فصل ٦. کتابخانه توابع عمومی emu8086

برخی توابع عمومی وجود دارند که جهت آسان‌تر شدن برنامه‌نویسی می‌توان آن‌ها را به برنامه خود اضافه نمایید. برای اینکه از توابع تعریف‌شده در فایل دیگری بهره ببرید، باید دستور <نام فایل> INCLUDE را در ابتدای کد خود اضافه نمایید. کامپایلر به‌طور خودکار ابتدا فایل موردنظر را در شاخه‌ای که برنامه‌ی شما قرار دارد جستجو می‌کند و درصورتی‌که فایل را نیابد، در زیرشاخه inc (\emu8086\inc) به جستجو آن می‌پردازد. در حال حاضر ممکن است به‌طور کامل محتوای فایل emu8086.inc (که در زیرشاخه inc قرار دارد) را درک نکنید، اما تنها کافی است بدانید توابع موجود در این فایل می‌توانند چه‌کاری انجام دهند. برای استفاده از هر یک از این توابع موجود در کتابخانه ابتدا باید دستور زیر را در برنامه تایپ کنید:

```
include 'emu8086.inc'
```

٦-١. ماکروهای کتابخانه emu8086.inc

ماکروهای زیر از طریق کتابخانه emu8086 قابل‌دسترسی هستند:

- **PUTC char**: یک کاراکتر را در محل مکان‌نما چاپ خواهد کرد.
- **GOTOXY col, row**: این ماکرو با دو پارامتر سطر و ستون، مکان‌نما (cursor) را به محل موردنظر خواهد برد.
- **PRINT string**: ماکرویی با یک پارامتر که یک‌رشته را چاپ می‌کند.
- **PRINTN string**: ماکرویی با یک پارامتر که پس از چاپ رشته، به‌طور خودکار به خط بعد خواهد رفت.
- **CURSOROFF**: مکان‌نما را مخفی می‌کند.

- **CURSORON**: مکان‌نما را ظاهر می‌کند.

برای اســـتفاده از هر یک از ماکروهای بالا تنها کافی اســـت نام و پارامترهای آن را در برنامه تایپ نمایید. برای مثال:

```
include ‘emu8086.inc’
ORG   100h

PRINT 'Hello World!'
GOTOXY 10, 5
PUTC 65          کد اسکی ’A‘ برابر 65 است;
PUTC 'B'

RET              برگشت به سیستم عامل ;
END              دستور توقف کامپایلر ;
```

وقتی‌کـه کـامپـایلر کـد منبع را پردازش می‌کنـد فایل emu8086.inc را جهت تعریف مـاکروهـا و جایگزین کردن کد واقعی به‌جای نام ماکرو جســـتجو می‌کند. ماکروها معمولاً قســمت‌های نسبتاً کوچکی از کد هستند، اما استفاده مکرر از آن‌ها ممکن است حجم برنامه نهایی بسـیار زیاد نماید؛ بنابراین بهتر است از رَویه‌های (Procedures) موجود در کتابخانه emu8086.inc اســـتفـاده نماییم (رَویه‌ها جهت بهینه‌ســـازی اندازه بهتر از ماکروها عمل می‌کنند).

٦-٢. رَویه‌های کتابخانه emu8086.inc

در فایل کتابخانه‌ای emu8086.inc رَویه‌های زیر نیز تعریف‌شده‌اند:

- **PRINT_STRING**: رویه‌ی که رشـــته‌ای را به نام NULL ختم شـــده در محل جـاری چاپ می‌کند. آدرس رشـــته از طریق ثبات DS:SI دریافت می‌شـــود. جهت استفاده بایستی قبل از دستور END عبارت DEFINE_PRINT_STRING را اعلان نماییم.

- **PTHIS**: کاملاً شـــبیه PRINT_STRING عمل می‌کند با این تفاوت که آدرس رشته را از پشته دریافت خواهد کرد. برای مثال:

```
CALL PTHIS
db 'Hello World!', 0
```

برای اســـتفـاده از این رویـه قبـل از END عبارت DEFINE_PTHIS را تایپ می‌کنیم.

- **GET_STRING**: یک‌رشـــته ختم شـــده به تهی را از کاربر دریافت و آن را در آدرس DS:DI ذخیره می‌کند. قبل از اسـتفاده از این دسـتور باید اندازه بافر در DX ذخیره شود. این رویه پس از فشرده شدن کلید ENTER به کار خود پایان می‌دهد. جهت اسـتفاده از این رویه بایسـتی DEFINE_GET_STRING قبل از END تعریف شود.
- **CLEAR_SCREEN**: رویه‌ی است که صفحه را پاک‌کرده و محل مکان‌نما را به بـالـای صـــفـحـه انـتـقـال مـی‌دهـد. بـایـد قـبـل از END دســـتـور DEFINE_CLEAR_SCREEN تعریف شود.
- **SCAN_NUM**: رویـه‌ی کـه اعـداد چنـدرقمی علامت‌دار را از صـــفحه‌کلید دریافت کرده و آن‌ها را در ثبات CX قرار می‌دهد جهت اســـتفاده باید قبل از END دستور DEFINE_SCAN_NUM را اعلان نماییم.
- **PRINT_NUM**: رویه‌ی که یک عدد علامت‌دار را در ثبات AX قرار می‌دهد و چاپ می‌کند. جهت اســـتفاده بایســـتی عبارات DEFINE_PRINT_NUM و DEFINE_PRINT_NUM_UNS قبل از END اعلان شود.
- **PRINT_NUM_UNS**: رویـه‌ی که عدد بدون علامت قرارگرفته در AX را در خـروجـی چـاپ مـی‌کـنـد. جـهـت اســـتـفـاده بـایـد عـبـارت DEFINE_PRINT_NUM_UNS قبل از END اعلان شود.

جهت اسـتفاده از هرکدام از رویه‌ها ابتدا باید قبل از END نام تابع را اعلان و سـپس از دستور CALL جهت اجرای آن‌ها کمک بگیریم.

```
include 'emu8086.inc'

ORG   100h

LEA   SI, msg1              ; پیام ورود یک عدد
CALL  print_string
CALL  scan_num              ; عدد ورودی را در ثبات CX قرار می دهد


 MOV   AX, CX               ;عدد ورودی را در Ax کپی می کند

CALL  pthis                 ; رشته خط بعدی را چاپ می کند
DB  13, 10, 'You have entered: ', 0

 CALL  print_num            ; عددی که در AX قرار دارد را چاپ می کند

RET                         ; برگشت به سیستم عامل
msg1  DB  'Enter the number: ', 0

DEFINE_SCAN_NUM
DEFINE_PRINT_STRING
DEFINE_PRINT_NUM
DEFINE_PRINT_NUM_UNS  ;برای دستور print_num نیاز است
DEFINE_PTHIS

END                         ; دستور توقف کامپایلر
```

در ابتـدا کـامپایلر تعریف‌ها (اعلان‌ها) را پردازش می‌نماید. پس اینکه کامپایلر دســـتور CALL را دریافت می‌کند، نام رویه را با آدرس محلی که کد رویه در آن تعریف‌شـده است، جایگزین می‌نماید. وقتی‌که دسـتور CALL اجرا می‌شـود، کنترل به رویه برمی‌گردد. این کار بسـیار مفید اسـت، چون اگر حتی ۱۰۰ مرتبه رویه‌ای را در برنامه صدا زده باشید، همچنان برنامه اجرایی نسبتاً کوچکی خواهید داشت.

کمی پیچیده به نظر می‌رســـد، این‌طور نیســـت؟ نگران نباشـــید به‌مرور زمان مطالب بیشتری فرا خواهید گرفت. در حال حاضر نیاز به درک اصول پایه دارید.

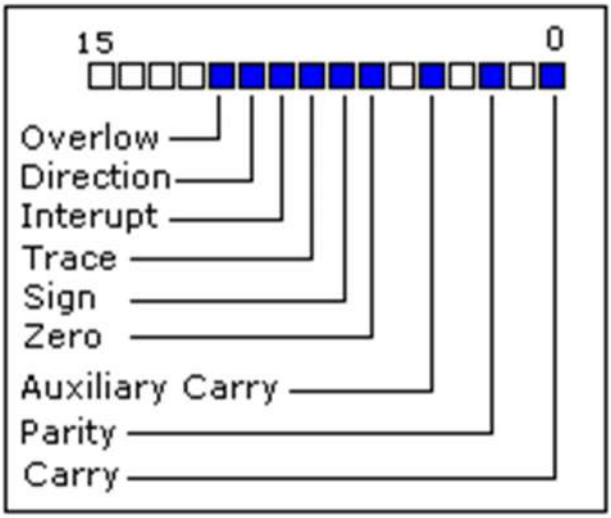
٦-٣. تمرین فصل ششم

۱. برنامه‌ای بنویسید که رشته‌ای کاراکتری را از کاربر دریافت کرده، سپس آن را در وسط صفحه، نمایش دهد.

۲. برنامه‌ای بنویسید که رشته‌ای کاراکتری را از کاربر دریافت کرده، سپس آن را در سطر و ستونی که کاربر مشخص می‌کند نمایش دهد.

فصل ۷. دستورات منطقی و محاسباتی

اغلب دستورات منطقی و محاسباتی بر ثبات وضعیت (پرچم یا flags) تأثیر می‌گذارند.



شکل ۷-۱) ثبات وضعیت (Status register) یا ثبات پرچم‌ها (Flags)

۷-۱. معرفی پرچم‌ها در ثبات وضعیت

ثبات وضـــعیت یک ثبات 16 بیتی اســـت که در زیر به تعریف هر یک از بیت‌های آن می‌پردازیم. هر بیت یک پرچم نامیده می‌شود و می‌تواند ارزش 0 یا 1 داشته باشد.

- **Carry Flag (CF)**: این پرچم زمانی که یک ســـرریز بدون علامت[1] رخ دهد مقدار 1 خواهد گرفت. به‌عنوان مثال وقتی‌که شما 255 را با یک جمع می‌کنید

[1] unsigned overflow

دیگر نتیجه در بازه 0 تا 255 نخواهد بود، بنابراین CF یک خواهد. زمانی که سرریز وجود ندارد این پرچم صفر خواهد بود.

- **Zero Flag (ZF)**: زمانی که محاسبه نتیجه صفر داشته باشد این پرچم یک خواهد شد. برای نتیجه غیر صفر این پرچم صفر خواهد شد.
- **Sign Flag (SF)**: زمانی که نتیجه محاسبه منفی باشد مقدار این پرچم یک و اگر نتیجه مثبت باشد مقدار این پرچم صفر خواهد بود. در حقیقت این پرچم ارزش بالاترین[۲] bit را خواهد گرفت.
- **Overflow Flag (OF)**: این پرچم زمانی که سرریز علامت‌دار رخ دهد برابر با 1 خواهد شد. برای مثال زمانی که 100 با 50 جمع شود بازه اعداد دیگر در بین 128- تا 127 نخواهد بود.
- **Parity Flag (PF)**: این پرچم برابر با 1 خواهد شد زمانی که درنتیجه محاسبه تعداد بیت‌های «یک» زوج باشد (مثلاً اگر عدد 11010100 نتیجه باشد) و زمانی که نتیجه دارای تعداد بیت‌های «یک» فرد باشد، مقدار این پرچم 0 خواهد بود. اگر نتیجه یک کلمه باشد تنها 8 بیت پایینی بررسی می‌شود.
- **Auxiliary Flag (AF)**: زمانی که سرریز در نیبل پایینی[۳] (نیبل = 4 بیت) رخ دهد این پرچم برابر ۱ خواهد شد.
- **Interrupt enable Flag (IF)**: زمانی که این پرچم برابر 1 شود CPU به وقفه دستگاه خارجی پاسخ می‌دهد.
- **Direction Flag (DF)**: این پرچم با بعضی از دستورات که مربوط به پردازش داده‌های رشته‌ای می‌شوند ارتباط دارد، وقتی مقدار این پرچم 0 باشد پردازش روبه‌جلو (از چپ به راست) و زمانی که 1 باشد پردازش رو به عقب (از راست به چپ) خواهد بود.

---

[2] most significant bit

[3] Low nibble

۷-۲. دستورات محاسباتی

سه گروه دستور محاسباتی در زبان اسمبلی وجود دارد:

- دستورالعمل‌هایی ADD, SUB,CMP, AND, TEST, OR, XOR با دو عملوند
- دستورالعمل‌هایی MUL, IMUL, DIV, IDIV با یک عملوند
- دستورالعمل‌های INC, DEC, NOT, NEG با یک عملوند

در ادامه به معرفی هر یک از سه گروه بالا خواهیم پرداخت.

۷-۲-۱. دستورات محاسباتی: ADD, SUB,CMP, AND, TEST, OR, XOR

این دستورالعمل‌ها عملوندهای زیر را پشتیبانی می‌کنند:

REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
memory: [BX], [BX+SI+7], variable, etc...
immediate: 5, -24, 3Fh, 10001101b, etc...

بعد از انجام عملیات توسط دستورالعمل‌های فوق همیشه نتیجه در عملوند اول ذخیره می‌شود. دستورات CMP و TEST تنها بر روی پرچم‌ها اثر می‌کنند و نتیجه را ذخیره نمی‌کنند (از این دستورالعمل‌ها جهت اخذ تصمیم در طول اجرای برنامه استفاده می‌شود.)

این دستورالعمل‌ها تنها بر روی پرچم‌های CF, ZF, SF, OF, PF, AF تأثیر می‌گذارند.

۷-۲-۱-۱. الگوریتم عملکرد دستورالعمل‌های گروه اول

- **ADD**: عملوند دوم را به عملوند اول اضافه می‌کند.
- **SUB**: عملوند دوم را از عملوند اول کم می‌کند.
- **CMP**: عملوند دوم را از اولی کم کرده و تنها بر روی پرچم‌ها تأثیر می‌گذارند.

- **AND و OR و XOR**: برحسب دستورالعمل، عملیات منطقی را بر روی تمامی بیت‌های بین دو عملوند اعمال می‌کنند. جدول زیر نمایش‌دهنده نتیجه

جدول ۷-۲) نتیجه اعمال عملگرهای منطقی بین دو بیت دو عملوند

bite 1	bite 2	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

اِعمال عملگرهای منطقی بالا، بین تنها دو بیت هم ارزش از دو عملوند، است:

- **TEST**: مانند دستور AND است، اما نتیجه تنها بر روی پرچم‌ها اعمال می‌شود.

۷-۲-۲. گروه دوم دستورات محاسباتی: `MUL, IMUL, DIV, IDIV`

این گروه از دستورالعمل‌های تک عملوندی، عملوندهای زیر را پشتیبانی می‌کنند:

REG
Memory

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
memory: [BX], [BX+SI+7], variable, etc...

دستورالعمل‌های MUL و IMUL تنها بر روی پرچم‌های CF و OF اثر می‌گذارند. به بیانی دیگر هنگامی‌که نتیجه از اندازه عملوند بیشتر باشد این پرچم‌ها 1 و در غیر این صورت 0 هستند. برای دستورالعمل‌های DIV, IDIV پرچمی در نظر گرفته نشده است.

۷-۲-۲-۱. الگوریتم عملکرد دستورالعمل‌های گروه دوم

- **MUL**: ضرب بدون علامت

عملوند * AL = AX: وقتی عملوند بایت (BYTE) باشد.

عملوند * AL = (DX AX): وقتی عملوند کلمه (WORD) باشد.

- **IMUL**: **ضرب علامت‌دار**

عملوند * AL = AX: وقتی عملوند بایت (BYTE) باشد.

عملوند * AL = (DX AX): وقتی عملوند کلمه (WORD) باشد.

- **DIV**: **تقسیم بدون علامت**

وقتی عملوند بایت (BYTE) باشد:

AL = AX /عملوند

AH = باقیمانده (modulus).

وقتی عملوند کلمه (WORD) باشد:

AL = (DX AX)/عملوند

DX = باقیمانده (modulus).

- **IDIV**: **تقسیم علامت‌دار**

وقتی عملوند بایت (BYTE) باشد:

AL = AX /عملوند

AH = باقیمانده (modulus).

وقتی عملوند کلمه (WORD) باشد:

AL = (DX AX)/عملوند

DX = باقیمانده (modulus).

۷-۲-۳. گروه سوم دستورات محاسباتی: INC, DEC, NOT, NEG

این گروه از دستورالعمل‌ها عملوندهای زیر را پشتیبانی می‌کنند:

REG
Memory

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
memory: [BX], [BX+SI+7], variable, etc...

دستورالعمل‌های INC, DEC تنها بر روی پرچم‌های ZF, SF, OF, PF, AF تأثیر می‌گذارند. دستورالعمل NEG علاوه بر پرچم‌های یادشده بر روی پرچم CF نیز اثر می‌گذارند. دستورالعمل NOT بر هیچ پرچمی تأثیر ندارد.

۷-۲-۳-۱. الگوریتم عملکرد دستورالعمل‌های گروه سوم

- **NOT**: تک‌تک بیت‌های یک عملوند را معکوس می‌کند.
- **NEG**: علامت عملوند را تغییر می‌دهد (متمم دو). به‌طور مشخص تمامی بیت‌های عملوند را معکوس کرده و سپس عملوند را با یک جمع می‌کند. برای مثال عملوند 5 به 5- تبدیل و عملوند 2- به 2 تغییر خواهد شد.
- **INC**: به عملوند یک واحد اضافه می‌کند.
- **DEC**: از عملوند یک واحد کم می‌کند.

۷-۳. تمرین فصل هفتم

۱. دو عدد 0F345h و 3219h را با یکدیگر جمع و مقدار بیت‌ها در ثبات وضعیت را مشخص نمایید. سپس برنامه‌ای برای این‌کار بنویسید و نتیجه را با محاسبات خود چک نمایید.

۲. دو عدد را از کاربر دریافت و نتیجه جمع، تفریق، ضرب و تقسیم آن‌ها را در خروجی با قالب مناسب نمایش دهید.

۳. برنامه‌ای بنویسید که عملگرهای منطقی AND، OR و XOR را برروی اعداد 0EFh و 0F8h اعمال و نتیجه را با فرمت مناسب در خروجی نمایش دهد.

٤. برنامه‌ای بنویسید که طول و عرض مستطیلی را دریافت و محیط و مساحت آن را در خروجی چاپ نماید.

۵. برنامه‌ای بنویسید که قاعده و ارتفاع مثلثی را از کاربر دریافت نموده و مساحت آن را در خروجی چاپ نماید.

۶. برنامه‌ای بنویسید که دو عدد X و Y را دریافت سپس محتوای آن دو را با یکدیگر عوض نماید. (برنامه را با استفاده از متغیر کمکی و یک‌بار دیگر بدون استفاده از آن بنویسید.)

۷. برنامه‌ای بنویسید که ۳ عدد را از کاربر دریافت نموده سپس میانگین آن‌ها را در خروجی چاپ نماید.

فصل ۸. کنترل جریان برنامه

کنترل جریان برنامه از اهمیت بالایی برخوردار است در اینجاست که برنامه شما می‌تواند تصمیم بگیرد که بر اساس شرایط خاص چه عکس‌العملی داشته باشد.

۸-۱. پرش‌های غیرشرطی

دستور پایه که مسئولیت انتقال کنترل برنامه به مکان دیگر را برعهده دارد، دستور JMP است. نحوه استفاده از این دستور به‌صورت زیر است:

```
JMP label
```

برای مشخص کردن یک برچسب[1] در برنامه تنها کافی است نام آن را تایپ و در انتهای آن علامت ':' بگذاریم. برچسب‌ها با ترکیبی از کاراکترها ساخته می‌شوند، به‌شرط اینکه با عدد آغاز نشوند. سه مثال زیر نمونه‌ای از تعریف صحیح برچسب است:

```
label1:
label2:
a:
```

برچسب و دستورات برنامه را می‌توان در خط جداگانه و یا در یک خط به کاربرد؛ مانند:

```
x1:
MOV AX, 1

x2: MOV AX, 2
```

[1] lable

مثال: برنامه زیر جمع دو عدد با استفاده از دستور JMP را نمایش می‌دهد. همان‌طور که از برنامه مشخص است دستور JMP توانایی انتقال کنترل به دستورات قبل و یا دستورات بعد[۲] از خود را دارد. همچنین این دستور می‌تواند به نقطه‌ای از سگمنت کد جاری پرش داشته باشد (65,535 byte).

```
ORG 100H

MOV  AX, 5              ; AX = 5.
MOV  BX, 2              ; BX = 2.
JMP   CALC              ; برو به برچسب 'CALC'
BACK:
      JMP STOP          ; برو به برچسب 'STOP'
CALC:
      ADD AX, BX        ; AX =  AX + BX
      JMP BACK          ; برو به برچسب 'BACK'
STOP:
RET                     ; برگشت به سیستم عامل
```

۸-۲. پرش شرطی کوتاه

برخلاف دستورالعمل JMP دستورالعمل‌های دیگری وجود دارند که پرش شرطی را انجام می‌دهند (پرش درصورتی‌که برخی از شرایط برقرار است) این دستورالعمل‌ها به سه دسته تقسیم می‌شوند:

- دسته اول دستورالعمل‌هایی که تنها یک پرچم را چک می‌کند.
- دسته دوم دستورالعمل‌هایی که دو عدد علامت‌دار را مقایسه می‌کند.
- دسته سوم دستورالعمل‌هایی که اعداد بدون علامت را مقایسه می‌کند.

[2] forward and backward

۸-۲-۱. پرش‌های شرطی که تنها یک پرچم را آزمایش می‌کنند

جدول ۸-۱) تعریف دستورالعمل‌های پرش شرطی با بررسی یک پرچم

دستور	توصیف عملکرد	شرط	دستور مقابل
JZ, JE	پرش در صورت مساوی (صفر) بودن.	ZF = 1	JNZ, JNE
JC, JB, JNAE	پرش در صورت وجود رقم نقلی داشتن. (کوچک‌تر بودن، نه بزرگ‌تر مساوی)	CF = 1	JNC, JNB, JAE
JS	پرش در حالت منفی بودن.	SF = 1	JNS
JO	پرش درصورتی‌که سرریز وجود داشته باشد.	OF = 1	JNO
JPE, JP	پرش درصورتی‌که بیت توازن زوج باشد.	PF = 1	JPO
JNZ, JNE	پرش در صورت مساوی (صفر) نبودن.	ZF = 0	JZ, JE
JNC, JNB, JAE	پرش درصورتی‌که رقم نقلی موجود نباشد. (بزرگ‌تر مساوی بودن، نه کوچک‌تر)	CF = 0	JC, JB, JNAE
JNS	پرش در حالت مثبت بودن.	SF = 0	JS
JNO	پرش درصورتی‌که سرریز وجود نداشته باشد.	OF = 0	JO
JPO, JNP	پرش درصورتی‌که بیت توازن فرد باشد.	PF = 0	JPE, JP

همان‌طور که ممکن است تاکنون متوجه شده باشید دستورالعمل‌هایی وجود دارند که عمل مشابهی انجام می‌دهند، این امر درست است. آن‌ها حتی با کد ماشین[3] مشابهی اسمبل می‌شوند. خوب خواهد بود به خاطر بسپارید زمانی که دستور JE کامپایل می‌شود، مانند دستورات JZ و JC به شکل JB اسمبل می‌شود.

اسامی مختلف دستورات کمک به درک آسان‌تر برنامه کمک می‌کنند و همچنین سهولت در به خاطر آوردن کد را برای برنامه‌نویس به همراه دارند.

[3] Machine code

اگر کد زیر را شبیه‌سازی نمایید مشاهده خواهید نمود تمامی دستورات به JNB اسمبل خواهند شد، کد مؤثر برای این دستور 73h با طول ۲ بایت است، اگر شرط درست باشد بایت دوم تعداد بیت‌های که باید به ثبات IP اضافه شوند را مشخص می‌کند.

این دستور تنها یک بایت برای نگهداری آفست دارد که انتقال کنترل برنامه را به 128- بایت به عقب و یا 127 بایت به جلو محدود می‌نماید. بایت آفست همواره علامت‌دار است.

```
ORG 100H

 JNC A
 JNB A
 JAE A

 MOV AX, 4
A: MOV AX, 5

RET             برگشت به سیستم عامل ;
```

۸-۲-۲. پرش شرطی برای اعداد علامت‌دار

معمولاً زمانی که نیاز به مقایسه ارزش اعداد (علامت‌دار یا بدون علامت) داشته باشیم از دستور CMP استفاده می‌کنیم.

عملکرد این دستور شبیه به دستور SUB است با این تفاوت که نتیجه تنها بر روی پرچم‌ها[۴] تأثیر خواهد کرد. برای مثال به مقایسه بین اعداد (5, 2) و (7, 7) در مثال زیر توجه کنید:

```
CMP 5,2    ; 5-2=3   ZF = 0 (Zero Flag is set to 0).

CMP 7,7    ; 7-7=0   ZF = 1 (Zero Flag is set to 1 and JZ or JE will do the jump).
```

[4] Flags

جدول ۸-۲) تعریف دستورالعمل‌های پرش شرطی مقایسه کننده دو عدد علامت‌دار

دستور	توصیف عملکرد	شرط	دستور مقابل
JE, JZ	پرش در صورت مساوی بودن (=). پرش در صورت صفر بودن.	ZF = 1	JNE, JNZ
JNE, JNZ	پرش در صورت نامساوی بودن (< >). پرش در صورت صفر نبودن.	ZF = 0	JE, JZ
JG, JNLE	پرش در صورت بزرگ‌تر بودن (>). پرش در صورت کوچک‌تر مساوی نبودن (not <=).	ZF = 0 and SF = OF	JNG, JLE
JL, JNGE	پرش در صورت کوچک‌تر بودن (<). پرش در صورت بزرگ‌تر مساوی نبودن (not >=).	SF <> OF	JNL, JGE
JGE, JNL	پرش در صورت بزرگ‌تر مساوی بودن (>=). پرش در صورت کوچک‌تر نبودن (not <).	SF = OF	JNGE, JL
JLE, JNG	پرش در صورت کوچک‌تر مساوی بودن (<=). پرش در صورت بزرگ‌تر نبودن (not >).	ZF = 1 or SF <> OF	JNLE, JG

تذکر: علامت <> به معنای نامساوی بودن[5] است.

[5] not equal

۸-۲-۳. پرش شرطی برای اعداد بدون علامت

جدول ۸-۳) تعریف دستورالعمل‌های پرش شرطی مقایسه کننده دو عدد بدون علامت

دستور	توصیف عملکرد	شرط	دستور مقابل
JE, JZ	پرش در صورت مساوی بودن (=). پرش در صورت صفر بودن.	ZF = 1	JNE, JNZ
JNE, JNZ	پرش در صورت مساوی نبودن (< >). پرش در صورت صفر نبودن.	ZF = 0	JE, JZ
JA, JNBE	پرش در صورت بیشتر بودن (>). پرش در صورت کمتر یا مساوی نبودن (not <=).	CF = 0 and ZF = 0	JNA, JBE
JB, JNAE, JC	پرش در صورت کمتر بودن (<). پرش در صورت بیشتر یا مساوی نبودن (not >=). پرش در صورت وجود رقم نقلی.	CF = 1	JNB, JAE, JNC
JAE, JNB, JNC	پرش در صورت بیشتر یا مساوی بودن (>=). پرش در صورت کمتر نبودن (not <). پرش در صورت وجود رقم نقلی.	CF = 0	JNAE, JB
JBE, JNA	پرش در صورت کمتر یا مساوی بودن (<=). پرش در صورت بیشتر نبودن (not >).	CF = 1 or ZF = 1	JNBE, JA

در زیر مثالی جهت آشنایی با نحوه استفاده از دستور CMP و دستورات شرطی آورده شده است. برای اجرای این مثال که با ارزش‌دهی دو مقدار متفاوت به ثبات‌های AL و BL

آغازشده است، بر روی دکمه Flags کلیک کرده و با اجرای مرحله‌به‌مرحله[۶] دستورات فرآیند اجرای برنامه را دنبال نمایید.

می‌توانید با استفاده از کلید میانبر F5 جهت کامپایل مجدد و بارگذاری مجدد برنامه در شبیه‌ساز اقدام کنید. بهتر است این برنامه را با دو مقدار برابر در AL و BL نیز اجرا نمایید.

```
include "EMU8086.INC"
ORG 100H
MOV AL, 25              ; AL = 25
MOV BL, 10              ; BL = 10
CMP AL, BL              ; AL – BL مقایسه
JE EQUAL                ; AL = BL (ZF = 1) پرش اگر
PUTC 'N'                ; (AL<> BL اگر) 'N' چاپ کن
JMP STOP                ; STOP پرش کن به
EQUAL:
        PUTC 'Y'        ; (AL= BL اگر) 'Y' چاپ کن
STOP:
RET                     ; برگشت به سیستم عامل
```

۸-۳. حلقه

حلقه‌ها اساساً شبیه به دستورات پرش هستند و می‌توان آن‌ها را با استفاده از پرش‌های شرطی و دستور مقایسه ساخت.

تمام دستورات حلقه از ثبات CX جهت شمارش تعداد تکرار حلقه استفاده می‌کنند. همان‌طور که می‌دانید ثبات CX شانزده بیتی است و بیشترین مقداری که می‌تواند نگهداری نماید 65535 یا 0FFFFh است.

بااین‌حال با حلقه‌های تودرتو[۷] می‌توان این مقدار را تا بی‌نهایت[۸] (انتهای حافظه اصلی یا حافظه پشته) افزایش داد.

---

[6] Single Step
[7] Nested loop
[8] 65535 * 65535 * 65535 ...till infinity

جدول ۸-٤) تعریف دستورالعمل‌های حلقه

دستور	عملگر و شرط پرش	دستور مقابل
LOOP	از CX یکی کم کرده و درصورتی‌که CX برابر صفر نباشد به برچسب پرش خواهد کرد.	DEC CX and JCXZ
LOOPE	از CX یکی کم کرده، اگر CX صفر نبوده و zf برابر یک باشد (نتیجه محاسبات دستورالعمل قبل از حلقه) به برچسب پرش می‌کند.	LOOPNE
LOOPZ		LOOPNZ
LOOPNE	از CX یکی کم کرده و درصورتی‌که CX صفر نبوده و zf برابر صفر باشد (نتیجه محاسبات دستورالعمل قبل از حلقه) به برچسب پرش می‌کند.	LOOPE
LOOPNZ		LOOPZ
JCXZ	درصورتی‌که CX صفر باشد به برچسب پرش می‌کند.	OR CX, CX and J NZ

همانطور که از جدول ۸-٤ نشان می‌دهد دو دستور LOOPZ و LOOPE و دو دستور LOOPNE و LOOPNZ مانند هم عمل می‌کنند.

برای استفاده از حلقه‌ها به‌صورت تودرتو می‌توان با استفاده از دستور push CX مقدار CX در حلقه خارجی[9] را در پشته قرار داد و پس از پایان حلقه داخلی[10]، مقدار CX مربوط به حلقه خارجی را با دستور pop CX از پشته فراخوانی نمود. به‌منظور درک روش مدیریت صحیح مقادیر CX حلقه‌ها، در هنگامی‌که چند حلقه‌ی تودرتو در برنامه وجود دارد، به مثال بعد توجه نمایید؛ در این مثال BX تعداد کل مراحل را می‌شمارد؛ به‌صورت پیش‌فرض شبیه‌ساز اعداد را در مبنای شانزده نمایش می‌دهد، می‌توانید با کلیک بر روی ثبات موردنظر در شبیه‌ساز مقدار آن را در سایر سیستم‌های عددی نیز مشاهده نمایید.

[9] External loop
[10] Internal loop

```
ORG 100H

MOV BX, 0                       ; شمارنده کلی مراحل

MOV CX, 5                       ; مقداردهی شمارنده حلقه اول
K1:  ADD BX, 1                  ; شروع دستورات حلقه اول
     MOV AL, '1'
     MOV AH, 0EH
     INT 10H
     PUSH CX                    ; قراردادن شمارنده حلقه اول در پشته
     MOV CX, 5                  ; مقداردهی شمارنده حلقه دوم
        K2: ADD BX, 1           ; شروع دستورات حلقه دوم
            MOV AL, '2'
            MOV AH, 0EH
            INT 10H
            PUSH CX             ; قراردادن شمارنده حلقه دوم در پشته
            MOV CX, 5           ; مقداردهی شمارنده حلقه سوم
               K3: ADD BX, 1    ; شروع دستورات حلقه سوم
                   MOV AL, '3'
                   MOV AH, 0EH
                   INT 10H
               LOOP K3          ; حلقه داخلی در داخل حلقه داخلی دیگر
         POP CX                 ; بازیابی مقدار شمارنده حلقه دوم از پشته
         LOOP K2                ; پایان حلقه دوم
 POP CX                         ; بازیابی مقدار شمارنده حلقه سوم از پشته
 LOOP K1                        ; پایان حلقه اول

RET
```

همانند سایر دستورات شرطی، حلقه‌ها نیز دارای دستور مقابل[11] هستند که در ایجاد راه‌حل کمک خواهند کرد. زمانی که آدرس محل دلخواهی از برنامه خیلی دور است، اسمبلر به‌طور خودکار دستورات ۵ بایتی را با ۲ بایتی جایگزین خواهد نمود که این موضوع را می‌توان در برگردان زبان ماشین[12] به زبان اسمبلی به‌خوبی مشاهده کرد.

[11] opposite companion
[12] disassembler

۸-٤. ترفند چیرگی بر محدودیت پرش‌های شرطی

تمامی پرش‌های شـرطی یک محدودیت بزرگ دارند، برخلاف دسـتور JMP آن‌ها تنها می‌توانند 127 بایت به جلو و یا 128 بایت به عقب پرش نمایند (یادآور می‌شـود کدماشین دسـتورات، اغلب ۳ بایت یا بیشتر است). با یک ترفند دوست‌داشتنی، به شکل زیر، می‌توان بر این محدودیت چیره شد:

۱- با کمک دستورات مقابل در جداول این فصل، به برچسب Label_x پرش کنید.

۲- با استفاده از دستور JMP به محل موردنظر پرش کنید

۳- برچسب Label_x را بعد از دستور JMP تعریف نمایید.

```
INCLUDE "EMU8086.INC"
ORG 100H
MOV AL, 5
MOV BL, 5
CMP AL, BL                  ; AL – BL مقایسه

; JE EQUAL                  ; تنها یک بایت است
JNE NOT_EQUAL               ; AL <> BL (ZF = 0) پرش اگر
JMP EQUAL

NOT_EQUAL:
            ADD BL, AL
            SUB AL, 10
            XOR AL, BL
            JMP SKIP_DATA

DB 256 DUP(0)               ; 256 BYTES
SKIP_DATA:
          PUTC 'N'          ; AL<> BL ‘N’ را چاپ کن اگر
          JMP STOP
EQUAL:
        PUTC 'Y'            ; AL= BL ‘Y’ را چاپ کن اگر

STOP:
RET
```

برچسب label_x می‌تواند هر نام دلخواهی معتبر و غیرتکراری داشته باشد. برای روشن شدن موضوع به مثال صفحه قبل توجه نمایید.

آخرین نسخه[۱۳] اسمبلر ۸۰۸۶ به‌طور خودکار با جایگزین کردن دستور مقابل پرش شرطی راه‌حل را ایجاد می‌کند و یک پرش غیرشرطی بزرگ را اضافه می‌نماید.

نکته: به‌ندرت از روش جایگزینی مقدار بی‌واسطه[۱۴] به‌جای برچسب استفاده می‌شود. یک مقدار بی‌واسطه که با کاراکتر $ آغاز می‌شود یک پرش نسبی را ایجاد می‌کند، در غیر این صورت کامپایلر تعداد دستورات را برای پرش مستقیم به افست داده‌شده محاسبه می‌کند.

به مثال زیر دقت کنید:

```
ORG 100H

; پرش غیر شرطی به جلو
; عبور از ۳ بایت بعدی  + خودش
;دستور ماشین پرش غیر شرطی کوتاه (JMP) دو بایت است.

 JMP $3+2
A DB 3        ; 1 BYTE.
B DB 4        ; 1 BYTE.
C DB 4        ; 1 BYTE.

; پرش شرطی پنج بایتی به عقب
MOV BL,9
DEC BL        ; 2 BYTES.
CMP BL, 0     ; 3 BYTES.
JNE $-5       ; پرش پنج بایت به عقب

RET
```

[۱۳] برای مطلع شدن از در اختیار داشتن آخرین نسخه به منوی Help->check for an update مراجعه نمایید.

[۱۴] immediate value

۸-۵. تمرین فصل هشتم

۱. برنامه‌ای بنویسید که دو عدد را از ورودی خوانده، مقدار بزرگ‌تر را چاپ نماید.
۲. برنامه‌ای بنویسید که سه عدد را دریافت و مقدار بزرگ‌تر را در خروجی چاپ نماید.
۳. برنامه‌ای بنویسید که سه عدد را از ورودی خوانده، آن‌ها را به ترتیب نزولی چاپ نماید.
٤. برنامه‌ای بنویسید که عدد N را از ورودی خوانده، سپس از N تا صفر را چاپ نماید.
۵. برنامه‌ای بنویسید که N عدد را از یک آرایه خوانده، سپس مجموع آن را چاپ نماید.
٦. برنامه‌ای بنویسید که N عدد را دریافت، تعداد اعداد مثبت، صفر و منفی را چاپ نماید.
۷. برنامه‌ای بنویسید که عدد صحیح و مثبت N را خوانده، فاکتوریل آن را محاسبه نماید.
۸. برنامه‌ای بنویسید که عدد N و M را دریافت، سپس اعداد بین آن‌ها را چاپ نماید.
۹. برنامه‌ای بنویسید که ۱۰ مضرب اول عدد N را چاپ نماید.
۱۰. برنامه‌ای بنویسید که ۲۰ جمله اول دنباله فیبوناچی را تولید نماید.
۱۱. برنامه‌ای بنویسید که سری‌های زیر را تا N (ورودی کاربر) محاسبه و نتیجه را چاپ نماید:

$S_1 = 1 + 2 + 3 + ... + N$ $S_2 = 1 - 2 + 3 - 4 + ... + N$

۱۲. برنامه‌ای بنویسید که N عدد را از ورودی خوانده، سپس تعداد اعدادی که بر پنج قابل‌قسمت هستند را در خروجی چاپ نماید.
۱۳. برنامه‌ای بنویسید که تعدادی عدد را دریافت، سپس اعدادی که رقم سمت راست آن‌ها پنج است را مشخص نماید. آخرین عدد صفر است.
۱٤. برنامه‌ای بنویسید که دو عدد را دریافت حاصل‌ضرب آن‌ها را با عمل جمع محاسبه گند.
۱۵. برنامه‌ای بنویسید که دو عدد را دریافت و آن‌ها را با عمل تفریق برهم تقسیم کند.
۱٦. برنامه‌ای بنویسید که عددی را دریافت، سپس تعداد ارقام آن را مشخص نمایید.
۱۷. برنامه‌ای بنویسید که عددی را دریافت، سپس وارون آن را محاسبه و چاپ نماید.
۱۸. برنامه‌ای بنویسید که جدول ضرب اعداد ۱ تا ۱۰ را چاپ نماید.
۱۹. برنامه‌ای بنویسید که عددی را دریافت، سپس مقسوم‌علیه آن را چاپ نماید.
۲۰. برنامه‌ای بنویسید که عددی را دریافت، سپس مشخص نماید عدد اول است یا خیر.

۲۱.برنامه‌ای بنویسید که عددی را دریافت، سپس مشخص نماید عدد کامل است یا خیر.
(راهنمایی: ٦ عدد کامل است زیرا 6 = 1 + 2 + 3)

۲۲.مربع هر عدد صحیح مثبت N را می‌توان به‌صورت مجموع N عدد فرد متوالی که از یک شروع می‌شود به دست آورد. به‌عنوان مثال: 6^2= 1 + 3 + 5 + 7 + 9 + 11

۲۳.برنامه‌ای بنویسید که یک عدد صحیح مثبت را خوانده طبق این روند، مربع آن را حساب کند و به همراه خود عدد در خروجی چاپ نماید.

۲٤.برنامه‌ای بنویسید که عددی را از ورودی خوانده مشخص نماید آیا عدد متقارن است یا خیر. نمونه اعداد متقارن 652256 و 151

۲۵.برنامه‌ای بنویسید که عدد N و M را دریافت و مقدار M^N را محاسبه و در خروجی چاپ نماید.

۲٦.برنامه‌ای بنویسید که آدرس یک متغیر رشته‌ای را دریافت و با کمک وقفه ۱٦ (10h) ماکروی PRINT را شبیه‌سازی نماید.

فصل ۹. رَویه‌ها[۱]

رویه‌ها قسمت‌هایی از کد هستند که می‌توان به‌منظور انجام کارهای خاصی در برنامه آن‌ها را فراخوانی کرد. رویه‌ها برنامه را ساخت‌یافته[۲] و درک برنامه را آسان‌تر می‌سازند. معمولاً رویه به همان‌جایی که ازآنجا فراخوانی شده برمی‌گردد. نحوه اعلان رویه‌ها به‌صورت زیر است:

```
name PROC

    کدهای مربوط به رَویه ;
    در این قسمت هستند. ;

RET
name ENDP
```

- **name**: نام رویه است. نام یکسانی باید در ابتدا و انتهای یک رویه نوشته می‌شود، زیرا ابتدا چک می‌شود تا مشخص شود چه رویه‌ای باید بسته شود. شاید بدانید که دستور RET برای برگشتن به سیستم‌عامل استفاده می‌شود. از همین دستور برای برگشت از رویه استفاده می‌شود (در حقیقت سیستم‌عامل برنامه شما را یک رویه می‌بیند)
- دستورات **PROC** و **END P** دستورات هدایتگر کامپایلر هستند. بنابراین آن‌ها به کد ماشینی واقعی ترجمه نمی‌شوند. کامپایلر تنها آدرس رویه را به خاطر می‌آورد.
- دستور **CALL** برای فراخوانی یک رویه استفاده می‌شود.

[۱] Procedures

[۲] Structural

به مثال زیر توجه کنید:

```
ORG 100h
CALL m1
MOV AX, 2
RET             ; برگشت به سیستم عامل

m1 PROC
    MOV BX, 5
    RET         ; برگشت به فراخوان
m1 ENDP

END
```

در مثال بالا ابتدا رویه m1 فراخوانی می‌شود و مقدار پنج را به ثبات BX انتقال می‌دهد و پس از بازگشت از رویه، به دستور بعد از فراخوانی بازمی‌گردد (MOV AX, 2).

چند روش برای انتقال پارامترها به رویه وجود دارد، آسان‌ترین راه انتقال پارامترها به رویه استفاده از ثبات‌ها است. در اینجا مثال دیگری از رویه‌ها وجود دارد که دو پارامتر را

```
ORG 100h

MOV AL, 1
MOV BL, 2

CALL m2         ; AX = 2
CALL m2         ; AX = 4
CALL m2         ; AX = 8
CALL m2         ; AX = 16 or 10h

RET             ; برگشت به سیستم عامل

m2 PROC
    MUL BL      ; AX = AL * BL.
    RET         ; برگشت به فراخوان
m2 ENDP

END
```

در ثبات‌های AL و BL قرار می‌دهد، رویه m2 این پارامترها را درهم ضـرب می‌کند و نتیجه را در ثبات AX برمی‌گرداند.

در مثال قبل مقدار ثبات AL در هر بار فراخوانی رویه به‌روز می‌شـود و ثبات BL بدون تغییر می‌ماند، بنـابراین ایـن الگوریتم ۲ به توان ٤ را محاسبه می‌کند و سرانجام نتیجه را در ثبـات AX قـرار می‌دهد (AX=10h=16).

در اینجـا مثـال دیگری وجود دارد کـه برای چـاپ پیغام"Word Hello" از یک رویه استفاده می‌کند.

```
ORG 100h
LEA SI, msg                 ; بارگذاری آدری msg در SI
CALL print_me
RET                         ; برگشت به سیستم عامل

; ================================================
; این روال رشته ختم شده به صفر را در خروجی چاپ می نماید
;  آدرس رشته باید در ثبات SI قرار گیرد.;

print_me PROC
next_char:
        CMP b.[SI], 0       ; در صورت صفر بودن بیت به stop خواهد رفت
        JE  stop

        MOV  AL, [SI]       ; کاراکتر اسکی بعدی را می گیرد
        MOV  AH, 0Eh        ; شماره زیربرنامه فرستادن کاراکتر به خروجی
        INT  10h            ; استفاده از وقفه ۱۶ (10h) جهت چاپ
        ADD  SI, 1          ; افزایش اندیس آرایه
        JMP  next_char      ; برگشت به عقب، به منظور چاپ کاراکتر بعدی

stop:
      RET                   ; برگشت به فراخوان
print_me   ENDP
; ================================================

msg   DB  'Hello World!', 0  ; رشته ای ختم شده به تهی
END
```

در این مثال پیشـوند "b." قبل از [SI] به این معنی است که به مقایسه بایت‌ها[۳] احتیاج داریم نه کلمه‌ها[٤].

وقتی نیاز بـه مقایسـه کلمه‌ها باشد به‌جای "b." از پیشوند"w." استفاده می‌کنیم. وقتی یکی از عملوندهای مورد مقایسه یـــک ثبـــات است، نیازی به پیشوند نیست زیرا کامپایلر اندازه هر ثبات را می‌داند.

۹-۱. تمرین فصل نهم

۱. رویه‌ای بنویسید که دو عدد a و b را از ورودی دریافت و a^b را محاسبه نماید.
۲. رویه‌ای بنویسـید که با کمک وقفه‌ها در محل X و Y از صـفحه خروجی، رشته‌ای که آدرس آن در SI قرار دارد را چاپ نماید.
۳. رویه‌ای بنویسید که طول رشته‌ای که آدرس آن در SI است را چاپ نماید.
٤. رویه‌ای بنویسید که رشته‌ای که آدرس آن در SI است را معکوس نماید.
۵. رویه‌ای بنویسـید که کارکترهای رشته‌ای که آدرس آن در SI قرارگرفته است را به حروف بزرگ انگلیسی تبدیل نماید.
٦. رویه‌ای بنویسـید که رشـته‌ای که آدرس آن در SI است را دریافت و تعداد کلمات آن را محاسبه نماید.
۷. رویه‌ای بنویسـید که سـاعت سـیستم را با استفاده از وقفه‌ها محاسبه و در قسمت بالای صفحه‌نمایش دهد و مکان‌نما را در ابتدای خط دوم قرار دهد.

---

[۳] Bytes
[٤] Words

فصل ۱۰. پشته[1]

پشته فضایی از حافظه اصلی (RAM) به‌منظور ذخیره موقت داده‌ها است. پشته جهت ذخیره آدرس برگشت روال از دستور CALL استفاده می‌کند و دستور RET نیز این مقدار (آدرس برگشت روال) را از پشته دریافت کرده و به آن آفست برمی‌گردد. دقیقاً همین مراحل توسط دستور INT زمانی که یک وقفه را فراخوانی می‌کند، رخ می‌دهد. این دستور ثبات وضعیت، سگمنت کد و آفست را در پشته ذخیره می‌کند. دستور IRET نیز برای بازگشت از یک فراخوانی وقفه استفاده می‌شود.

می‌توان از پشته برای ذخیره هر داده‌ای استفاده کرد. دو دستور وجود دارد که استفاده از پشته را امکان‌پذیر می‌کند:

PUSH: یک مقدار ۱۶ بیتی را در پشته ذخیره می‌کند.

POP: یک مقدار ۱۶ بیتی را از پشته برمی‌دارد.

نحوه‌ی نگارش دستور PUSH به شکل زیر است:

PUSH REG
PUSH SREG
PUSH memory
PUSH immediate

REG: AX, BX, CX, DX, DI, SI, BP, SP.
SREG: DS, ES, SS, CS.
memory: [BX], [BX+SI+7], 16 bit variable, etc...
immediate: 5, -24, 3Fh, 10001101b, etc...

[1] The Stack

نحوه‌ی نگارش دستور POP به شکل زیر است:

POP REG
POP SREG
POP memory

REG: AX, BX, CX, DX, DI, SI, BP, SP.
SREG: DS, ES, SS, (except CS).
memory: [BX], [BX+SI+7], 16 bit variable, etc...

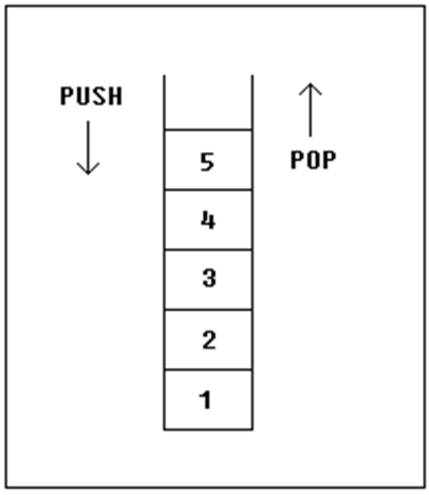
نکته:

- دستورات PUSH و POP فقط با مقادیر ١٦ بیتی کار می‌کنند.
- نگارش PUSH immediate فقط روی پردازنده ٨٠١٨٦ و بالاتر کار می‌کند.

پشته از الگوریتم LIFO[۲] (آخرین ورودی، اولین خروجی است) استفاده می‌کند. این بدان معنی است که اگر ما مقادیر زیر را یکی پس از دیگری در پشته PUSH کنیم:

1, 2, 3, 4, 5

اولین مقداری که از POP به دست می‌آید 5 خواهد بود و سپس به‌ترتیب مقادیر 4، 3، 2 درنهایت 1.



شکل ۱۰-۱) نحوه‌ی اجراشدن PUSH و POP در پشته

[۲] Last In First Out

برابر بودن تعداد PUSHها و POPها از اهمیت خاصی برخوردار است، در غیر این صورت پشته ممکن است خراب شود و برگشت به سیستم‌عامل غیرممکن شود. همان‌طور که می‌دانید دستور RET برای برگشت به سیستم‌عامل استفاده می‌شود، بنابراین زمانی که برنامه شروع به اجرا می‌کند آدرس برگشت به سیستم‌عامل (که معمولاً 0000h است) در پشته قرار می‌گیرد.

به این دلیل که ثبات‌های زیادی را در هنگام برنامه‌نویسی در دسترس نداریم دستورات PUSHو POP می‌توانند بسیار سودمند باشند؛ برای این کار می‌توان از شیوه‌ی زیر استفاده کرد:

- ذخیره مقدار اصلی ثبات در پشته (با استفاده از PUSH)
- استفاده از ثبات برای منظور دیگر
- بازیافت مقدار اصلی ثبات از پشته (با استفاده از POP)

به مثال زیر توجه کنید:

```
ORG   100h

MOV   AX, 1234h
PUSH  AX          ; مقدار AX را در پشته قرار می دهد

MOV   AX, 5678h   ; مقدار AX را تغییر می دهد

POP   AX          ; مقدار اولیه AX برگردانده می شود

RET

END
```

در ادامه مثال دیگری از کاربرد پشته آورده شده است که مقادیر دو ثبات را جابه‌جا می‌کند:

```
ORG   100h

MOV   AX, 1212h  ; AX = 1212h
MOV   BX, 3434h  ; BX = 3434h

PUSH  AX         ;  مقدار AX را در پشته قرار می دهد
PUSH  BX         ;  مقدار BX را در پشته قرار می دهد

POP   AX         ; AX = 3434h
POP   BX         ; BX = 1212h

RET
END
```

دلیل اینکه در مثال فوق جابجایی صورت خواهد گرفت، این است که پشته از الگوریتم LIFO استفاده می‌کند؛ در مثال بالا ابتدا مقدار 1212h و پس از آن 3434h در پشته PUSH می‌شوند، در نتیجه در زمان اجرا شدن دستور POP ابتدا مقدار 3434h و سپس مقدار 1212h از پشته خارج خواهد شد.

فضای پشته توسط ثبات SS (سگمنت پشته) و ثبات SP (اشاره‌گر پشته) مشخص می‌شود. معمولاً سیستم‌عامل در ابتدای شروع برنامه این ثبات‌ها را ارزش‌دهی می‌کند.

دستور PUSH source به‌صورت زیر عمل می‌کند:

- از ثبات SP مقدار ۲ را کم می‌کند.
- مقدار source را در آدرس SS:SP می‌نویسد.

دستور POP destination به‌صورت زیر عمل می‌کند:

- مقدار موجود در آدرس SS:SP را در destination می‌نویسد.
- ثبات SP را با ۲ جمع می‌کند.

آدرس جاری که توسط SS:SP به آن اشاره می‌شود، بالای پشته[۳] نامیده می‌شود. برای فایل‌های COM سگمنت پشته معمولاً همان سگمنت کد است و اشاره‌گر پشته باارزش 0FFFEh مقداردهی می‌شود. در آدرس SS:0FFFEh مقدار آدرس بازگشت برای دستور RET که آخرین دستور اجرایی در برنامه است ذخیره می‌شود.

می‌توان عملکرد پشته را با کلیک روی کلید[Stack] در پنجره شبیه‌ساز emu8086 مشاهده کنید. بالاترین عنصر پشته با علامت '>' مشخص‌شده است.

۱۰-۱. تمرین فصل دهم

۱. روالی بنویسید که کلیه مقادیر ثبات‌های عمومی را در پشته قرار داده و سپس با استفاده از روال دیگری مقادیر اولیه را جایگزین نماید.
۲. روالی بنویسد که با استفاده از پشته محتوی یک رشته را معکوس نماید.
۳. روالی بنویسید که رشته شامل یک عبارت پس‌ترتیب را دریافت و با استفاده از پشته مقدار آن عبارت را ارزشیابی نماید. (2 ➔ / * 2 2 + 5 3)

[۳] the top of the stack

فصل ۱۱. ماکروها[1]

ماکروها مشابه رویه‌ها هستند، اما نه کاملاً. ماکروها تا زمانی که برنامه کامپایل نشده شبیه به رویه‌ها به نظر می‌رسند، اما زمانی که کد برنامه کامپایل می‌شود، ماکروها با کدهای واقعی جایگزین می‌شوند. اگر یک ماکرو در برنامه اعلان شود ولی از آن در کد برنامه استفاده نشده باشد، کامپایلر به‌راحتی آن را نادیده می‌گیرد. یک مثال خوب از چگونگی استفاده از ماکروها فایل کتابخانه‌ای emu8086.inc است. این فایل شامل چندین ماکرو است که برنامه‌نویسی را برای شما راحت می‌کند. روش اعلان ماکرو به‌صورت زیر است:

```
name   MACRO  [parameters,...]
       ; کدهای مربوط به ماکرو در این قسمت نوشته می‌شوند
ENDM
```

برخلاف رویه‌ها، ماکروها باید در قسمت بالایی کدهای برنامه اصلی اعلان شوند. در مثال زیر ماکروی MyMacro عمل مقداردهی به سه ثبات را انجام می‌دهد:

```
MyMacro   MACRO  p1, p2, p3
          MOV AX, p1
          MOV BX, p2
          MOV CX, p3
ENDM
ORG 100h
MyMacro 1, 2, 3
MyMacro 4, 5, DX
RET
```

[1] Macros

کد مثال قبل به‌صورت زیر بسط داده می‌شود:

```
MOV AX, 00001h
MOV BX, 00002h
MOV CX, 00003h
MOV AX, 00004h
MOV BX, 00005h
MOV CX, DX
```

۱۱-۱. برخی نکات مهم در مورد ماکروها و رویه‌ها[۲]

- زمانی که از یک رویه استفاده می‌کنید، بایستی از دستور CALL برای فراخوانی آن استفاده کنید. برای مثال:

```
CALL MyProc
```

- زمانی که از یک ماکرو استفاده می‌کنید، تنها کافی است نام آن را برای فراخوانی تایپ کنید. به‌عنوان مثال:

```
MyMacro
```

- رویه در آدرس خاصی از حافظه قرار می‌گیرد و اگر ۱۰۰ بار از آن رویه در کد برنامه استفاده شود، CPU کنترل را به همان بخش از حافظه منتقل خواهد کرد و کنترل با دستور RET به برنامه برخواهد گشت. از پشته برای ذخیره آدرس برگشت استفاده می‌شود. دستور CALL حدود سه بایت فضا را به خود اختصاص می‌دهد، بنابراین اندازه فایل خروجی اجرایی بسیار ناچیز رشد می‌کند.
- ماکروها مستقیماً در کد برنامه بسط[۳] داده می‌شوند؛ بنابراین اگر از ماکروای در کد برنامه۱۰۰ بار استفاده کنید، کامپایلر ماکرو را ۱۰۰ مرتبه بسط می‌دهد، که این عمل باعث می‌شود هر بار که دستورات ماکرو در کد برنامه درج شوند، فایل خروجی اجرایی بزرگ و بزرگ‌تر شود.
- باید از پشته یا ثبات‌های همه‌منظوره برای ارسال پارامترها به رویه استفاده شود.

---

[۲] Procedures

[۳] Expanded

- برای ارســـال پارامترها به ماکرو، می‌توان آن‌ها را بعد از اســـم ماکرو تایپ کرد. برای مثال: MyMacro 1, 2, 3
- برای پایان دادن به یک ماکرو تنها کافی است از دستور ENDM استفاده نمایید.
- برای پایان دادن به یک رویه باید نام رویه قبل از ENDP درج شود.
- ماکروها مســتقیماً در کد برنامه بســط داده می‌شوند. بنابراین اگر برچسب‌هایی در کنار اعلان ماکرو وجود داشته باشد، زمانی که ماکرو چند بار در کد برنامه درج شود، ممکن اســت خطای «اعلامیه‌های تکراری[۴]» دریافت شــود. برای جلوگیری از این مشــکل، از دســتور LOCAL قبل از نام متغیرها، برچســب‌ها یا رویه استفاده کنید. به مثال زیر توجه نمایید:

```
MyMacro2    MACRO
            LOCAL label1, label2
            CMP  AX, 2
            JE label1
           CMP  AX, 3
           JE label2
           label1:
                   INC  AX
           label2:
                   ADD  AX, 2
ENDM
ORG 100h
MyMacro2
MyMacro2
RET
```

نکته: اگر قصد دارید از ماکروهایی که نوشته‌اید در چندین برنامه استفاده نمایید، ایده خوبی است اگر تمامی آن‌ها را در یک فایل جدا قرار دهید؛ سپس آن فایل را در پوشه inc قرار دهید و از دستور INCLUDE file-name برای دسترسی به ماکروها استفاده کنید.

[۴] Declaration Duplicate

۱۱-۲. تمرین فصل یازدهم

۱. برنامه‌ای بنویسید که با استفاده از یک ماکرو بزرگ‌ترین عدد از بین سه عدد را مشخص نماید.

۲. ماکروی بنویسید که دو عدد را به‌صورت پارامتر دریافت و پارامتر اول را به توان پارامتر دوم برساند.

۳. ماکروی بنویسید که ماکروی PUTC را شبیه‌سازی نموده و امکان ورود محل نمایش کاراکتر را فراهم آورد.

٤. ماکروی بنویسید که یک کاراکتر را به‌عنوان پارامتر دریافت و سپس آن کاراکتر را در صفحه‌نمایش از سمت چپ به راست و از بالا به پایین حرکت دهد.