
STC، ۴۱
STCK، ۴۱
STCM، ۴۱
STD، ۵۱
STE، ۵۱
STH، ۴۲
STM، ۴۲
SU، ۵۲
SUR، ۵۲
SW، ۵۲
SWR، ۵۲
SXR، ۵۲

TR، ۴۳
TRT، ۴۳

UNPK، ۴۴

X، ۳۳
XC، ۳۳
XI، ۳۳
XR، ۳۳

ZAP، ۴۷

کتاب‌نامه

[۱] والتر جی راد ، *برنامه‌نویسی به زبان اسمبلی سیستم* IBM360/370 ، ترجمه‌ی سیدمحمدتقی روحانی رانکوهی ، جلوه ،۱۳۷۴.

[2] *Assembly language programming: The IBM system 360 and 370*, George Strubble, MC-Graw Hill.

[3] *Assembly language Step-by-Step*, Jeff Dunterman, Wiley, 2nd Ed.

[4] *IBM System/370 Principles of Operations*, Order No. GA22-7000-4, IBM Corp., 1974.

[5] *IBM System/370 System Summary*, Order no. GA22-7001-4, IBM Corp., 1975.