U

فصل هفتم:

فرآینـــدهای تکـــرار و

دستورات شرطی

هدف‌گذاری این فصل:

- آشنایی با مفهوم پرش به نقطه‌ای از برنامه
- تفاوت بین پرش‌های شرطی و غیر شرطی
- آشنایی با فرآیندهای تکرار
- آشنایی با دستورات مقایسه‌ای

۷-۱- مقدمه:

در این فصل با مباحث مهمی از جمله دستورات شرطی و عملیات تکراری در یک برنامه آشنا خواهیم شد. در اغلب برنامه‌های زبان اسمبلی از این ساختارها، برای مقاصد برنامه‌نویسی استفاده می‌شود. تمام برنامه‌هایی که تاکنون مورد بحث قرار گرفتند، به ترتیب و به صورت متوالی اجرا می‌شدند تا به پایان برسند؛ به این شیوه ساختار خطی گفته می‌شود. دستورات پرش، این امکان را به برنامه‌نویس می‌دهند تا بتواند موقعیت اجرای یک برنامه را از محلی به محل دیگری از برنامه، تغییر دهد. در حقیقت دستورات پرش و فرآیندهای تکرار، با تغییر مقادیر ثبات‌های IP و CS این تغییر را انجام می‌دهند. همانطور که در فصل ۲ توضیح داده شد، آدرس آفست خطوط برنامه در سگمنت کد، در ثبات IP نگهداری و آدرس شروع سگمنت کد نیز در ثبات CS ذخیره می‌شود.

دستورات پرش، معادل دستور goto در زبان‌های سطح بالا می‌باشند. گاهی اوقات براساس شرایط برنامه، نیاز داریم تا به نقطه خاصی از برنامه انشعاب نماییم.

دستورات تکرار، معادل حلقه‌های تکرار از قبیل While، For و Do در زبان‌های سطح بالا می‌باشند. اغلب برنامه‌ها از این ساختارها استفاده می‌نمایند.

۷-۲- دستورات پرش:

دستورات پرش به طور کلی به دو دسته تقسیم می‌شوند: دستورات پرش بدون شرط و دستورات پرش با شرط.

۱- دستورات پرش بدون شرط:

دستور پرش بدون شرط در زبان اسمبلی JMP[1] نام دارد. شکل کلی این دستور به صورت زیر می‌باشد:

```
JMP     Label
```

---

[1] Jump

Label، نام برچسب محلی است که مقصد پرش می‌باشد. این برچسب قبلاً بایستی در برنامـه مشخص شده باشد. پرش بدون شرط به سه صورت انجام می‌شود:

الف) پرش به صورت Short: هرگاه بعد از دستور JMP از کلمه Short اسـتفاده شـود، آنگـاه آدرس مقصد پرش حداکثر می‌تواند ۱۲۷+ بایت پایین‌تر یا ۱۲۸– بایت بالاتر نسبت به دستور بعـد از JMP باشد. به عبارت دیگر آدرس هدف، یک بایتی است. پردازنده، این آدرس را بـه عنـوان آفسـت آدرس در نظر می‌گیرد و به مقدار اشاره‌گر دستور IP اضافه می‌کند تا پرش به مقصد مورد نظر انجـام شود. فرم استفاده از این دستور به شکل زیر می‌باشد:

```
JMP   SHORT   Label
```

ب) پرش به صورت Near: هرگاه بعد از دستور JMP از کلمه Near استفاده شود، آنگاه آدرس مقصد پرش حداکثر می‌تواند ۳۲۷۶۷+ بایت پایین‌تر یا ۳۲۷۶۸– بایت بالاتر نسبت به دسـتور بعـد از JMP باشد. به عبارت دیگر آدرس هدف، دو بایتی است. فرم استفاده از ایـن دسـتور بـه شـکل زیـر می‌باشد:

```
JMP   NEAR   Label
```

ب) پرش به صورت Far: اگر قصد داشته باشیم که از سـگمنت جـاری بـه سـگمنت دیگـری انشعاب انجام دهیم، از کلمه Far بعد از دستور JMP استفاده می‌کنیم. در این روش عموماً، پـرش از سگمنت کد به محلی در سگمنت داده انجام می‌شود. فرم استفاده از این دستور به شکل زیر می‌باشد:

```
JMP   FAR PTR   TABEL
```

محل پرش در داخل سگمنتی که متغیر TABEL در آنجا قرار دارد، مشخص می‌شود و دو بایت از آدرس TABEL و TABEL+1 در ثبات IP برای انجام عمل پرش، قرار می‌گیـرد. در حقیقـت بـرای انجام پرش از محل فعلی که آدرس CS:IP می‌باشد به محلی که قرار است پرش انجام شـود، CS:IP بعدی، میزان اختلاف این دو محل پرش را تعیین می‌کند.

مثال ۷-۱:

```
        JMP     Target
        ...
Target: ADD     AX, BX
```

نکته ۱: در صورتی که یک برچسب، قبل از دستورات زبان اسمبلی قرار گیرد، باید با : همراه باشـد. برچسب‌هایی که قبل از شبه دستورات قرار می‌گیرند فاقد : می‌باشند.

نکته ۲: اگر آدرس مقصد انشعاب را داشته باشیم، می‌توانیم به طور مستقیم بـه آن محـل انشـعاب کنیم. به عنوان مثال JMP 1000 یعنی انشعاب به آدرس 1000 برنامه.

نکته ۳: دستور JMP بر روی بیت‌های ثبات پرچم تأثیری نمی‌گذارد.

مثال ۷-۲:

```
Target1:  JMP   Target2
Target2:  JMP   Target1
```

در خط اول برنامه به محل Target2 انشعاب انجام می‌شود و در خط دوم برنامه دوباره به Target1 انشعاب می‌شود. در این حالت این عمل بدون توقف ادامه می‌یابد و یا بعد از مدتی پیغام خطـا ایجـاد می‌شود.

۲- دستورات پرش شرطی:

اغلب دستورات پرش با محقق شدن یک شرط انجام می‌شود. به عنوان مثال اگر شمارنده یک حلقه برابر صفر باشد، یا بخواهیم از حلقه‌ای خارج شویم و یا به محل خاصی انشعاب شود.

شرایط دستورات پرش از روی وضعیت بیت‌های ثبات پرچم مورد بررسی قـرار مـی‌گیرنـد. بـه عنوان مثال هرگاه نتیجه عملی منجر به پاسخ صفر شود، پرچم Z برابر یک خواهد بود. با کنترل این پرچم می‌توان انشعابی انجام داد که با صفر شدن نتیجه یک عمل ارتباط داشته باشد.

به طور کلی در دستورات پرش با شرط، ابتدا دستوری اجرا می‌شود که وضعیت بیت مورد نظـر از ثبات پرچم را براساس منطق برنامه، تغییر دهد؛ آنگاه با محقق شدن شرط، انشعاب به طور مشروط به محل مورد نظر انجام می‌شود. دستورات پرش شرطی دارای انواع متعددی هستند که شـکل کلـی این دستورات به صورت زیر می‌باشد:

JXXX Label یا JXX Label یا JX Label

که در آن X بیانگر یک حرف مرتبط با نوع پرش مـی‌باشـد. نـوع پـرش نیـز براسـاس تغییـر بیت‌های ثبات پرچم انجام می‌شود.

مثال ۷-۳:

```
MOV    AL, 0D8H
MOV    BL, 0AFH
SUB    AL, BL
JC     Target
```

در این مثال، ثبات AL منهای ثبات BL می‌شود. اگر این تفریق منجر به ایجاد رقم نقلی شود، بیت C ثبات پرچم برابر یک شده، بنابراین شرط دستور JC[1] (پرش در صورت یک بودن بیـت نقلـی) برقـرار خواهد بود که در این صورت پرش به برچسب Target انجام خواهد شـد. اگـر شـرط برقـرار نباشـد (یعنی C=0) دستور بعد از دستور JC اجرا خواهد شد.

مثال ۷-٤:

```
MOV    AL, -1
MOV    BL, 1
ADD    AL, BL
JNZ    Target
```

در این مثال، ابتدا اعداد 1- و 1 در ثبات‌ها AL و BL بارگذاری شده و سپس این دو ثبات با هم جمـع می‌شوند. بدیهی است که حاصل جمع این دو ثبات برابر صفر خواهد بود؛ پـس پـرچم Z برابـر یـک خواهد شد. در دستور JNZ[2] (پرش در صورت صفر بودن بیت پرچم Z) در صورتی بـه آدرس Target انشعاب می‌شود که Z=0 باشد. با توجه به اینکه در این مثال دستور ADD AL, BL سبب شده اسـت تا بیت Z=1 شود، بنابراین شرط انشعاب برقرار نبوده و پرش به آدرس Target انجـام نمـی‌شـود. در این صورت دستور بعد از JNZ اجرا خواهد شد.

دستور مقایسه CMP[3]:

این دستور دو عملوند خود را با هم مقایسه می‌کند. شکل این دستور به صورت زیر می‌باشد:

```
CMP    Operand1, Operand2
```

---

[1] Jump Carry (JC)
[2] Jump Not Zero (JNZ)
[3] Compare

نحوه مقایسه به این صورت است که عملوند اول منهای عملوند دوم می‌شود. باید توجه داشته باشید که در مقدار عملوندها تغییری ایجاد نخواهد شد؛ اما براساس تفریق انجام شده، تأثیر خود را روی بیت‌های O، S، A، Z، C و D از ثبات پرچم، خواهد گذاشت.

عملوند اول این دستور می‌تواند یک ثبات و یا یک متغیر حافظه‌ای باشد. عملوند دوم می‌تواند یک ثبات، یک متغیر حافظه‌ای و یا یک عدد ثابت باشد. همچنین هر دو عملوند نمی‌توانند به صورت توأم حافظه‌ای باشند.

دستور مقایسه می‌تواند ۳ حالت داشته باشد: در حالت اول، عملوند اول بزرگتر از عملوند دوم می‌باشد؛ در حالت دوم، عملوند اول مساوی عملوند دوم می‌باشد و بالاخره در حالت سوم، عملوند اول کوچکتر از عملوند دوم می‌باشد. براساس هر یک از این سه حالت بیت‌های C و Z از ثبات پرچم به صورت زیر تغییر می‌کنند.

عملوند دوم > عملوند اول ⟵ C = 0 و Z = 0
عملوند دوم = عملوند اول ⟵ C = 0 و Z = 1
عملوند دوم < عملوند اول ⟵ C = 1 و Z = 0

از دستور CMP معمولاً قبل از دستورات انشعاب استفاده می‌شود. حال پس از دستور CMP و براساس تغییر بیت‌های C و Z، انشعاب انجام شده و یا اینکه به علت برقرار نبودن شرط انشعاب، دستور بعدی اجرا خواهد شد.

به طور کلی دستورات انشعاب مبتنی بر بیت‌های ثبات پرچم، اعداد علامت‌دار و اعداد بدون علامت می‌باشند. جدول شماره (۱) حاوی دستورات انشعاب براساس بیت‌های پرچم می‌باشد. مبنای انشعاب براساس این جدول، با توجه به وضعیت بیت‌های ثبات پرچم خواهد بود.

جدول شماره (۱): انشعاب براساس بیت‌های ثبات پرچم

دستور	توضیح	شرط انشعاب
JC	Jump Carry	C = 1
JNC	Jump Not Carry	C = 0
JS	Jump Sign	S = 1
JNS	Jump Not Sign	S = 0
JO	Jump Overflow	O = 1
JNO	Jump Not Overflow	O = 0
JP	Jump Parity	P = 1
JNP	Jump Not Parity	P = 0

مثال ۷-۵:

```
MOV   AL, 0D8H          11111
MOV   BL, 0ACH    AL = 1101 1000
ADD   AL, BL      BL = 1010 1100
JC    Target        11000 0100
                C=1 و S=1 ، Z=0 ، P=1 ، A=1
```

با توجه به اینکه در این مثال Carry=1 شده است، پس شـرط انشـعاب برقـرار مـی‌باشـد؛ بنـابراین انشعاب به آدرس Target انجام می‌شود.

در این مثال اگر به جای دستور JC، از دستور JNP استفاده می‌کردیم، با توجه به اینکه P=1 می‌باشد، بنابراین شرط JNP برقرار نبوده و انشعاب به آدرس Target انجام نمی‌گیرد.

انشعاب می‌تواند براساس مقایسه اعداد علامت‌دار و بدون علامت نیز انجام شود. جداول شماره (۲) و (۳) حاوی این دستورات می‌باشد.

جدول شماره (۲): دستورات انشعاب براساس اعداد علامت‌دار

شرط انشعاب	دستور
عملوند دوم = عملوند اول	JZ یا JE
عملوند دوم ≠ عملوند اول	JNZ یا JNE
عملوند دوم > عملوند اول	JNLE یا JG
عملوند دوم =< عملوند اول	JNL یا JGE
عملوند دوم < عملوند اول	JNGE یا JL
عملوند دوم <= عملوند اول	JNG یا JLE

جدول شماره (۳): دستورات انشعاب براساس اعداد بدون علامت

شرط انشعاب	دستور
عملوند دوم = عملوند اول	JZ یا JE
عملوند دوم > عملوند اول	JNBE یا JA
عملوند دوم < عملوند اول	JNAE یا JB
عملوند دوم ≠ عملوند اول	JNZ یا JNE
عملوند دوم =< عملوند اول	JNB یا JAE
عملوند دوم <= عملوند اول	JNA یا JBE

حروف اختصاری به کار رفته در این جداول، در جدول شماره (۴) آورده شده است.

جدول شماره (۴): حروف اختصاری به کار رفته در جداول شماره ۲ و ۳

حرف	N	E	G	L	A	B
معنی	Not	Equal	Greater	Less	Above	Below

نکته ٤: تمام دستورات پرش شرطی، Short می‌باشند.

مثال ۷-٦:

```
MOV    AL, 0D8H
MOV    BL, 0E7H
CMP    AL, BL
JNA    Target
```

در این مثال ثبات AL با BL به صورت بدون علامت مقایسه می‌شود. با توجه به اینکه مقدار ثبات AL کوچکتر از ثبات BL می‌باشد؛ بنابراین شرط دستور JNA برقرار می‌باشد و انشعاب بـه آدرس Target انجام می‌شود.

نکته ۵: دستور CMP همیشه عملوندهای خود را به صورت بدون علامت در نظر می‌گیرد.

مثال ۷-۷: قطعه برنامه‌ای بنویسید که حاصل جمع یک آرایه ۵ عضـوی از نـوع بایـت را محاسـبه نموده و حاصل را در حافظه ذخیره کند.

```
1.          X       DB      5, 2, 4, 8, 6           ; سگمنت داده
2.          SUM     DB      ?
; ---------------------------------------
3.                  LEA     SI, X                   ; سگمنت کد
4.                  MOV     CL, 5
5.                  MOV     AL, 0
6.          BACK:   ADD     AL, [SI]
7.                  INC     SI
8.                  DEC     CL
9.                  JNZ     BACK
10.                 MOV     SUM, AL
```

قطعه برنامه شماره (۱): محاسبه حاصل جمع ۵ عضو یک آرایه

۱- پنج بایت متغیر حافظه‌ای با آدرس شـروع X بـا مقـادیر اولیـه 5، 2، 4، 8 و 6 در سـگمنت داده، تعریف می‌شود.

۲- متغیر حافظه‌ای SUM برای نگهداری حاصل جمع ۵ عدد واقع در آدرس X به بعد تعریف می‌شود.

۳- آدرس شروع X در ثبات SI ذخیره می‌شود.

۴- شمارنده CL برابر ۵ می‌شود؛ زیرا قرار است حلقه ۵ بار تکرار شود.

۵- ثبات AL مقدار اولیه صفر می‌گیرد. این ثبات قرار است حاصل جمع ۵ عدد فوق را در خود نگهداری نماید؛ بنابراین مقدار اولیه آن را صفر می‌کنیم.

۶- اولین عدد یعنی عدد ۵ با ثبات AL که مقدار صفر دارد، جمع می‌شود.

۷- ثبات SI که آدرس X را در خود نگهداری می‌کند، یک واحد افزایش می‌یابد تا در تکرار دوم حلقه، عدد بعدی را از حافظه دریافت کند.

۸- شمارنده را یک واحد کاهش می‌دهیم؛ تا زمانی که شمارنده صفر شود حلقه ادامه خواهد یافت.

۹- صفر بودن شمارنده بررسی می‌شود. اگر شمارنده CL صفر نباشد، حلقه ادامه می‌یابد.

۱۰- در صورتی که شمارنده صفر شد و از حلقه خارج شدیم، ثبات AL که حاوی جمع ۵ عدد می‌باشد را در متغیر حافظه‌ای SUM ذخیره می‌کنیم.

مثال ۷-۸: قطعه برنامه‌ای بنویسید که بزرگترین عدد بدون علامت را از میان یک لیست ۱۰ عضوی محاسبه نماید.

```
1. NUMBERS   DB     5, 0, 4, ...          ; سگمنت داده
2.       MAX DB     ?
; ---------------------------------------
3.           LEA    SI, NUMBERS           ; سگمنت کد
4.           MOV    CX, 9
5.           MOV    AL, NUMBERS
6.     AGAIN: INC    SI
7.           CMP    AL, [SI]
8.           JAE    SKIP
9.           XCHG   AL, [SI]
10.      SKIP: DEC    CX
11.          JNZ    AGAIN
12.          MOV    MAX, AL
```

قطعه برنامه شماره (۲): یافتن بزرگترین عدد بدون علامت از بین ده عدد موجود در حافظه

۱- ده بایت اطلاعات از آدرس NUMBERS به بعد در حافظه ذخیره می‌شود.

۲- متغیر MAX برای نگهداری بزرگترین عدد آرایه تعریف می‌شود.

۳- آدرس شروع اعداد در ثبات SI ذخیره می‌شود.

۴- شمارنده برنامه با عدد ۹ بارگذاری می‌شود. با توجه به اینکه در مرحله اول ۲ عدد مقایسه می‌شوند، بنابراین به ۹ بار تکرار حلقه نیاز خواهیم داشت.

۵- عدد اول وارد ثبات AL می‌شود.

۶- ثبات SI که آدرس متغیرها را در خود نگهداری می‌کند، را یک واحد افزایش می‌دهیم تا به عنصر بعدی آرایه اشاره کند.

۷- محتوای SI که در واقع عدد دوم آرایه است با ثبات AL که عدد اول آرایه را در خود نگهداری می‌کند، مقایسه می‌شود.

۸- اگر عدد اول که در ثبات AL قرار دارد، بزرگتر مساوی عدد دوم باشد، به برچسب SKIP انشعاب می‌کند تا اعداد بعدی را بررسی نماید.

۹- اگر عدد اول کوچکتر از عدد دوم باشد، با دستور XCHG دو عدد را با هم جابجا کرده تا ثبات AL همواره بزرگترین عدد را در خود نگهداری نماید.

۱۰- شمارنده CX را یک واحد کاهش می‌دهیم تا تکرار بعدی شروع شود.

۱۱- اگر شمارنده CX صفر نشده باشد، به آدرس AGAIN انشعاب می‌کند تا عدد بعدی دریافت شود.

۱۲- در صورتی که حلقه ۹ بار تکرار شود و تمام اعداد برای مقایسه بررسی شوند؛ محتوای ثبات AL که بزرگترین عدد آرایه را در خود نگهداری می‌کند را به متغیر حافظه‌ای MAX منتقل می‌کنیم.

۷-۳- دستورات تکرار:

۱- دستور LOOP:

همانطور که در مثال‌های قبل ملاحظه شد، برای اطمینان از تعداد تکرارهای صحیح برنامه، از یک شمارنده استفاده می‌کردیم و در هر بار اجرای حلقه، شمارنده را یک واحد کاهش داده و سپس مقدار شمارنده با عدد صفر مقایسه می‌شد و در صورت صفر بودن آن، حلقه پایان می‌پذیرفت.

دستور LOOP، با هر بار تکرار، به طور خودکار از ثبات CX به عنوان شمارنده، یـک واحـد کـم می‌کند و تا زمانی که این ثبات صفر نشده، حلقه تکرار می‌شود. قالب کلی این دستور بـه شـکل زیـر می‌باشد:

```
LOOP    Label
```

نکاتی که در دستور LOOP باید در نظر گرفت، عبارتند از:

۱- تعداد شمارش حلقه را در ثبات CX قرار می‌دهیم. حلقه برنامه به تعداد عـدد واقـع در ثبـات CX تکرار می‌شود.

۲- عمل کنترل صفر شدن شمارنده برنامه، توسط دستور LOOP به طور خودکار انجام می‌شود.

مثال ۷-۹: قطعه برنامه‌ای زیر عدد ۷ را ۵ بار با خودش جمع می‌کند که همان ضـرب عـدد ۷ در ۵ می‌باشد.

```
1.      RESULT  DB      ?                       ; سگمنت داده
; ---------------------------------------
2.              MOV     AL, 0                   ; سگمنت کد
3.              MOV     CX, 5
4.              MOV     BL, 07
5.      AGAIN:  ADD     AL, BL
6.              LOOP    AGAIN
7.              MOV     RESULT, AL
```

قطعه برنامه شماره (۳): محاسبه حاصل ضرب دو عدد بدون استفاده از دستور ضرب

۱- متغیر حافظه‌ای RESULT برای نگهداری حاصلضرب تعریف شده است.

۲- ثبات AL که قرار است حاصلضرب دو عدد را در خود نگه دارد، مقدار صفر می‌گیرد.

۳- شمارنده دستور LOOP با عدد 5 بارگذاری می‌شود.

۴- ثبات BL با مقدار 7 بارگذاری می‌شود.

۵- اولین تکرار حلقه LOOP برای ۵ بار جمع عدد ۷ آغاز می‌شود.

۶- دستور LOOP باعث تکرار ۵ بار حلقه می‌شود.

۷- پس از خروج از حلقه، ثبات AL که حاوی حاصلضرب دو عدد است، در متغیر حافظه‌ای RESULT ذخیره می‌شود.

مثال ۷-۱۰: قطعه برنامه سطح بالای زیر را به زبان اسمبلی بنویسید.

```
For  i=1  To  100
        S=S+i
```

```
1.              MOV   AX, 0
2.              MOV   CX, 100
3.    AGAIN:    ADD   AX, CX
4.              LOOP  AGAIN
5.              MOV   S, AX
```

قطعه برنامه شماره (۴): تبدیل دستور For برنامه‌های سطح بالا به زبان اسمبلی

ثبات CX از مقدار ۱۰۰ تا صفر ادامه می‌یابد؛ بنابراین هم به عنوان شمارنده دستور LOOP و هم به عنوان حاصل جمع اعداد یک تا ۱۰۰ می‌تواند مورد استفاده قرار گیرد.

نکته ٦: بیشترین تکرار در دستور LOOP هنگامی است که مقدار ثبات CX برابر صفر باشد؛ زیرا دستور LOOP ابتدا یک واحد از ثبات CX کم می‌نماید و در صورت مخالف بودن آن با صفر، حلقه LOOP ادامه پیدا می‌کند. اگر ثبات CX حاوی عدد صفر باشد، کاستن یک واحد از آن، ثبات CX را برابر 1- یا FFFF خواهد کرد که این مقدار بیشترین مقداری است که یک ثبات ۱۶ بیتی می‌تواند داشته باشد.

۲- دستور LOOPZ و LOOPE:

این دستورات که کاملاً مثل هم هستند، مشابه دستور LOOP عمل می‌کنند؛ با این تفاوت که شرط خاتمه حلقه با یکی از دو روش زیر محقق می‌شود:

۱- صفر شدن ثبات شمارنده CX

۲- صفر شدن بیت ثبات پرچم Z

در صورتی که هر کدام از دو شرط فوق زودتر محقق شود، حلقه LOOP خاتمه می‌یابد. دستورات LOOPZ و LOOPE از نظر عملکرد مشابه هم هستند. در مواقعی که صفر شدن نتیجه مدنظر باشد، دستور LOOPZ به خوانایی برنامه می‌افزاید و هرگاه تساوی دو عدد مورد بررسی قرار گیرد از دستور LOOPE استفاده می‌شود.

۳- دستور LOOPNZ و LOOPNE:

این دستورات که کاملاً مشابه هم هستند، همانند دستور LOOP عمل می‌کنند؛ با ایـن تفـاوت که شرط خاتمه حلقه با یکی از دو روش زیر محقق می‌شود:

۱- صفر شدن ثبات شمارنده CX

۲- یک شدن بیت ثبات پرچم Z

در صورتی که هر کدام از دو شرط فوق زودتـر محقـق شـود، حلقـه LOOP خاتمـه مـی‌یابـد. دستورات LOOPNZ و LOOPNE از نظر عملکرد مشابه هم هستند.

٤- دستور JCXZ:

این دستور، یک دستور تکرار است و در صورتی که مقدار CX≠0 باشد، حلقه خاتمه می‌یابد.

مثال ۷-۱۱: بررسی نمایید که آیا تمام عناصر یک آرایه ۲۰ عضوی باعدد صفر پر شده است یاخیر؟

```
1.          LEA     SI, ARRAY       ; درج آدرس آرایه در ثبات SI
2.          SUB     SI, 1           ; کاهش یک واحدی آدرس آرایه
3.          MOV     CX, 20          ; تعیین شمارنده برنامه
4.  REPEAT: ADD     SI, 1           ; افزایش یک واحدی آدرس آرایه
5.          CMP     [SI], 00        ; مقایسه عنصر آرایه با عدد صفر
6.          LOOPE   REPEAT          ; تکرار حلقه در صورت برقراری شرط
```

قطعه برنامه شماره (۵): بررسی صفر بودن عناصر یک آرایه ۲۰ عنصری

نکته ۷: دستورات فراخوانی روال‌ها، پروسه‌ها، توابع و حتی وقفه‌ها نیز نوعی مکانیزم پـرش هسـتند که با تغییر مقادیر IP و CS در داخل سگمنت برنامه، عمل پرش را انجام داده و پس از پایـان، قطعـه برنامه مربوط به عمل بازگشت به محل فراخوانی شده، اجرا می‌شود*.

در ادامه به چند مثال دیگر می‌پردازیم تا با کاربرد دستورات مطرح شده در ایـن فصـل بیشـتر آشنا شویم.

---

* این بحث در فصول مربوط به روال‌ها و وقفه‌ها به طور کامل تشریح خواهد شد.

مثال ۷-۱۲: برنامه‌ای بنویسید که حاصل جمع دو آرایه ۱۰۰ عنصری X و Y از نوع بایت را محاسبه کرده و نتیجه را در آرایه X قرار دهد.

```
1.          X       DB      1, 50, 25, ...          ; تعریف آرایه X
2.          Y       DB      5, 3, 20, ...           ; تعریف آرایه Y
; ------------------------------------------------------
3.                  LEA     SI, X                   ; ذخیره آدرس X در ثبات SI
4.                  LEA     DI, Y                   ; ذخیره آدرس Y در ثبات DI
5.                  MOV     CX, 100                 ; تعیین مقدار شمارنده
6.          REPEAT: MOV     AL, [SI]                ; انتقال عنصری از آرایه اول به ثبات AL
7.                  ADD     AL, [DI]                ; جمع عنصری از آرایه Y با ثبات AL
8.                  MOV     [SI], AL                ; ذخیره جمع دوعضو آرایه درمکان آرایه X
9.                  INC     SI                      ; افزایش آدرس آرایه X
10.                 INC     DI                      ; افزایش آدرس آرایه Y
11.                 LOOP    REPEAT                  ; تکرار حلقه به تعداد ۱۰۰ بار
```

قطعه برنامه شماره (۶): محاسبه حاصل جمع دو آرایه ۱۰۰ عنصری

مثال ۷-۱۳: برنامه‌ای بنویسید که ۱۰ عدد ابتدایی سری فیبوناچی را ایجاد نماید.

حل: هر عدد از سری فیبوناچی از مجموع دو عدد قبلی ایجاد می‌شود، عدد اول و دوم سری فیبوناچی، 1 می‌باشد؛ بنابراین دنباله این سری به صورت زیر است:

1, 1, 2, 3, 5, 8, ...

```
1.                  MOV     AX, 00                  ; بارگذاری صفر در ثبات AX
2.                  MOV     BX, 1                   ; قرار دادن اولین عدد سری در ثبات BX
3.                  MOV     CX, 10                  ; مقداردهی شمارنده برای محاسبه ۱۰ عدد اول سری
4.                  MOV     DX, 00                  ; بارگذاری صفر در ثبات DX
5.          NEXT:   ADD     AX, BX                  ; جمع دو عدد سری فیبوناچی
6.                  MOV     BX, DX                  ; انتقال جملات سری فیبوناچی-
7.                  MOV     DX, AX                  ; به مرحله بعد
8.                  LOOP    NEXT                    ; تکرار حلقه به اندازه شمارنده CX
```

قطعه برنامه شماره (۷): محاسبه ۱۰ عدد اول سری فیبوناچی

مثال ۷-۱٤: قطعه برنامه‌ای بنویسید که در یک آرایه ۲۰ عنصری به نام NUMBERS، بررسی نماید که آیا عنصری با مقدار صفر وجود دارد یا خیر؟ برنامه به محض یافتن عدد صفر متوقف شده و نیازی به جست‌وجوی سایر عناصر آرایه نمی‌باشد.

```
1. NUMBERS    DB      20   DUP (?)          ; سگمنت داده
; ----------------------------------------
2.            LEA     BX, NUMBERS          ; سگمنت کد
3.            DEC     BX
4.            MOV     CX, 20
5.      NEXT: INC     BX
6.            CMP     [BX], 00
7.            LOOPNE  NEXT
8.            MOV     AX, 4C00H
9.            INT     21H
```

قطعه برنامه شماره (۸): یافتن مقدار صفر در یک آرایه ۲۰ عنصری

۷-٤- خلاصه:

در این فصل با دستورات تکرار، شـرطی و مقایسـه آشـنا شـدیم و کـاربرد ایـن دسـتورات در برنامه‌های متعدد، مورد بررسی قرار گرفت. در اغلب برنامـه‌هـا، سـاختارهای شـرطی و تکـرار مـورد استفاده قرار می‌گیرند. این ساختارها برای انجام عملیات مربوط به خود از شرایط بیت‌های ثبات پرچم بهره گرفته و در تمام انشعابات و فرآیندها، از آنها جهت کنترل برنامه استفاده می‌شود.

سوالات تستی فصل ۷

۱) در یک پرش شرطی، مقدار کدام ثبات حتماً تغییر می‌یابد؟

الف) AX ب) IP ج) SP د) BX

۲) دستور L LOOP معادل کدامیک از دستورات ذیل می‌باشد؟

الف) INC CX
JNZ L

ب) INC AX
JNZ L

ج) DEC CX
JNZ L

د) DEC AX
JNZ L

۳) کدامیک از دستورات زیر، محتوای عملوندهای خود را تغییر نمی‌دهد؟

الف) INC ب) XCHG ج) LEA د) CMP

۴) حاصل نهایی ثبات AX پس از اجرای قطعه برنامه زیر چیست؟

```
        MOV   CX, 5
        MOV   AX, 0
BACK:   INC   AX
        INC   AX
        LOOPNE  BACK
```

الف) 5 ب) 8 ج) 10 د) 4

۵) قطعه برنامه اسمبلی زیر چه شرطی را اجرا می‌کند؟

```
        CMP   AX, 10
        JNE   L1
        CMP   BX, 20
        JE    L1
        ADD   BX, AX
L1:     ADD   AX, BX
```

الف) If $AX \neq 0$ and $BX \neq 20$
AX = AX + BX

ب) If $AX = 0$ and $BX = 20$
BX = AX + BX

ج) If $AX \neq 10$ or $BX = 20$
AX = BX + AX

د) If $AX = 10$ or $BX \neq 20$
BX = BX + AX

۶) شرط خاتمه دستور LOOP چیست؟

الف) صفر شدن ثبات CX
ب) صفر شدن بیت پرچم Z
ج) یک شدن بیت پرچم Z
د) گزینه الف و ج

۷) شرط خاتمه دستور LOOPZ چیست؟

الف) صفر شدن ثبات CX
ب) صفر شدن بیت پرچم Z
ج) یک شدن بیت پرچم Z
د) گزینه الف و ب

۸) بعد از اجرای قطعه برنامه زیر، محتوای AX چیست؟

```
        MOV   CX, 10
        MOV   AX, 1
        MOV   DL, 2
FOR:    MUL   DL
        CMP   AX, 64
        LOOPNE FOR
```

الف) 0 ب) 64 ج) 10 د) 1024

۹) کدامیک از دستورات زیر مجاز است؟

الف) CMP CH, AX
ب) CMP TEXT, TEMP
ج) MOV ES, AX
د) MOV IP, DX

۱۰) دستورات زیر کدامیک از موارد را محاسبه می‌کنند و در AX قرار می‌دهند؟ (برای N>0)

```
        MOV   AX, X
        MOV   CX, N
P:      ADD   AX, AX
        DEC   CX
        CMP   CX, 0
        JNZ   P
```

الف) 2NX ب) $2^N X$ ج) $N^2 X$ د) N^2

۱۱) پس از اجرای قطعه کد زیر، محتوای DX چیست؟

```
        MOV    CX, 25
        MOV    DX, 15
FOR1:   DEC    DX
        CMP    DX, 0
        LOOPNE  FOR1
```

الف) 0 ب) 5- ج) 10- د) 15-

۱۲) کدام جمله صحیح نمی‌باشد؟

الف) در دستور CMP، هر دو عملوند نمی‌توانند متغیر حافظه‌ای باشند.

ب) دو دستور JNP و JPO معادل هم هستند.

ج) در دستور LOOP مقدار ثبات BX یک واحد کم می‌شود.

د) دستورات LOOPNZ و LOOPNE معادل هم هستند.

تمرین‌های فصل ۷

۱) قطعه برنامه‌ای بنویسید که فاکتوریل اعداد یک تا ۵ را محاسبه نماید.

۲) برنامه‌ای بنویسید که تعداد بیت‌های 1 ثبات AX را محاسبه نماید. راهنمایی: از دستورات شیفت استفاده نمایید.

۳) برنامه‌ای بنویسید که بین نمرات ۴۰ دانشجوی یک درس که در آدرس NUMBERS وجود دارد، جست‌وجو نماید و به محض یافتن عدد ۲۰، محتوای ثبات AL را برابر ۲۰ نماید. در غیر این صورت، مقدار ثبات AL صفر شود.

۴) برنامه‌ای بنویسید که در یک آرایه ۴۰ عنصری، تعداد عناصری که بزرگتر یا مساوی ۱۰ باشند را پیدا نماید.

۵) قطعه کدهای زیر را با استفاده از دستورات پرش و فرآیندهای تکراری، به زبان اسمبلی بازنویسی نمایید.

الف)
```
For  i=1  To  n
     S=S+i
```

ب)
```
For  i=n  DownTo  1
     S=S+i
```

ج)
```
i=1
While  i<=n  Do
{    i=i+2
     S=S+i
}
```

فصل هشتم:

عملیات بر روی رشته‌ها

هدف‌گذاری این فصل:

- آشنایی با ساختار و تعریف رشته‌ها
- عملیات انتقال، مقایسه، جستجو و بررسی بر روی رشته‌ها
- نحوه استفاده از سگمنت اضافی در عملیات رشته‌ای
- اجرای برنامه‌های رشته‌ای

۸-۱- مقدمه:

در فصول قبل، اغلب دستورالعمل‌ها و مثال‌ها بر روی داده‌های عددی انجام می‌گرفت. در این فصل با ساختار رشته‌ها و کاراکترها، آشنا خواهیم شد. می‌توان با استفاده از دستورات رشته‌ای، داده‌هایی از این نوع را تعریف و پیغام‌های مناسبی در حین یک برنامه در صفحه نمایش چاپ کرد.

رشته به مجموعه‌ای از داده‌های پشت سرهم اطلاق می‌شود. در زبان‌های برنامه‌نویسی سطح بالا، رشته‌ها با نگاه حروف و کاراکتر ارزیابی می‌شوند. در زبان اسمبلی نیز داده‌های رشته‌ای مورد استفاده قرار می‌گیرند؛ اما چنانچه در فصول قبلی اشاره شد، تمام داده‌ها در زبان اسمبلی، صرف نظر از هر نوعی، هنگام ذخیره‌سازی در حافظه به صورت الگویی از بیت‌های صفر و یک قرار می‌گیرند.

برای شروع کار به تعریفی از متغیرها اشاره می‌کنیم:

```
X  DB  10  DUP (?)
Y  DB  'I am a student', '$'
```

در خط اول، یک آرایه ده بایتی بدون مقدار اولیه و در خط دوم، یک آرایه ۱۵ حرفی (رشته) تعریف شده است. همانطور که قبلاً اشاره شد، هنگام تعریف رشته‌ها، مقادیر آن بین ' ' قرار می‌گیرند.

هر حرف از رشته، دارای کد اسکی خاص خود می‌باشد. در تعریف یک رشته، فواصل خالی که با کلید Space ایجاد می‌شود نیز دارای کد اسکی مربوط به خود می‌باشد و فضایی از آن رشته را به خود اختصاص می‌دهد.

طبق مطالب گفته شده، مشخص می‌شود که تفاوتی بین تعریف یک متغیر عددی با تعریف یک متغیر رشته‌ای وجود ندارد؛ بلکه فقط هنگام تخصیص مقادیر اولیه به رشته‌ها، آن مقادیر بین ' ' قرار می‌گیرند.

۸-۲- دستورات رشته‌ای:

دستوراتی که در این فصل به آنها اشاره خواهد شد به شرح زیر می‌باشد:

MOVS	انتقال یک رشته از یک محل به محل دیگر
LODS	بارگذاری یک رشته به ثبات AL یا AX
STOS	ذخیره محتوای ثبات AL یا AX در یک رشته اطلاعات

CMPS مقایسه دو رشته

SCAS جستجوی محتوای ثبات AL یا AX در یک رشته اطلاعات

نکته اساسی که در تمام عملیات رشته‌ای باید در نظر گرفت این است که آدرس محل یا موقعیت رشته مبدأ توسط ثبات SI و آدرس یا محل موقعیت رشته مقصد توسط ثبات DI نگهداری می‌شود. دستورات رشته‌ای برای پیدا کردن آدرس‌های مبدأ و مقصد رشته‌ها، به طور اتوماتیک به این ثبات‌ها مراجعه خواهند کرد؛ بنابراین قبل از بکار بردن دستورات رشته‌ای، بایستی آدرس‌های مورد نظر در این ثبات‌ها بارگذاری شده باشند. به نمونه‌ای از این مسأله توجه کنید:

```
SOURCE   DB   'Hello', '$'
DESTINATION   DB   'Welcome', '$'
```

کاراکتر $، به معنی پایان رشته می‌باشد؛ اما استفاده از آن اجباری نیست و فقط کمک می‌کند تا پایان یک رشته مشخص شود. در این مثال ۲ متغیر از نوع رشته تعریف شده که در متغیر با آدرس SOURCE عبارت Hello و در متغیر با آدرس DESTINATION عبارت Welcome قرار گرفته است.

اگر بخواهیم رشته اول را در رشته دوم کپی کنیم، آدرس رشته اول که رشته مبدأ می‌باشد را در ثبات SI و آدرس رشته دوم که رشته مقصد می‌باشد را در ثبات DI به صورت زیر ذخیره می‌کنیم.

```
LEA      SI, SOURCE
LEA      DI, DESTINATION
```

نکته مهم دیگری که در بحث رشته‌ها مطرح می‌باشد، جهت پیمایش یا عملیات در رشته‌ها می‌باشد. عملیات رشته‌ای می‌تواند از سمت چپ به راست یا از سمت راست به چپ پیمایش شود. اما برای اینکه تعیین نماییم که مایل هستیم از چه جهتی عملیات را انجام دهیم، باید در ابتدای عملیات رشته‌ای، آن را مشخص کنیم. برای این کار از دو دستور CLD و STD استفاده می‌نماییم.

۱- دستور CLD[1]:

این دستور که بدون عملوند می‌باشد، باعث می‌شود که بیت پرچم D که قبلاً در فصل دوم به آن اشاره کردیم، برابر صفر شود. صفر شدن این بیت سبب می‌شود که ترتیب عملیات روی رشته‌ها

[1] Clear Direction Flag

از چپ به راست آغاز شود. یا به عبارتی آدرس رشته اطلاعات که در ثبات‌های SI و DI قرار دارند، با هر بار استفاده از دستورات انتقال رشته‌ای، یک یا دو واحد افزایش پیدا کنند.

۲- دستور STD[1]:

این دستور که بدون عملوند می‌باشد، باعث می‌شود که بیت پرچم D، برابر یک شود. یک شدن این بیت سبب می‌شود که ترتیب عملیات روی رشته‌ها از انتهای رشته یعنی از سمت راست به چپ انجام شود. به عبارتی آدرس‌های مبدأ و مقصد رشته‌ها که در ثبات‌های SI و DI قرار دارند، با هر بار استفاده از دستورات انتقال رشته‌ای، یک یا دو واحد کاهش می‌یابند.

۳- کلمه تکرار REP:

دستورات رشته‌ای که در ادامه با آنها آشنا خواهیم شد، پس از هر بار اجرا، روی یک بایت یا یک کلمه عمل می‌کنند و برای اینکه روی کل رشته‌ها عمل نمایند، باید پس از هر عمل روی یک بایت یا یک کلمه، تکرار شوند. برای اینکه مجبور نشویم به تعداد طول رشته‌ها، دستورات رشته‌ای را تکرار نماییم، از کلمه REP قبل از این دستورات استفاده می‌کنیم.

این کار باعث می‌شود که دستورات رشته‌ای به اندازه عدد موجود در ثبات شمارنده CX، تکرار شوند. پس از هر تکرار، یک واحد از ثبات CX کاسته می‌شود. این کار تا زمانی ادامه خواهد یافت که ثبات CX برابر صفر شود. در هر تکرار نیز بسته به اینکه جهت عملیات از چپ به راست باشد یا از راست به چپ، محتوای ثبات‌های SI و DI یک یا دو واحد افزایش یا کاهش می‌یابند.

٤- دستور انتقال اطلاعات MOVS[2]:

این دستور باعث می‌شود که یک واحد از رشته مبدأ به رشته مقصد کپی شود. این انتقال می‌تواند یک بایتی یا دو بایتی باشد.

---

[1] Set Direction Flag
[2] Move String

دستور انتقال یک بایتی MOVSB، باعث می‌شود که یک بایت از رشته مبدأ روی یک بایت از رشته مقصد کپی شود. رشته مبدأ بدون تغییر باقی می‌ماند. اگر قبل از اجرای این دستور از دستور CLD استفاده شود، پس از انتقال یک واحد به ثبات‌های SI و DI افزوده می‌شود و در صورت استفاده از دستور STD، پس از انتقال، یک واحد از ثبات‌های SI و DI کاسته خواهد شد.

دستور انتقال دو بایتی MOVSW، باعث می‌شود که یک کلمه (۲ بایت) از رشته مبدأ روی یک کلمه از رشته مقصد کپی شود. در صورتی که از دستور CLD قبل از این دستور استفاده شود، پس از انتقال، دو واحد به ثبات‌های SI و DI افزوده خواهد شد و اگر از دستور STD استفاده شود، دو واحد از محتوای ثبات‌های SI و DI کاسته می‌شود.

مثال ۸-۱:

```
X   DB   'Ali and Reza', '$'
Y   DB   'Hadi and Omid', '$'
```

برای انتقال متغیر رشته‌ای X به محل متغیر رشته‌ای Y به ترتیب زیر عمل می‌کنیم:

```
LEA     SI, X       ; آدرس شروع رشته X در ثبات SI قرار می‌گیرد
LEA     DI, Y       ; آدرس شروع رشته Y در ثبات DI قرار می‌گیرد
CLD                 ; انتقال رشته مبدأ به مقصد از چپ به راست انجام شود
MOV     CX, 12      ; طول رشته مبدأ (۱۲ بایت) در ثبات شمارنده CX قرار می‌گیرد
REP     MOVSB       ; آغاز انتقال به تعداد شمارنده
```

توسط این دستور محتوای آدرس مبدأ که در ثبات SI قرار دارد به محتوای آدرس مقصد که در ثبات DI قرار دارد، منتقل می‌شود و پس از هر انتقال، یک واحد به ثبات‌های SI و DI افزوده می‌شود. این عمل ۱۲ بار انجام می‌شود تا کل رشته مبدأ منتقل شود.

اگر می‌خواستیم انتقال از سمت راست به چپ انجام شود، مجبور بودیم از دستور STD استفاده کنیم و آدرس خاتمه رشته X یعنی X+11 را در ثبات SI و آدرس خاتمه رشته Y یعنی Y+12 را در ثبات DI قرار دهیم و با استفاده از دستور REP MOVSB عمل انتقال از انتهای رشته آغاز و به ابتدای رشته ادامه پیدا کند. قبل از آن نیز ثبات CX باید با مقدار ۱۲ بارگذاری می‌شد.

در این شیوه ابتدا حرف پایانی رشته X یعنی a روی حرف پایانی رشته Y یعنی d کپی می‌شود و این فرآیند به صورت بایت به بایت تا ابتدای رشته تکرار می‌شود.

اگر از دستور MOVSW برای انتقال رشته X به رشته Y استفاده کنیم، انتقال اطلاعات به صورت کلمه- کلمه انجام می‌شود. در صورتی که طول رشته مبدأ زوج باشد (مانند مثال قبل) استفاده از این دستور مشکلی ایجاد نمی‌نماید اما شمارنده برنامه به جای عدد ۱۲ باید با عدد ۶ بارگذاری شود. در صورتی که طول رشته مبدأ فرد باشد، استفاده از دستور MOVSW باعث می‌شود که یک بایت اضافی پس از رشته مبدأ، روی مقصد کپی شود.

مثال ۸-۲: برنامه‌ای بنویسید که یک رشته n حرفی را به صورت معکوس در آفست 0100H قرار دهد.

حل: برنامه را برای یک رشته ۴ حرفی می‌نویسیم. می‌توان این برنامه را به رشته‌هایی با طول بیشتر تعمیم داد. با توجه به اینکه در دستورات رشته‌ای، سگمنت اضافی و ثبات ES نیز استفاده می‌شود، بنابراین برنامه به طور کامل نوشته شده است.

توجه داشته باشید که در خط ۲۵، دستور MOVSB با توجه به دستور STD، یک واحد از آدرس مقصد می‌کاهد؛ یعنی به یک بایت قبل از آغاز رشته مقصد اشاره خواهد داشت. برای حل این مشکل ۲ واحد به ثبات DI افزوده می‌شود تا به اولین بایت از مقصد اشاره شود.

```
1.      PAGE    100, 100
2.      TITLE   'String Reverse Copy'
3.      S_SEG   SEGMENT  STACK  'Stack'
4.              DW  100  DUP (0)
5.      S_SEG   ENDS
6.      D_SEG   SEGMENT  'Data'
7.              X   DB   'Reza', '$'              ;  تعریف متغیر رشته‌ای X
8.              N   DB   4                        ;  تعیین طول رشته
9.              ORG  100H                         ;  این دستور باعث می‌شود تا رشته مقصد Y
10.             Y   DB   4  DUP (?)               ;  از آفست 100H آغاز شود
11.     D_SEG   ENDS
12.     C_SEG   SEGMENT  'Code'
13.             ASSUME  SS:S_SEG, DS:D_SEG, ES:D_SEG, CS:C_SEG
14.     MAIN    PROC    FAR
15.             MOV     AX, D_SEG
16.             MOV     DS, AX
17.             MOV     ES, AX                    ;  مقداردهی به ثبات ES
```

```
18.             SUB     AX, AX          ; صفر شدن ثبات AX
19.             LEA     SI, X           ; بارگذاری آدرس شروع رشته مبدأ در SI
20.             LEA     DI, Y           ; بارگذاری آدرس شروع رشته مقصد در DI
21.             MOV     CX, N           ; بارگذاری شمارنده با طول رشته
22.             ADD     SI, N-1         ; افزایش ثبات SI به اندازه N-1 جهت
                                        ; اشاره نمودن به انتهای رشته
23.             STD                     ; انجام عملیات انتقال از راست به چپ
24.     BACK:   MOVSB                   ; آغاز عملیات انتقال
25.             ADD     DI, 2
26.             LOOP    BACK            ; عملیات انتقال به تعداد شمارنده

27.             MOV     AX, 4C00H
28.             INT     21H
29.     MAIN    ENDP
30.     C_SEG   ENDS
31.             END   MAIN
```

برنامه شماره (۱): معکوس کردن یک رشته n حرفی

مثال ۸-۳: برنامه‌ای بنویسید که حاصل جمع ارقام موجود در یک رشته n حرفی را محاسبه نماید.

```
X       DB  '2045'
SUM     DB  ?
```

در این مثال، برنامه باید بتواند مجموع ارقام 2+0+4+5=11 را حساب کند. برنامه را برای حالت کلی‌تری که طول رشته n باشد می‌نویسیم.

```
1.              LEA     SI, X           ; بارگذاری آدرس رشته در ثبات SI
2.              MOV     CX, n           ; بارگذاری طول رشته در ثبات CX
3.              MOV     AL, 0           ; بارگذاری صفر در ثبات AL
4.      BACK:   MOV     BL, [SI]        ; دریافت یک بایت از رشته
5.              SUB     BL, 30H         ; کاستن ۳۰ واحد از کد اسکی
6.              ADD     AL, BL          ; جمع عدد با ثبات AL
7.              INC     SI              ; افزایش SI برای اشاره به کاراکتر بعدی
8.              LOOP    BACK            ; اجرای حلقه تا صفر شدن ثبات CX
9.              MOV     SUM, AL         ; ذخیره نتیجه در ثبات AL
```

برنامه شماره (۲): حاصل جمع ارقام موجود در یک رشته n حرفی

نکته‌ای که باید اشاره شود این است که با توجه به کد اسکی واقع در ضمیمه کتاب، مشاهده می‌شود که کد اسکی اعداد نسبت به خود عدد ۳۰ واحد اختلاف دارند. یعنی مثلاً کد اسکی صفر برابر ۳۰ می‌باشد، کد اسکی یک برابر ۳۱ و... . بنابراین با کاستن ۳۰ واحد از هر عضو عددی رشته می‌توان به خود عدد رسید. اگر این ۳۰ واحد از کد اسکی هر رقم واقع در رشته کاسته نمی‌شد، کدهای اسکی با هم جمع می‌شدند و نه اعداد رشته.

نکته ۱: در دستورات MOVSB و MOVSW، سگمنت داده اضافی ES استفاده می‌شود.

نکته ۲: در دستورات MOVSB و MOVSW، باید آدرس شروع رشته مبدأ در ثبات SI و آدرس شروع رشته مقصد در ثبات DI قرار گیرد.

به مثالی در مورد تفاوت‌های دستور MOVSB و MOVSW می‌پردازیم.

مثال ۸-٤:

```
SRC   DB   'Reza is a teacher'
DST   DB   17   DUP (?)
LEA        SI, SRC
LEA        DI, DST
```

حال اگر بخواهیم دستورات انتقال رشته‌ای را برای این مثال به کار ببریم، می‌توانیم به موارد زیر اشاره کنیم:

الف)

```
MOV    CX, 5
REP    MOVSB
```

توسط این دو دستور ۵ بایت از رشته اول به رشته دوم، انتقال می‌یابد؛ یعنی DST = 'Reza '. توجه داشته باشید که کلمه Reza به همراه یک Space به رشته مقصد انتقال می‌یابد.

ب)

```
MOV    CX, 5
REP    MOVSW
```

در این حالت ۵ کلمه (۱۰ بایت) از رشته اول به رشته دوم انتقال می‌یابد؛ یعنی:

DST = 'Reza is a '

ج)

MOVS

در این حالت کل عناصر رشته اول به رشته دوم انتقال می‌یابد؛ یعنی:

DST = 'Reza is a teacher'

نکته ۳: هنگام انتقال رشته از مبدأ به مقصد، رشته مبدأ بدون تغییر باقی خواهد ماند.

۵- دستور LODS[1]:

این دستور باعث می‌شود که یک رشته به اندازه یک بایت یا یک کلمه، از آدرسی که آفست آن توسط ثبات SI تعیین می‌شود، در ثبات AL یا AX قرار گیرد. این دستور به دو شکل LODSB برای انتقال یک بایت و LODSW برای انتقال دو بایت از رشته استفاده می‌شود.

آدرس رشته در ابتدا در ثبات SI بارگذاری می‌شود. عبارت REP معمولاً در این دستور استفاده نمی‌شود. اما اگر این عبارت قبل از دستور استفاده شود، آخرین اجرای دستور LODS محاسبه می‌شود.

توسط دستور LODSB، یک بایت از رشته‌ای که آدرس آن در ثبات SI قرار گرفته است وارد ثبات AL می‌شود. اگر از دستور CLD قبل از اجرای این دستور استفاده شده باشد، پس از انتقال یک بایت از رشته به ثبات AL، یک واحد به محتوای ثبات SI اضافه می‌شود. در صورت استفاده از دستور STD، پس از انتقال، یک واحد از ثبات SI کاسته خواهد شد.

توسط دستور LODSW، دو بایت از رشته‌ای که آدرس آن در ثبات SI قرار دارد، وارد ثبات AX می‌شود. اگر از دستور CLD قبل از این دستور استفاده شود، پس از عمل انتقال، دو واحد به محتوای ثبات SI افزوده خواهد شد. در صورتی که از دستور STD استفاده شود، پس از انتقال، دو واحد از ثبات SI کاسته خواهد شد.

---

[1] Load String

مثال ۸-۵:

```
SRC     DB      'Reza'
        CLD
        LEA     SI, SRC
        LODSW
```

در این مثال حروف Re وارد ثبات AX می‌شود و SI دو واحد افزایش پیدا می‌کند.

مثال ۸-٦:

```
SRC     DB      'Ali'
        LEA     SI, SRC
        MOV     CX, 2
        REP     LODSB
```

در این مثال، پس از اجرای دستور آخر، حرف I که آخرین حرف رشته SRC می‌باشد، وارد ثبات AL می‌شود. در این دستور همواره مقدار آخرین تکرار مهم می‌باشد. مثلاً در اینجا ابتدا حرف A، سپس حرف L و در نهایت حرف I وارد ثبات AL می‌شود. هنگامی که ثبات CX مساوی صفر شود، عملیات متوقف شده و آخرین مقدار در ثبات AL نگهداری خواهد شد.

مثال ۸-۷: قطعه برنامه‌ای بنویسید که تعداد حروف A در یک رشته را بشمارد.

```
1.      STRING  DB      'AABAC'
2.              CLD
3.              LEA     SI, STRING      ; بارگذاری آدرس رشته در ثبات SI
4.              MOV     CX, 5           ; بارگذاری طول رشته در ثبات CX
5.              MOV     DL, 'A'         ; بارگذاری کاراکتر مورد جستجو در ثبات DL
6.              MOV     BH, 0           ; شمارنده تعداد کاراکتر A
7.      L2:     LODSB                   ; دریافت یک بایت از رشته در ثبات AL
8.              CMP     AL, DL          ; مقایسه کاراکتر دریافتی با حرف A
9.              JNE     L1              ; در صورت عدم تساوی انشعاب به L1
10.             INC     BH              ; افزایش شمارنده در صورت یافتن A
11.     L1:     LOOP    L2              ; انشعاب به L2 برای دریافت کاراکتر بعدی
```

برنامه شماره (۳): شمارش تعداد حروف A

نکته ٤: در دستورات LODSB و LODSW، ثبات سگمنت داده باید DS باشد.

٦- دستور STOS[1]:

دستور STOS باعث می‌شود که مقدار موجود در ثبات‌های AL و AX به محلی کـه آفسـت آن توسط ثبات DI مشخص شده، انتقال یابد.

این دستور به دو شکل STOSB برای انتقال محتوای ثبات AL و STOSW بـرای انتقـال یـک کلمه از ثبات AX به محلی که آفست آن در ثبات DI قرار دارد، به کار می‌رود.

نکته ۵: در دستور STOS ثبات سگمنت داده باید ES باشد.

مثال ۸-۸: در ۲۰ بایت خانه حافظه به آدرس STRING، مقدار صفر قرار دهید.

```
MOV    CX, 20
CLD
LEA    DI, STRING
MOV    AL, 0
REP    STOSB
```

دستور REP باعث می‌شود که محتوای ثبات AL در هر بار اجرای حلقه وارد حافظه STRING شـود و آدرس رشته هم در هر تکرار یک واحد افزایش یابد. در پایـان برنامـه و پـس از ۲۰ بـار اجـرا، تمـام حافظه مورد نظر با صفر پر می‌شود.

۷- دستور CMPS[2]:

این دستور برای مقایسه دو رشته که یکی از رشته‌ها به عنوان رشـته مبـدأ در سـگمنت داده و رشته دیگر به عنوان رشته مقصد در سگمنت اضافی وجود دارد، به کار می‌رود.

SOURCE ⟶ DS:SI
DEST ⟶ ES:DI

---

[1] Store String
[2] Compare String

آدرس آفست شروع رشته اول در ثبات SI و آفست شروع رشته دوم در ثبات DI قرار می‌گیرد. این دستور مقایسه به دو شکل CMPSB و CMPSW به کار می‌رود. عمل مقایسه به صورت تفریق رشته اول و دوم می‌باشد و روی بیت‌های C، P و A از ثبات پرچم تأثیر می‌گذارد. توجه داشته باشید که محتوای رشته‌ها بعد از مقایسه تغییری نخواهند کرد.

با اجرای دستور CMPSB، یک بایت از رشته مبدأ، واقع در سگمنت داده، با یک بایت از رشته مقصد واقع در سگمنت اضافی، مقایسه می‌شود. نکته‌ای که باید در نظر گرفت این است که تکرار مقایسه‌ها برای دو رشته، تا جایی انجام می‌شود که شرط خواسته شده برقرار باشد. مثلاً اگر بخواهیم تساوی دو رشته را بررسی نماییم، با اولین مورد نقض در تساوی دو رشته، مقایسه رشته‌ها متوقف خواهد شد.

در دستورات مقایسه‌ای علاوه بر دستور تکرار REP، دستورهای دیگری نیز مانند REPE، REPNE و REPNZ وجود دارند.

در دستور REP، عمل مقایسه تا صفر شدن ثبات CX ادامه می‌یابد.

دستور REPE، کار مقایسه دو رشته را تا زمانی که این دو رشته بایت به بایت باهم برابر باشند، ادامه می‌دهد.

دستور REPNE، کار مقایسه دو رشته را تا زمانی که این دو رشته بایت به بایت با هم مخالف باشند، ادامه می‌دهد و با اولین تساوی بین یک بایت از هر دو رشته، عمل مقایسه خاتمه می‌یابد. دستور REPNZ نیز همانند دستور REPNE می‌باشد.

بعد از هر بار اجرای دستور CMPSB، در صورتی که از دستور CLD قبل از این دستور استفاده شده باشد، یک واحد به ثبات‌های SI و DI افزوده خواهد شد و اگر از دستور STD استفاده شده باشد، یک واحد از ثبات‌های SI و DI کاسته می‌شود.

مثال ۸-۹: قطعه برنامه‌ای بنویسید که از بین ۲ رشته معرفی شده، تشخیص دهد که آیا دو رشته با هم برابر می‌باشند یا خیر؟ اگر برابر نباشند ثبات BX مساوی صفر و اگر برابر باشند ثبات BX برابر یک شود.

```
1.    STRING1  DB      'ALIREZA'          ; معرفی رشته اول
2.    STRING2  DB      'ALIREZA'          ; معرفی رشته دوم
;-------------------------------------------------------
3.             MOV     BX, 0              ; بارگذاری اولیه BX با عدد صفر
4.             CLD                        ; انتخاب جهت مقایسه از چپ به راست
5.             MOV     CX, 7              ; بارگذاری شمارنده به اندازه طول رشته
6.             LEA     SI, STRING1        ; آدرس شروع رشته اول در ثبات SI
7.             LEA     DI, STRING2        ; آدرس شروع رشته دوم در ثبات DI
8.       REPE  CMPSB                      ; مقایسه بایت به بایت دو رشته
9.             JNE     EXIT               ; در صورت عدم تساوی انشعاب به EXIT
10.            MOV     BX, 1              ; در صورت تساوی BX یک می‌شود

11.      EXIT: MOV     AX, 4C00H          ; دستورات خروج
12.            INT     21H
```

برنامه شماره (۴): تشخیص برابر بودن دو رشته

دستور CMPSW عمل مقایسه را به صورت کلمه به کلمه بین دو رشته انجام می‌دهد. در هر بار یک کلمه از رشته مبدأ با یک کلمه از رشته مقصد مقایسه می‌شود. این دستور می‌تواند برای تکرار از دستورات REP، REPE، REPNE و REPNZ استفاده نماید.

هنگام استفاده از دستور CMPSW، اگر از دستور CLD استفاده شده باشد، ۲ واحد به ثبات‌های SI و DI افزوده خواهد شد و اگر از دستور STD استفاده شود، پس از هر بار اجرای دستور CMPSW، ۲ واحد از ثبات‌های SI و DI کاسته خواهد شد.

۸- دستور SCAS[1]:

دستور SCAS برای جستجوی عنصری در داخل یک رشته که در یکی از ثبات‌های AL و یا AX قرار دارد، به کار گرفته می‌شود. رشته مورد جستجو یک رشته از نوع مقصد بوده و باید در سگمنت اضافی تعریف شود. همچنین آفست آدرس این رشته از طریق ثبات DI تعیین می‌شود. این

[1] Scan String

دستور نیز مثل بقیه دستورات رشته‌ای به صورت یک بایتی و دو بایتی وجود دارد. همچنین از کلمه‌های تکرار REP، REPE، REPNE و REPNZ نیز می‌توان استفاده نمود.

توسط دستور SCASB، یک بایت اطلاعات واقع در ثبات AL با یک بایت اطلاعات رشته تعریف شده در برنامه، مقایسه می‌شود. پس از عمل مقایسه، در صورتی که از دستور CLD قبل از این دستور استفاده شده باشد، یک واحد به محتوای ثبات DI افزوده می‌شود و اگر از دستور STD استفاده شده باشد، یک واحد از ثبات DI کاسته خواهد شد.

مثال ۸-۱۰: قطعه برنامه‌ای بنویسید که بتواند در یک رشته، محل وقوع اولین کاراکتر 'A' را مشخص نماید و آن را در ثبات BX قرار دهد و در صورت موجود نبودن، مقدار BX برابر صفر شود.

```
1.        DST     DB      'Reza and Ali'          ; تعریف رشته
2.        N       DW      12                      ; تعریف طول رشته
;-----------------------------------------
3.                MOV     AL, 'A'                 ; قرار دادن کاراکتر مورد جستجو در AL
4.                MOV     CX, N                   ; قرار دادن مقدار شمارنده در ثبات CX
5.                LEA     DI, DST                 ; تعیین آدرس مقصد در ثبات DI
6.                MOV     BX, 0                   ; بارگذاری ثبات BX با مقدار صفر
7.                CLD                             ; انتخاب جهت مقایسه از چپ به راست
8.        REPNE   SCASB                           ; جستجوی رشته
9.                CMP     CX, 0                   ; مقایسه CX با صفر
10.               JE      EXIT                    ; اگر CX=0 پس کاراکتر پیدا نشد
11.               SUB     N, CX                   ; تعیین محل کاراکتر A
12.               MOV     BX, N                   ; قرار دادن محل کاراکتر در ثبات BX
13.       EXIT:   MOV     AX, 4C00H               ; دستورات خروج
14.               INT     21H
```

برنامه شماره (۵): مشخص نمودن محل وقوع اولین کاراکتر A در یک رشته

توسط دستور SCASW، دو بایت اطلاعات واقع در ثبات AX با دو بایت اطلاعات از رشته تعریف شده در برنامه، مقایسه می‌شود. پس از عمل مقایسه، در صورتی که از دستور CLD استفاده شده باشد، پس از عمل مقایسه، دو واحد به محتوای ثبات DI افزوده می‌شود و اگر از دستور STD استفاده شده باشد، دو واحد از محتوای ثبات DI کاسته خواهد شد.

این دستور نیز می‌تواند از دستورات تکرار REP، REPE، REPNE و REPNZ استفاده نماید.

۸-۳- خلاصه:

در این فصل با دستورات رشته‌ای و نحوه استفاده از این دستورات آشنا شدیم. استفاده از دستورات رشته‌ای می‌تواند به انعطاف و کارایی برنامه‌ها کمک نماید. از دستورات رشته‌ای می‌توان در مواردی مثل چاپ یک اخطار یا یک نتیجه در صفحه نمایش یا پردازش داده‌های رشته‌ای استفاده نمود.

سوالات تستی فصل ۸

۱) پس از اجرای قطعه برنامه زیر، متغیر S2 حاوی چه رشته‌ای خواهد بود؟

```
S1  DB  'Reza'
S2  DB  4  DUP (?)
CLD
MOV  CX, 2
LEA  SI, S1
LEA  DI, S2
REP  MOVSB
```

الف) Re ب) Reza ج) za د) azeR

۲) در سؤال ۱، اگر به جای دستور MOVSB از دستور MOVSW استفاده می‌کردیم، در نهایت چه رشته‌ای در S2 قرار می‌گرفت؟

الف) Reza ب) azeR ج) Re د) eR

۳) اگر بخواهیم عملیات پردازش رشته‌ها از انتها به ابتدای رشته انجام شود از چه دستوری استفاده می‌کنیم؟

الف) STC ب) STD ج) CLC د) CLD

۴) کاربرد دستور CLD در پردازش رشته‌ها چیست؟

الف) پردازش رشته‌ها از راست به چپ ب) پردازش رشته‌ها از چپ به راست

ج) افزایش مقادیر SI و DI د) کاهش مقادیر SI و DI

۵) در صورتی که DF=1 و SI=8 باشد، با اجرای دستور LODSW مقدار SI چه خواهد بود؟

الف) ۶ ب) ۱۰ ج) ۷ د) ۹

۶) در صورتی که DF=0 و SI=8 باشد، پس از اجرای دستور STOSW مقدار SI چه خواهد بود؟

الف) ۶ ب) ۱۰ ج) ۷ د) ۹

۷) برای ذخیره یک رشته به اندازه یک بایت از ثبات AL، از دستور ..... و برای فراخوانی رشته به اندازه یک کلمه به ثبات AX، از دستور ..... استفاده می‌کنیم.

الف) MOVSB و MOVSW ب) LODSB و STOSW

ج) STOSB و LODSW د) LODSB و LODSW

۸) حاصل اجرای قطعه برنامه زیر چیست؟

```
STD
MOV   CX, 3
REP     MOVSW
```

الف) سه بایت از رشته مبدأ از چپ به راست منتقل می‌شود.

ب) سه بایت از رشته مبدأ از راست به چپ منتقل می‌شود.

ج) سه کلمه از رشته مبدأ از چپ به راست منتقل می‌شود.

د) سه کلمه از رشته مبدأ از راست به چپ منتقل می‌شود.

۹) کدام دستور مقدار معینی را در یک رشته جستجو می‌کند؟

الف) CMPSB ب) MOVSB ج) SCASB د) LODSB

تمرین‌های فصل ۸

۱) با استفاده از دستورات رشته‌ای، برنامه‌ای بنویسید که تمام ۱۰۰ بایت متغیر حافظه‌ای Name را با صفر پر کند.

۲) برنامه‌ای بنویسید که تعداد کاراکترهای '*' را در یک رشته n حرفی شمارش نماید.

۳) برنامه‌ای بنویسید که روی رشته عددی '4506'، عملیات زیر را انجام دهد:

الف) محاسبه مجموع ارقام عددی این رشته

ب) تعداد ارقام این رشته

ج) مقلوب کردن رشته (به صورت '6054')

۴) برنامه‌ای بنویسید که در یک رشته n بایتی، تمام حروف بزرگ موجود در رشته را به رشته مقصد کپی نماید.

۵) برنامه‌ای بنویسید که در یک رشته n بایتی، تمام حروف کوچک را به حروف بزرگ تبدیل نماید.

۶) برنامه‌ای بنویسید که دو رشته را در کنار هم و روی رشته سوم کپی نماید.