بخش مفاهیمی از ریزپردازنده

فهرست مطالب


فصل اول
آشنائی با پردازنده و بخش
های مختلف آن

- پردازنده و تفاوت آن با ریزپردازنده
- قسمت های مختلف:
  - واکشی Fetch
  - رمزگشائی Decode
  - اجرائی Execution
  - نهائی سازی Commit


فصل دوم
پردازنده سنتی

- پردازنده سنتی
- مشکلات آن


فصل سوم
پردازنده خط لوله

- پردازنده خط لوله
- مشکلات آن

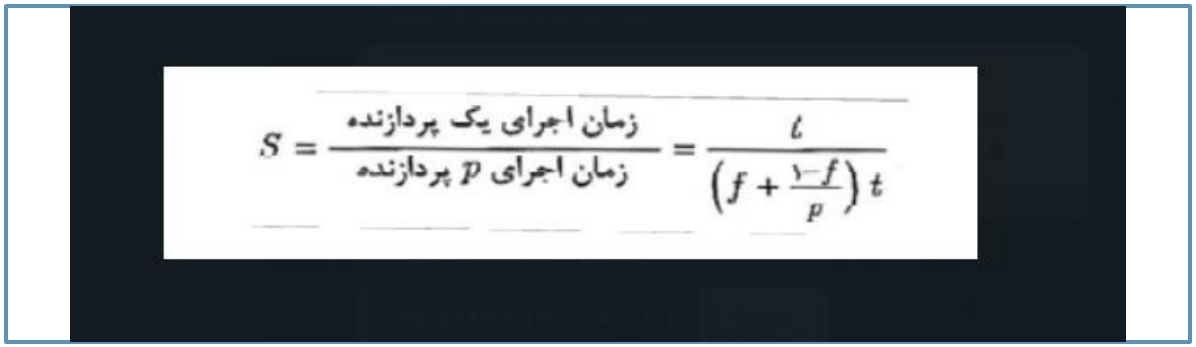

فصل چهارم
پردازنده سوپراسکالر

- پردازنده سوپراسکالر
- مشکلات آن


فصل پنجم
توماسولو و پنجره دستور

- توماسولو
- پنجره دستور


فصل ششم
اجرای موازی و محاسبات
تسریع



فرمول آمدال برای محاسبه تسریع

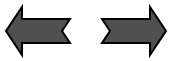
بخش برنامه نویسی به زبان اسمبلی


فصل هفتم
نمایش داده ها در کامپیوتر

فهرست مطالب فصل هفتم

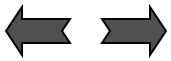
نمایش داده ها در کامپیوتر

در زبانهای سطح بالا نگران اینکه داده ها در کامپیوتر چگونه نمایش داده میشوند نیستیم ولی در زبان های اسمبلی بایستی بفکر چگونگی ذخیره داده باشیم و اغلب با کار تبدیل داده ها از یک نوع به نوع دیگر مواجه می باشیم.



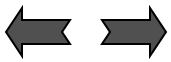
اعداد دودویی و شانزده شانزدهی

حافظه های کامپیوتر فقط می تواند ارقام **0** یا یک را در خود ذخیره نماید که به آنها بیت گفته میشود. در سیستم دودوئی اعداد از بیت ها تشکیل شده اند.



اعداد دودویی و شانزده شانزدهی

اعداد شانزدهی از ارقام ۰ تا ۱۵ تشکیل شده اند. برای راحتی، ارقام ۱۰ تا ۱۵ را A تا F نشان داده می شود.



مثال :

1011

سیستم دودوئی شبیه سیستم دهدهی است با این تفاوت که ارقام از سمت راست به چپ به جای ارزش 1 ، 100 ، 1000 ، ..... ارزش 1 ، 2 ، 4 ، 8 ، ..... دارند. بنابرین 1101 در سیستم دودوئی معادل 13 می باشد.

تبدیل اعداد شانزدهی به دودوئی

هر رقم در سیستم شانزدهی بوسیله چهار رقم در سیستم دودوئی قابل نمایش می باشد.

در اسلاید بعد مثالی آورده شده است.

مثال :

- 0100 معادل 4
- 1110 معادل E

برای تبدیل اعداد شانزدهی به دودوئی کافی است که به جای هر رقم ،چهار بیت معادل آن قرار داد.

در اسلاید بعد مثالی آورده شده است.

مثال :

برای تبدیل اعداد شانزدهی به دودوئی کافی است که به جای هر رقم ،چهار بیت معادل آن قرار داد.

2AD5 معادل 0010101011010101 در سیستم دودوئی می باشد.

تبدیل اعداد دودوئی به شانزدهی

برای تبدیل اعداد دودوئی به شانزدهی ،ارقام عدد داده شده را از سمت راست به ترتیب به صورت گروههای چهار بیتی درآورده آنگاه معادل هر گروه در سیستم شانزدهی را جایگزین می نماییم.

در اسلاید بعد مثالی آورده شده است.

مثال :

عدد 1010011101 در سیستم دودوئی در نظر بگیرید.

0010	1001	1101
2	9	D

کدهای کارکتری

کارکترهای قابل چاپ

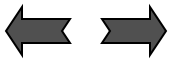
کد اسکی	کرکتر
48 تا 57	0 تا 9
65 تا 90	A تا Z
97 تا 122	a تا z

نکته :

- کرکترهای قابل چاپ دارای کدهای 32 تا 126 می باشند.
- کرکترهای کنترلی دارای کدهای 0 تا 31 می باشند.

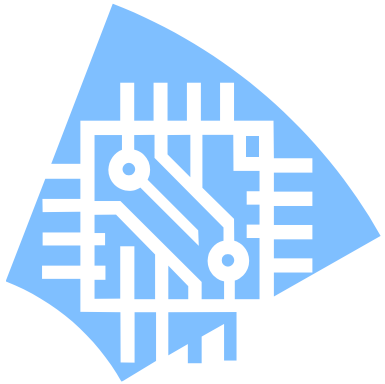
کارکترهای کنترلی

کد اسکی	کرکتر
27	ESC
10	CR
13	LF



نمایش مکمل ۲ برای اعداد صحیح علامتدار

اعداد منفی در کامپیوتر بصورت مکمل ۲ نمایش داده می شوند. وقتی یک عدد به شکل مکمل دو نشان داده می شود تعداد بیت های مورد استفاده باید از قبل مشخص گردد (۳۲ , ۱۶ , ۸) .



روش محاسبه مکمل ۲ یک عدد :

- **عدد را بصورت دودوئی درآورده.**
- **آنرا به تعداد بیت های مشخص شده تبدیل نموده .**
- **سپس صفر ها را 1 و 1 ها را به صفر تبدیل نموده .**
- **نتیجه را با یک جمع می نماییم.**

در اسلاید بعد مثالی آورده شده است.

مثال : -25

- 11001 معادل 25
- هشت بیتی نموده 00011001
- صفرها را به یک و یک ها را به صفر تبدیل نموده 11100110
- نتیجه را با یک جمع نموده 11100111

مقدار 11100111 در سیستم دودوئی نمایش عدد -25 می باشد.


فصل هشتم
قسمت های یک سیستم کامپیوتری

فهرست مطالب فصل هشتم

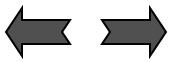
حافظه اصلی

حافظه اصلی یک PC را می توان بصورت مجموعه ای از سگمنت ها در نظر گرفت. هر سگمنت بطول ۶۴ کیلو بایت می باشد.

Memory

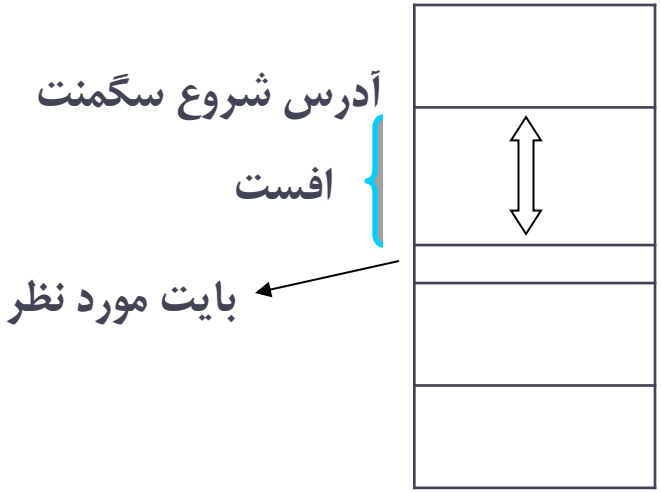
نکات :

- آدرس شروع هر سگمنت مضرب ۱۶ می باشد.
- آدرس شروع هر سگمنت در مبنای ۱٦ به رقم صفر ختم می شود .
- آدرس هر سگمنت برابر اولین چهار رقم شانزدهی آدرس آن می باشد.



نکته :

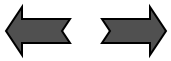
آدرس هر بایت از حافظه اصلی را می توان با سگمنت حاوی بایت مزبور و به دنبال آن افستی که از ابتدای سگمنت یاد شده در نظر گرفته می شود، آدرس دهی کرد.



مثال :

نماد 18A3 : 5B27 به بایتی که 5B27 بایت از اول سگمنت که از آدرس 18A30 شروع می شود، قرار دارد، اشاره می کند.

18A30 + 5B27 = 1E557



واحد پردازش مرکزی (CPU)

تراشه 8088 دارای 14 ثبات می باشد که هر کدام یک محل ذخیره سازی داخلی بوده و می تواند یک کلمه 16 بیتی را نگه دارد. دستورالعمل ها معمولا داده ها را بین این ثبات ها یا حافظه اصلی انتقال داده و یا عملیاتی را روی داده های ذخیره شده در ثبات ها یا حافظه انجام می دهند. تمام این ثبات ها دارای نام بوده و بسیاری از آنها دارای اهداف ویژه ای هستند.

اسامی و اهداف ثبات ها

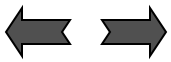
- AX اکومولاتور، کاربرد همگانی، بایت بالایی برابر AH و بایت پایینی برابر AL
- BX کاربرد همگانی، بایت بالایی برابر BH و بایت پایینی برابر BL
- CX کاربرد همگانی، بایت بالایی برابر CH و بایت پایینی برابر CL
- DX کاربرد همگانی، بایت بالایی برابر DH و بایت پایینی برابر DL
- CS شماره سگمنت حافظه ای می باشد که دستورالعمل های اجرائی جاری در آنجا قرار دارد.
- DS سگمنت داده ها را می دهد.
- ES سگمنت فوق العاده را می دهد.
- SS سگمنت پشته را می دهد.

اسامی و اهداف ثبات ها

- **SP** اشاره گر پشته، افست بالای پشته در سگمنت پشته.
- **BP** اشاره گر مبنا، افست نقطه مراجعه (**reference Point**) در سگمنت پشته.

اسامی و اهداف ثبات ها

- **SI** اندیس منبع، افست رشته کاراکتری منبع در انتقال رشته های کاراکتری.
- **DI** اندیس مقصد: افست رشته کاراکتری مقصد.
- **IP** اشاره گر دستور العمل ها، آفست دستور العمل بعدی در سگمنت کد برای دستیابی ثبات نشانه ها مجموعه ای از نشانه ها یا بیت های وضعیت.



ثبات نشانه

بعضی از ۱۶ بیت این ثبات برای نشان دادن نتیجه اجرای دستور العملها بوسیله دستور العمل های مختلف تغییر پیدا می کنند. هر کدام از این بیت ها را یک بیت وضعیت یا نشانه می گویند. اسامی برخی از این بیت ها عبارتند از :

۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

در اسلاید های بعد به توضیح هر یک از نشانه ها می پردازیم.

نشانه ها

- نشانه سرریزی — flow Flag — OF
- نشانه صفر — Zero Flag — ZF
- نشانه نقلی — Carry Flag — CF

۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

نشانه ها

- نشانه کمکی — AF — Auxiliary Flag
- نشانه توازن — PF — Parity Flag
- نشانه علامت — SF — Sign Flag

۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

نشانه ها

- نشانه جهت — Direct Flag — DF
- نشانه دام — Trap Flag — TF
- نشانه وقفه — Interrupt Flag — IF

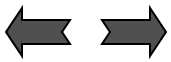
۱۵	۱۴	۱۳	۱۲	۱۱	۱۰	۹	۸	۷	۶	۵	۴	۳	۲	۱	۰
				OF	DF	IF	TF	SF	ZF		AF		PF		CF


فصل نهم
استفاده از اسمبلر

فهرست مطالب فصل نهم

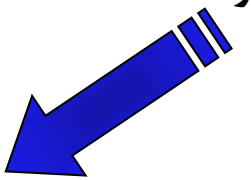
دستورالعملهای زبان اسمبلی

هر دستور زبان اسمبلی در روی یک خط فایل کد منبع وارد میشود.یک خط می تواند حد اکثر ۱۲۸ کرکتر داشته باشد. استفاده از توضیحات مناسب در برنامه مهم است. هر توضیحی با کرکتر ; شروع میشود و تا انتهای خط می تواند ادامه داشته باشد.



زبان اسمبلی دارای سه نوع دستورالعمل می باشد :

- دستورالعمل مانند **ADD AX , 244**
- دستور اسمبلی مانند **PAGE**
- ماکرو



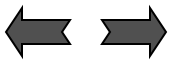
نوعی دستورالعمل است که در آن تعدادی دستورالعملها، دستورات اسمبلی یا حتی ماکروهای دیگر قرار گرفته اند.

کد منبع

کل برنامه از چهار قسمت تشکیل شده است. هر قسمت با دستور **SEGMENT** شروع و با **ENDS** ختم می گردد.

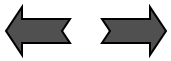
```
Segment _  name    SEGMENT
                      .
                      .
                      .
Segment _ name
ENDS
```

برنامه با **END** ختم می گردد. دستور **END** به اسمبلی می گوید که پردازش دستورات کد منبع را خاتمه دهد.



SEGMENT ها عبارتند از :

- **STACK SEGMENT**
- **DATA SEGMENT**
- **EXTRA SEGMENT**
- **CODE SEGMENT**

شکل کلی برنامه

```
STACK _ SEG      SEGMENT    PARA   STACK  'STACK'
```

اندازه پشته مشخص می گردد.

```
STACK _ SEG    ENDS
DATA _ SEG     SEGMENT PARA        'DATA'
```

متغیر ها اعلان می شوند

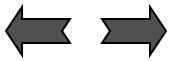
```
DATA _ SEG   ENDS
EXTRA _ SEG SEGMENT  PARA   'EXTRA'
```

متغیرهای مربوط به پردازش رشته ها اعلان می شوند

```
EXTRA _ SEG       ENDS
CODE _ SEG SEGMENT    PARA   'CODE'
START:
```

دستورالعمل های برنامه

```
CODE _ SEG     ENDS
END    START
```



عملوندهای دستورات DW , DB

عملوندهای عددی را میتوان به صورت دهدهی ، شانزده تایی، دودوئی یا هشت تایی بیان کرد.
پسوندهای مورد استفاده عبارتند از

سیستم عددی	مبنا	پسوند
شانزده تایی	۱۶	
دودوئی	۲	
هشت تایی	۸	O

در اسلایدهای بعدی چندین مثال آورده شده است.

مثال :

```
MASK0    DB    01111101B
MASK1    DB    1750
MASK2    DB    7DH
MASKL3   DB    12D
```

چهار مقدار فوق از نوع بایت تعریف شده و معادلند .

مثال :

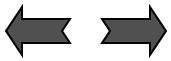
```
WORD1     DW     1000
```

WORD1 از نوع WORD تعریف شده با مقدار ۱۰۰۰ .

مثال :

```
X       DB    10 , 12 , 24 , 5 , 16
```

X یک آرایه پنج عنصری از نوع بایت می باشد.



مثال :

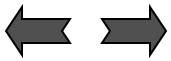
```
TABLE    DB    100    DUP ('*')
```

* آرایه TABLE از نوع بایت و ۱۰۰ عنصری ، با مقدار اولیه

عملوند دستورالعملها

عملوندها دارای انواع مختلف می باشند.بعضی ثابت بوده ، بعضی مشخص کننده ثبا تهای CPU و برخی به حافظه رجوع می نمایند. بسیاری از دستورالعملها دارای دو عملوند می باشند.بطور کلی عملوند اول ، مقصد عملایت را تعیین می کنند و عملوند دوم منبع عملیات.

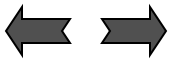
در اسلاید بعد مثالی آورده شده است.



مثال :

MOV AL , ‘*’

که کرکتر * را جایگزین محتوی قبلی ثبات AL می شود.مقصد ثابت نمی تواند باشد ولی منبع می تواند ثابت باشد.



حالت های آدرس دهی

- بلاواسطه
- ثبات
- مستقیم
- دارای مبنا
- دارای اندیس
- دارای مبنا و اندیس


فصل دهم
دستورالعمل های اساسی

فهرست مطالب فصل دهم

انتقال داده ها بین مکان های مختلف حافظه

اغلب کامپیوتر ها بایستی داده ها را از محلی به محل دیگر کپی نمایند.این کار بوسیله دستور MOV انجام می شود. شکل کلی دستور MOV به صورت زیر می باشد :

MOV Destination , Source

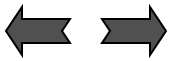
مثال :

```
MOV CX , Count
```

محتوی حافظه COUNT در CX قرار می گیرد.

نکته :

دستوالعمل MOV نمی تواند داده ای را از یک منبع حافظه به یک مقصد حافظه کپی نماید. معمولا برای انجام این کار از یک ثبات میانی استفاده می گردد.

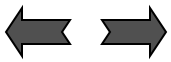


دستورالعمل ADD

شکل کلی آن عبارتست از :

ADD destination , Source

محتوی ***Source*** با محتوی ***destination*** جمع شده نتیجه در ***destination*** قرار می گیرد. این دستورالعمل روی فلگ ها اثر دارد.



مثال :

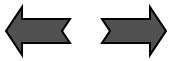
ADD AL , 5

۵ واحد به محتوی AL اضافه می گردد.

مثال :

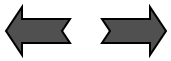
ADD X , BX

به محتوی X ، محتوی BX اضافه می گردد و محتوی BX تغییر نمی کند.



نکته :

در اسمبلی هر دو عملوند یک دستورالعمل نمی توانند از نوع متغییر باشند.



دستورالعمل SUB

شکل کلی آن عبارتست از :

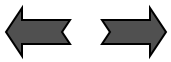
SUB destination , Source

محتوی Source از *destination* کم گردیده نتیجه در *destination* قرار داده می شود و محتوی *Source* تغییر نمی کند.

مثال :

SUB Y , 20

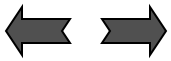
تغییر Y به اندازه ۲۰ واحد کاهش می یابد.



مثال :

SUB AX , X

محتوی X از AX کم شده و نتیجه در AX قرار می گیرد.

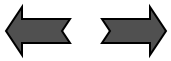


دستورالعمل های DEC , INC

شکل کلی آن عبارتست از :

```
DEC     destination
INC     destination
```

دستورالعمل DEC , INC به ترتیب عملوند مقصد را به اندازه یک واحد کاهش و یا افزایش می دهد.



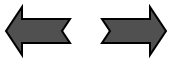
مثال :

INC X

محتوی X یک واحد افزایش می یابد .

DEC AX

محتوی AX یک واحد کاهش می یابد.



نکته :

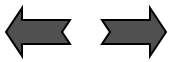
این دستورالعمل روی فلگ ها اثر دارد و عملوند نمی تواند ثابت باشد.

دستورالعمل NEG

شکل کلی آن عبارتست از :

NEG destination

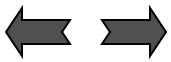
این دستورالعمل عملوند خود را منفی می نماید یعنی مکمل ۲ آن را محاسبه می نماید.



مثال :

```
MOV   AX ,100
NEG   AX
```

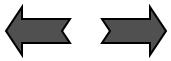
محتوی AX به ۱۰۰- تغییر می یابد.



دستورالعملهای ضرب

اسمبلی دارای دو دستورالعمل ضرب می باشد :

- **IMUL** **عملوندها را بصورت علامتدار در نظر می گیرد.**
- **MUL** **عملوندها را بصورت بدون علامت در نظر می گیرد.**



دستورالعملهای ضرب

شکل کلی آن عبارتست از :

MUL OPR
IMUL OPR

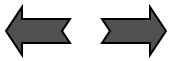
توضیحات :

- عملوند ثابت نمی تواند باشد.
- چنانچه OPR از نوع بایت باشد محتوی OPR در محتوی AL ضرب شده نتیجه در AX قرار می گیرد.
- چنانچه OPR از نوع WORD باشد محتوی OPR در محتوی AX ضرب شده نتیجه در DX : AX قرار می گیرد و محتوی ثبات های AX , DX از بین می رود.
- روی فلگ ها اثر دارد.

مثال :

```
MOV     AL , 10
MOV     X , - 8
IMUL      X
```

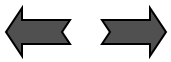
محتوی ثبات AX برابر با 80 - می شود.



دستورالعمل های تقسیم

اسمبلی دارای دو دستورالعمل تقسیم می باشد :

- **IDIV** **عملوند را بصورت علامتدار در نظر می گیرد.**
- **DIV** **عملوند را بصورت بدون عملوند در نظر می گیرد.**



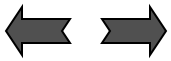
دستورالعمل های تقسیم

شکل کلی آن عبارتست از :

DIV OPR
IDIV OPR

توضیحات :

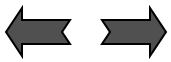
- عملوند ثابت نمی تواند باشد.
- چنانچه OPR از نوع بایت باشد محتویAX بر محتوی OPR تقسیم شده نتیجه در AL قرار می گیرد و باقیمانده تقسیم در AH قرار می گیرد.
- چنانچه OPR از نوع WORD باشد محتوی DX:AX بر محتوی OPR تقسیم شده نتیجه تقسیم در AX قرار می گیرد و باقیمانده در DX قرار می گیرد.

مثال :

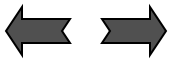
```
X       DB      13
MOV     AX , 134
DIV     X
```

پس از اجرای دستورالعمل های فوق محتوی AL برابر با ۱۰ و محتوی AH برابر با ۴ می باشد.



دستورالعمل های ADC , SBB

در اسمبلی برای جمع و تفریق دو مقدار از نوع **double word** دستورالعملی وجود ندارد. برای این منظور ار دستورالعملهای **SBB , ADC** استفاده می گردد.



دستورالعمل های ADC , SBB

شکل کلی آن عبارتست از :

ADC destination , Source

destination ⟵ destination + Source + CF

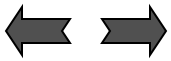
SBB destination , Sounce

destination ⟵ destination - Source – CF

مثال :

ADC X , BY

SBB AX , Y

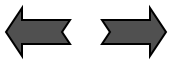


تفریق و جمع دو double word

می خواهیم محتوی دو متغییر X و Y از نوع double word را جمع نموده نتیجه را در Z قرار دهیم :

```
X       DD      ?
Y       DD      ?
Z       DD      ?
MOV     AX , X
ADD     AX , Y
MOV     Z , AX
MOV     AX , X + 2
ADC     AX , Y + 2
MOV     Z + 2 , AX
```

چنانچه بخواهیم محتوی دو متغییر را از هم کم نماییم بایستی در دستورالعملهای فوق ADD را به SUB , ADC را به SBB تبدیل نماییم.



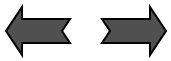

فصل یازدهم
انشعاب و حلقه

فهرست مطالب فصل یازدهم

پرش های غیر شرطی

دستور **JMP** شبیه **goto** در پاسگال می باشد. این دستور دارای فرم زیر است :

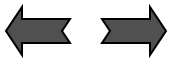
JMP STATEMENT – LABEL



مثال :

به محض اجرای دستورالعمل JMP کنترل بدون هیچ قید و شرطی به دستورالعمل MOV منتقل شده و دستورالعمل MOV اجرا می گردد.

```
        JMP     QUIT
          .
          .
          .
QUIT : MOV   AL , 0
```

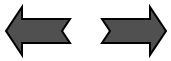


نکته :

در زبان اسمبلی معمولا **STATEMENT – LABEL** را با دستورالعمل **NOP** استفاده می کنند دستورالعمل **NOP** هیچ کاری انجام نمی دهد.

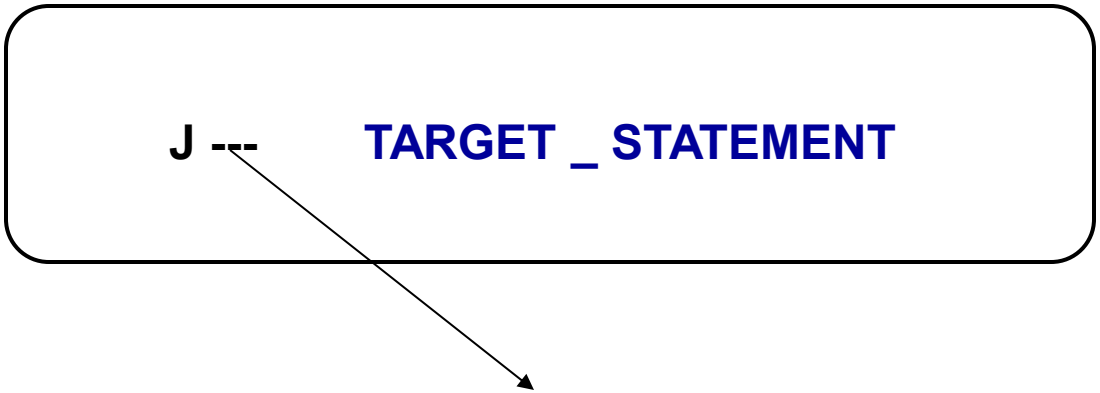
مثال :

```
QUIT :   NOP
         MOV     AL , 0
```

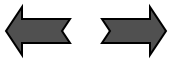


پرشهای شرطی

پرشهای شرطی به برنامه نویس این امکان را می دهد که ساختارهای IF و سایر ساختارهای کنترلی را ایجاد نماید.
شکل کلی بصورت زیر می باشد:



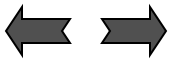
تعیین کننده وضعیتی است که تحت آن ، پرش اجرا می شود. اگر شرط تحقق یابد، پرش صورت خواهد گرفت، در غیر این صورت دستورالعمل بعدی اجرا خواهد گردید.



مثال :

JZ END _ WHILE

این دستورالعمل بدین معنی است که اگر فلگ ZF برابر، یک باشد کنترل به دستورالعمل با برچسب END _ WHILE منتقل می گردد در غیر این صورت کنترل به دستورالعمل بعدی می رود.



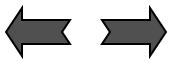
دستورالعمل مقایسه

برای مقایسه دو مقدار از دستورالعمل CMP استفاده می گردد.

شکل کلی عبارتند از :

```
CMP          OPR1 , OPR2
```

دستورالعمل CMP مانند دستورالعمل SUB عمل نموده ولی نتایج در جایی ذخیره نمی شود بلکه محتوی فلگ ها را تغییر می دهد.

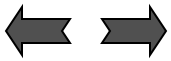


مثال :

CMP AX , 100

که محتوی AX را با ۱۰۰ مقایسه می نماید.

CMP X , ‘$’



نکته :

پس از دستور CMP در صورتی که عملوندها بدون علامت در نظر گرفته شوند از دستورالعملهای پرش شرطی زیر می توان استفاده نمود :

نام دستورالعمل	معنی	فلگها برای پرش
Jg	پرش در حالت بالاتر	CF=0,ZF=0
Jnbe	پرش در حالت پایین یا مساوی	
Jge	پرش در حالت بالاتر یا مساوی	CF=0
Jnb	پرش در حالت پایین تر نبودن	
Jb	پرش در حالت پایین تر	CF=1
Jnge	پرش در حالت پایین تر یا مساوی نبودن	
Jbe	پرش در حالت پایین تر یا مساوی	CF=1 یا ZF=1
Jna	پرش در حالت بالاتر نبودن	

نکته :

پس از دستور CMP در صورتیکه عملوندها با علامت در نظر گرفته شوند از دستورالعملهای پرش شرطی زیر می توان استفاده نمود :

نام دستورالعمل	معنی	فلگها برای پرش
Ja	پرش در حالت بزرگتر	SF=OF,ZF=0
Jnle	پرش در حالت کوچکتر یا مساوی نبودن	
Jae	پرش در حالت بزرگتر یا مساوی	SF=OF
Jnl	پرش در حالت کوچکتر نبودن	
JL	پرش در حالت کوچکتر	SF<>OF
Jnae	پرش در حالت بزرگتر یا مساوی نبودن	
Jle	پرش در حالت کوچکتر یا مساوی	ZF =۱ یا SF<>OF
Jna	پرش در حالت بزرگتر نبودن	

حلقه تکرار For در زبان اسمبلی

در حلقه تکرار **FOR** اغلب تعداد دفعاتی که بدنه حلقه باید اجرا شود از قبل معین می باشد. در زبان اسمبلی این تعداد را بایستی در ثبات **CX** قرار داد و دستورالعمل تکرار دستورالعمل **LOOP** می باشد.

شکل کلی عبارتست از:

```
LOOP        Statement _ label
```

مثال :

```
MOV          CX , 10
LABI :    _
          _
          -
      LOOP  LAB1
```

دستورالعملهای فوق باعث میشود که بدنه حلقه تکرار بار اجرا گردد. هر بار که دستورالعمل **LOOP** اجرا می شود یک واحد از محتوی **CX** کم می شود. شرط خاتمه تکرار این است که تعداد ثبات **CX** برابر با صفر گردد.

دستورالعمل JCXZ

دستورالعمل **JCXZ** یک نوع پرش می باشد. منتهی پرش روی فلگی انجام نمی شود بلکه چنانچه تعداد ثبات **CX** برابر با صفر باشد پرش انجام می شود.شکل کلی بصورت زیر می باشد :

JCXZ Statement _ label

مثال :

```
MOV       CX , 50
LABI:   .
          .
          .
   DEC       CX
   JCXZ              LABEND
   JMP               LABI

LABEND:
```

دستورالعملهای فوق باعث میشود که بدنه دستورالعمل تکرار ۵۰ بار اجرا گردد.

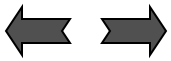
دستورالعملهای LOOPNZ , LOOPZ

دستورالعملهای LOOPZ , LOOPNZ شبیه دستورالعمل LOOP بوده با این تفاوت که این دو دستورالعمل بعد از دستورالعمل CMP در بدنه تکرار استفاده می گردند.

شکل کلی عبارتند از:

```
LOOPZ       Statement _ label
LOOPNZ      Statement _ label
```

```
CX = CX - 1
if (CX <> 0) and (ZF = 1) then
      jump
else
      no jump, continue
```

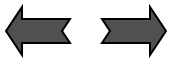


LOOPNZ

چنانچه مقدار جدید در ثبات CX صفر نباشد
و فلگ صفر برابر صفر باشد، دستورالعمل LOOPNZ
به دستورالعملی که در label _ Statement قرار دارد
پرش می کند.

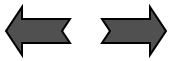
LOOPZ

چنانچه مقدار جدید در ثبات **CX** صفر باشد
و فلگ صفر، یک باشد، دستورالعمل **LOOPZ** به دستورالعملی
که در **Statement _ label** قرار دارد، پرش می کند.



مثال :

```
MOV         CX , 10
FOR :
        .
        .
        .
CMP             BX , 0
LOOPNE              FOR
```



دستورالعمل LEA

این دستورالعمل مخفف کلمات **Load effect address** می باشد.
شکل کلی دستورالعمل بصورت زیر می باشد :

```
LEA        destination, source
```

destination بایستی یک ثبات ۱۶ بیتی بوده و **source** هر گونه رجوعی به حافظه می باشد. این دستورالعمل آدرس **source** را در **destination** قرار می دهد.

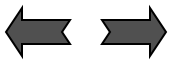
مثال :

LEA BX , X

آدرس متغییر X در ثبات BX قرار می گیرد.

این دستورالعمل معادل دستورالعمل زیر می باشد.

MOV BX , OFFSET

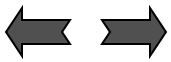



فصل دوازدهم
روال ها

فهرست مطالب فصل دوازدهم

روال ها

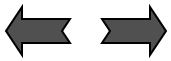
کلمه روال در زبان های برنامه نویسی سطح بالا برای بیان زیر برنامه ای که تقریبا یک واحد کامل می باشد به کار می رود. برای فراخوانی روال نام روال و بدنبال آن لیست آرگومانهای مورد نظر را در داخل پرانتز ذکر می کنند. آرگومانها بایستی متناظر با پارامتر های مجازی روال مزبور باشند. آدرس برگشت به برنامه فراخواننده زیر برنامه در پشته ذخیره می گردد. از طرف دیگر می توان مقادیر ثبات ها در زمان فراخوانی یک زیر برنامه را در پشته ذخیره نمود و در زمان برگشت به برنامه فراخواننده مقادیر ثبات ها را با استفاده از پشته بازسازی نمود. با استفاده از پشته می توان آرگومانها را به یک زیر برنامه انتقال داد و یا مقادیری را از یک زیر برنامه به برنامه ی فراخواننده انتقال داد.



بدنه یک روال

کد یک روال همیشه در داخل یک سگمنت کدی قرار می گیرد. بدنه یک روال در داخل دستورات **PROC** و **ENDP** قرار می گیرد و هر کدام از این دستورالعملها دارای برچسبی است که برابر نام روال مزبور می باشد.

ضمنا دستور **PROC** شامل یکی از مجموعه های **NEAR** یا **FAR** می باشد.



NEAR -- FAR

یک روال NEAR در همان سگمنت کدی که فراخوانی می شود تعریف می گردد و یک روال FAR معمولا در یک سگمنت کدی مجزائی تعریف می شود. روال با دستور کار فراخوانی می شود.

NEAR

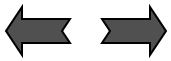
FAR

مثال :

```
INITIALIZE    PROC   NEAR
    .
    .
    .

INITIALIZE            ENDP
```

و برای فرا خوانی :

```
CALL          INITIALIZE
```



دستورالعمل PUSH

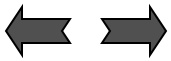
به منظور ذخیره کردن محتوی یک ثبات ۱۶ بیتی یا محتوی یک متغیر از نوع WORD در پشته از دستورالعمل PUSH استفاده می گردد.

شکل کلی دستورالعمل PUSH :

PUSH SOURCE

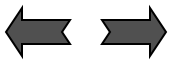
توجه :

بایستی توجه داشت که SOURCE ثابت نمی تواند باشد. در حقیقت محتوی SOURCE بعنوان عنصر رویی پشته قرار می گیرد. این دستورالعمل روی هیچ فلگی اثر ندارد.



مثال :

```
PUSH        AX

Y           DW      ?

PUSH        Y
```



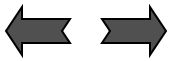
دستورالعمل POP

دستورالعمل POP باعث می شود که عنصر رویی پشته از پشته خارج شود.

فرم دستورالعمل بصورت زیر می باشد :

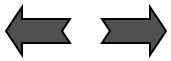
POP destination

مقداری که از پشته خارج می گردد در destination قرار می گیرد.



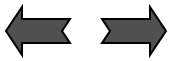
مثال :

```
POP     BX

Y       DW      ?

POP     Y
```



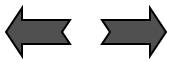
نکته :

این دستورالعمل روی هیچ فلگی اثر ندارد. قبل از عمل POP بایستی مطمئن شویم که پشته تهی نمی باشد.



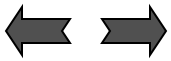
دستورالعملهای PUSHF , POPF

دستورالعمل PUSHF محتوی ثبات فلگ را روی پشته ذخیره می نماید و دستورالعمل POPF عنصر روی پشته را خارج نموده در ثبات فلگ کپی می نماید. با استفاده از دستورالعمل POPF می توان هر تعداد دلخواهی را داخل ثبات فلگ قرار دارد.



نکته :

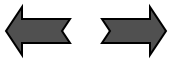
دستور العمل **PUSHF** روی فلگ ها اثر ندارند ولی دستورالعمل **POPF** روی فلگ ها اثر دارند. این دو دستورالعمل فاقد عملوند می باشند.



انتقال مقادیر به یک روال و یا بلعکس

بطرق مختلفی می توان مقادیر را به روال هایی به زبان های اسمبلی یا بلعکس انتقال داد.
دو روش ممکن برای انتقال یک مقدار به اندازه WORD عبارتند از:

- **قرار دادن مقدار مورد نظر در یک ثبات**
- **قرار دادن مقدار مورد نظر روی پشته**




فصل سیزدهم
عملیات رشته ها

فهرست مطالب فصل سیزدهم

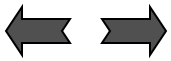
عملیات رشته ها

ریز پردازنده ۸۰۸۸ می تواند بروی رشته ها یی از WORD همانند رشته ای از بایت ها کارکند .
در اسمبلی پنج دستور العمل وجود دارد که برای عملیات برروی رشته ها طراحی شده اند.

۵ دستورالعمل ذکر شده در اسلاید بعد آورده شده اند.

دستورالعملهای رشته

MOVS	انتقال رشته ها	Move string
CMPS	مقایسه رشته ها	Compare string
SCAS	پرش رشته ها	Scan string
STOS	ذخیره رشته ها	Store string
LODS	بار کردن رشته ها	Load string

دستورالعمل MOVS

دستورالعمل **MOVS** برای کپی کردن یک رشته از یک موقعیت حافظه به موقعیت دیگر بکار می رود.

دستورالعمل CMPS

دستورالعمل **CMPS** برای مقایسه محتویات دو رشته مورد استفاده قرار می گیرد.

دستورالعمل SCAS

بوسیله **SCAS** می توان در یک رشته به دنبال یک مقدار معیین گشت.

دستورالعمل STOS

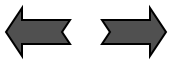
دستورالعمل **STOS** می تواند برای ذخیره کردن یک مقدار جدید در رشته بکار رود.

دستورالعمل LODS

دستورالعمل LODS یک مقدار را از یک رشته بدست می آورد.

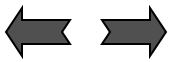
استفاده از دستورالعملهای رشته ای

هر دستورالعمل رشته ای روی یک رشته منبع، رشته مقصد یا هر دو عمل می کند. هر بایت یا WORD این رشته ها به وسیله دستورالعملهای پردازش رشته، یک به بک مورد پردازش قرار می گیرند. عنصر مبدا بوسیله ثبات های DS , SI و عنصر مقصد بوسیله ثباتهای DI , ES مشخص می شود.



استفاده از دستورالعملهای رشته ای

عنصر مبدا بایستی در سگمنت DATA و عنصر مقصد در سگمنت extra تعریف نمـوده و ثبـات های SI , DI بترتیب به ابتدای رشته های مبدا و مقصد اشاره می نمایند. فلگ DF مشخص کننده جهت پردازش از ابتدای رشته بطرف انتها با بلعکس می باشد.



استفاده از دستورالعملهای رشته ای

CLD **DF** ← **0** **از ابتدا بطرف انتهای رشته**

STD **DF** ← **1** **از انتها بطرف ابتدای رشته**

پیشوند های تکرار

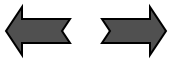
پیشوند های تکرار

پیشوند	شرط اتمام
REP	CX == 0
REPE	CX == 0 and ZF==0
REPZ	CX == 0 and ZF==0
REPNZ	CX == 0 and ZF==1
REPNE	CX ==0 and ZF==1

مثال :

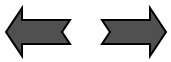
```
MOV     SI , OFFSET     SOURCE_STR
MOV     DI, OFFSET      DEST_STR
CLD
MOV     CX, COUNT
REP     MOVSB
```

پسوند B (در MOVSB) مشخصه بایت و پسوند W مشخصه WORD می باشد.



دستورالعمل ذخیره سازی STOS

دستورالعمل ذخیره سازی رشته ای STOS یک بایت یا یک کلمه از ثبات AL یا رشته AX به یک عنصر رشته ای مقصد کپی می کند. این دستوربرروی هیچ فلگی اثر نمی گذارد. بنابراین وقتی که این دستورالعمل با پیشوند rep تکرار شود یک مقدار را در موقعیت های متوالی یک رشته کپی می کند.



مثال :

برنامه زیر ، کارکتر فاصله خالی را در اولین ۳۰ بایت رشته string ذخیره می نماید.

```
MOV     CX , 30                 ;30 bytes
MOV     AL , ' '                ;character to store
MOV     DI , OFFSET string      ;address of string
CLD                             ;forward
REP     stosb                   ;store space
```

LODSB

```
LEA   SI, SOURCE_STR
LODSB
```

COMPS

برای مقایسه عنصر اول آرایه

```
MOV SI, OFFSET SOURCE_STR
MOV DI, OFFSET DEST_STR
CMPSB
```

برای مقایسه دو رشته

```
MOV SI, OFFSET STRG1
MOV DI, OFFSET STRG2
MOV CX, LENGTHSTRG3
NEXT: CMPSB
JNE EXIT
LOOP NEXT
JMP SAME
EXIT:
....
SAME:
....
```

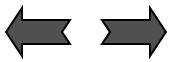
```
LEA SI, STR1
LEA DI, STR2
MOV CX, 100
CLD .
REPE CMPSB
JZ FOUND
. …
FOUND .
. ….
```

SCAS

```
STRG DB 50 DUP (?)
MOV AL , '*'
MOV CX, 50
LEA DI, STRG
CLD
REPNE SCASB
```

قطعه برنامه زیر رشته String را در نظر می گیرد و بدنبال کاراکتر & میگردد که به کارکترا فاصله تبدیل نماید.

```
STRLEN EQU 15
STRING DB 'The time & is now'
CLD
MOV    AL, '&'
MOV    CX, STRLEN
LEA    DI, STRING
REPNE   SCASB
JNZ   NOTFOUND
DEC  DI
MOV   BYTE PTR[DI], 20H
…..
NOTFOUND:
……
```




فصل چهاردهم
سایر حالت های آدرس دهی

فهرست مطالب فصل چهاردهم

حالت های آدرس دهی

عملوند های دستورالعمل های اسمبلی به سه گروه عمده تقسیم می شوند :

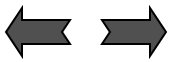
- بلا واسطه
- ثبات
- حافظه

بلا واسطه

عملوند های بلاواسطه مقادیری هستند که در داخل دستورالعمل ها قرار می گیرند.

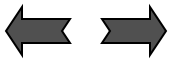
ثبات

عملوندهای ثبات شامل مقادیری هستند که از یک ثبات برداشته شده یا نتیجه دستورالعمل در یک ثبات مقصد قرار می گیرد.

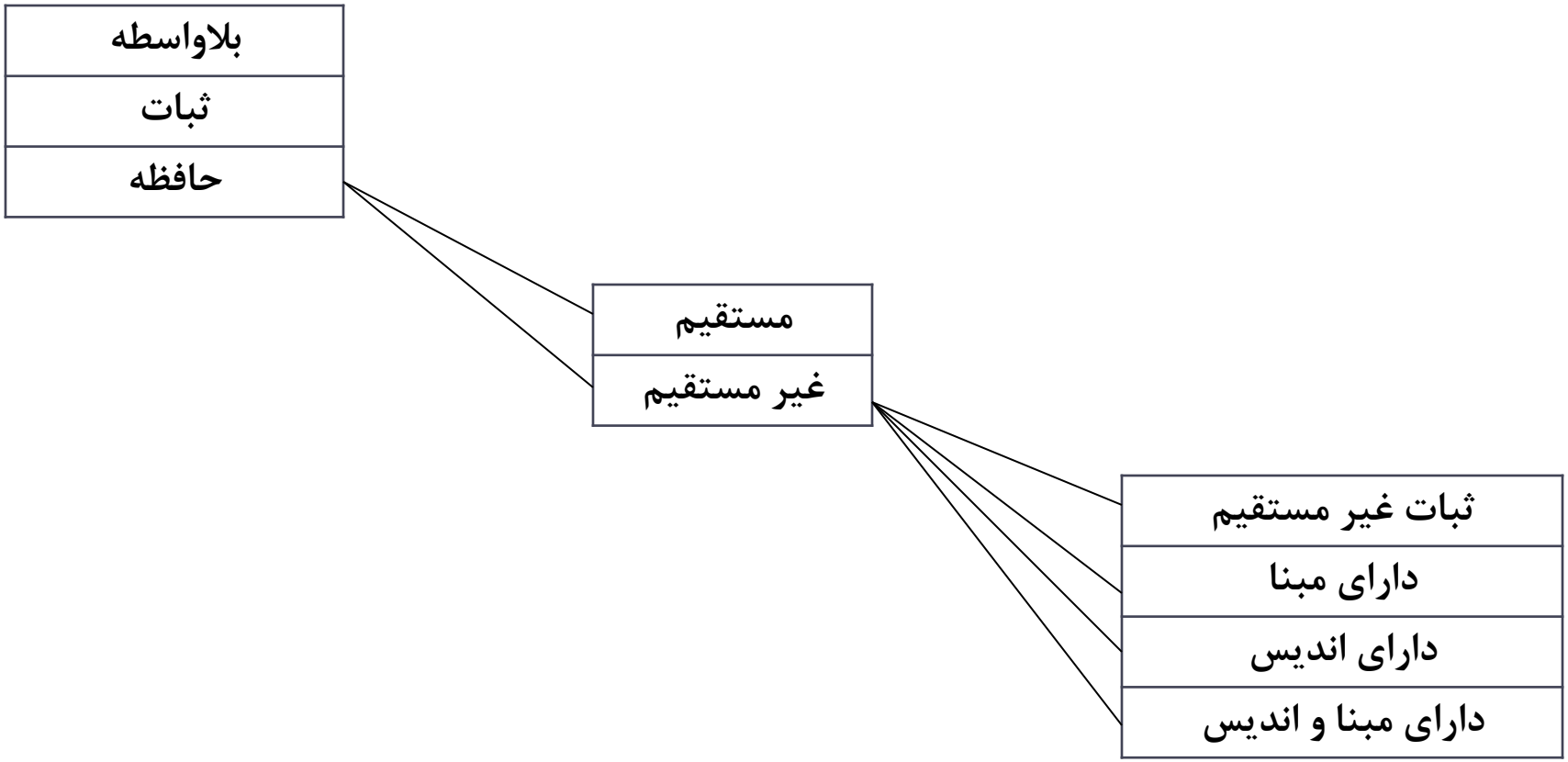


حافظه

عملوند های حافظه به دو گروه مستقیم و غیر مستقیم تقسیم می شوند. آفست یک عملوند مستقیم در داخل دستورالعمل قرار می گیرد. آفست یک عملوند با استفاده از یکی از حالت های غیرمستقیم ، از محتوی یک یا دو ثبات و (بعضی مواقع) یک مقدار جابجایی در داخل دستورالعمل قرار دارد، محاسبه می شود.



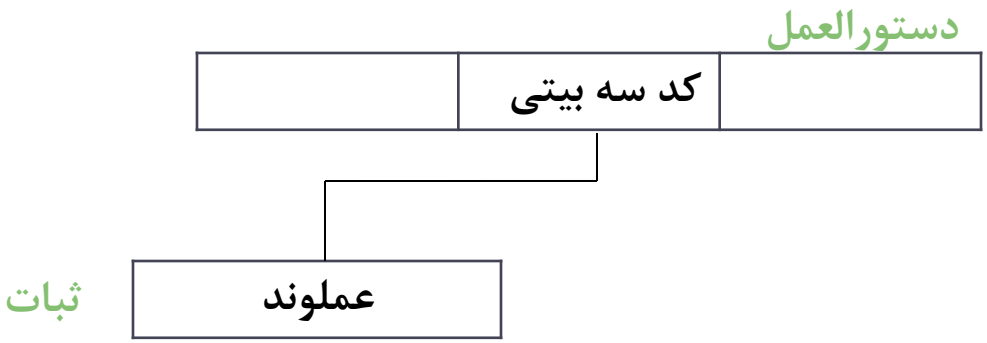
حالتهای مختلف آدرس دهی



حالتهای مختلف آدرس دهی

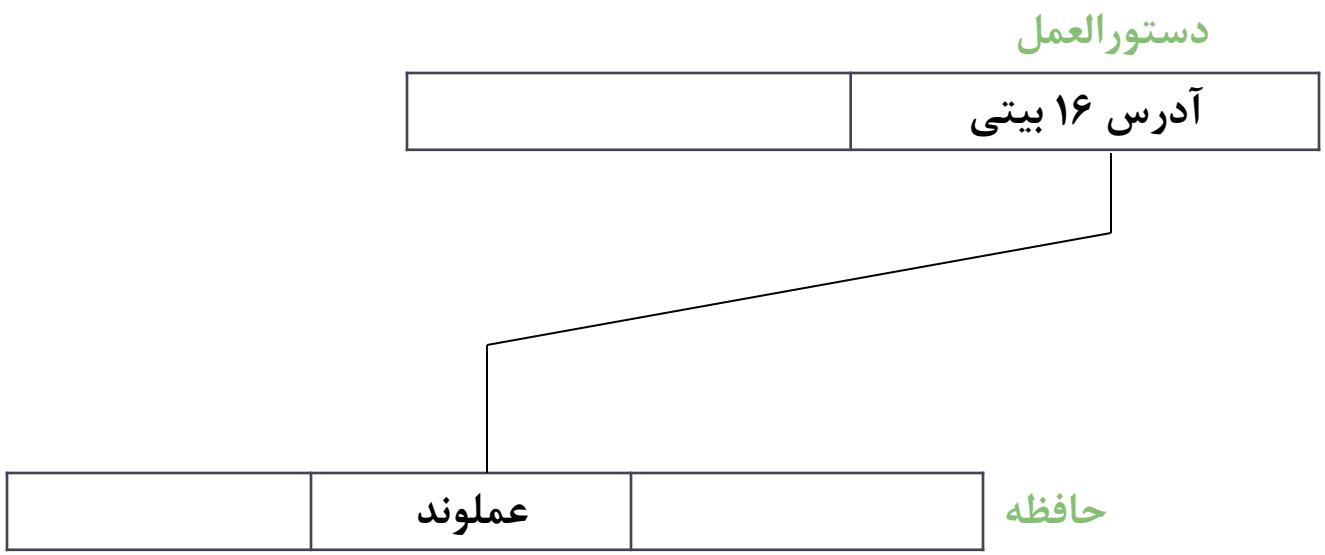
a) بلاواسطه

b) ثبات



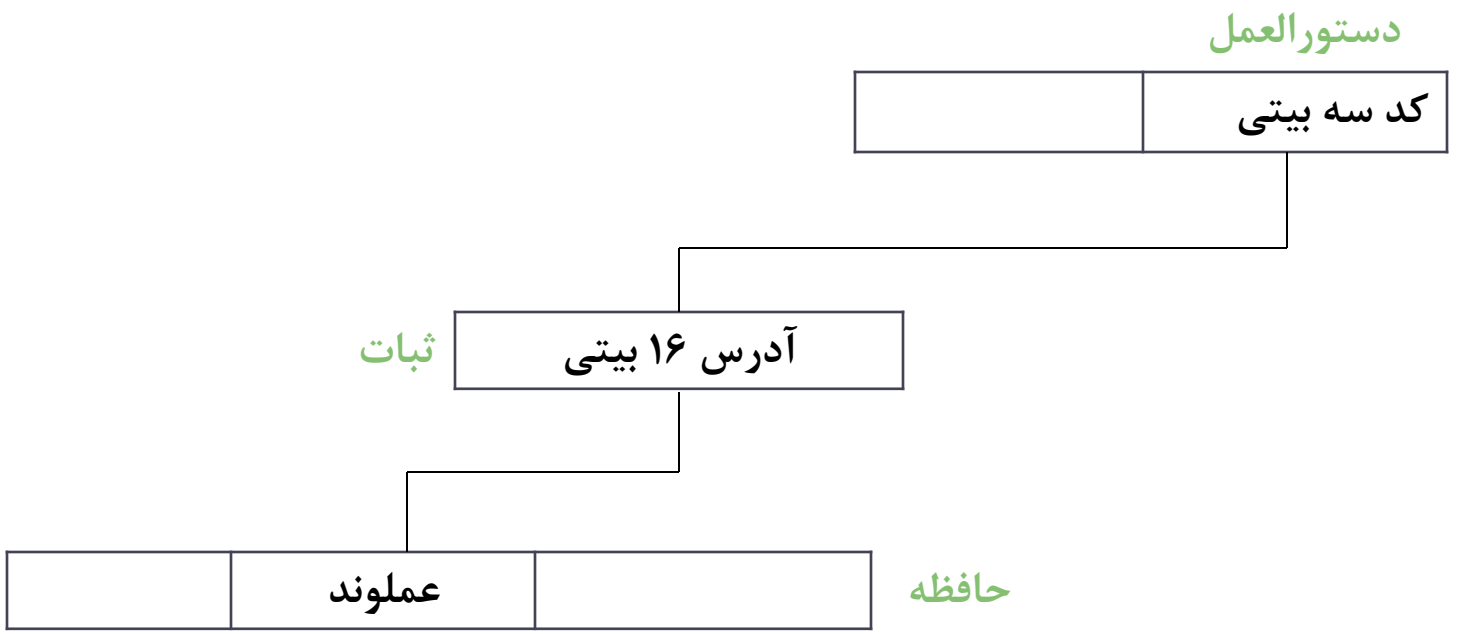
حالتهای مختلف آدرس دهی

C) مستقیم



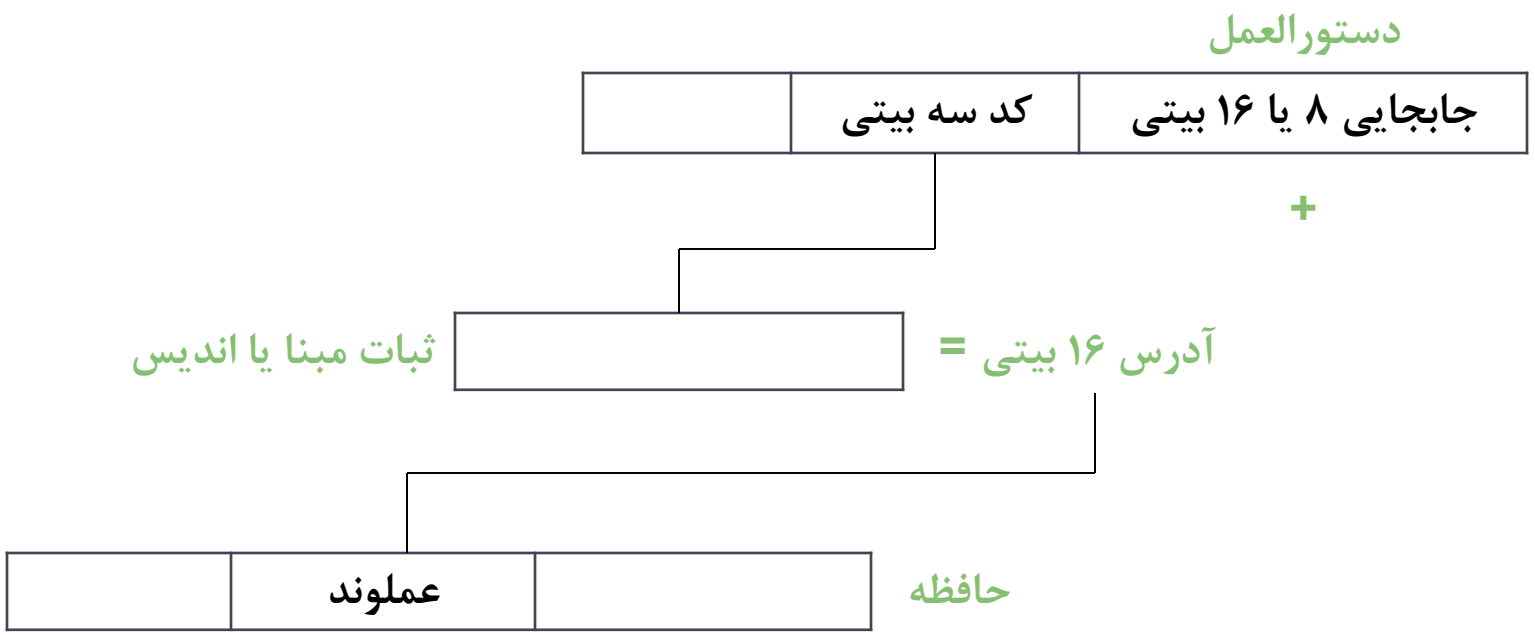
حالتهای مختلف آدرس دهی

d) ثبات غیر مستقیم



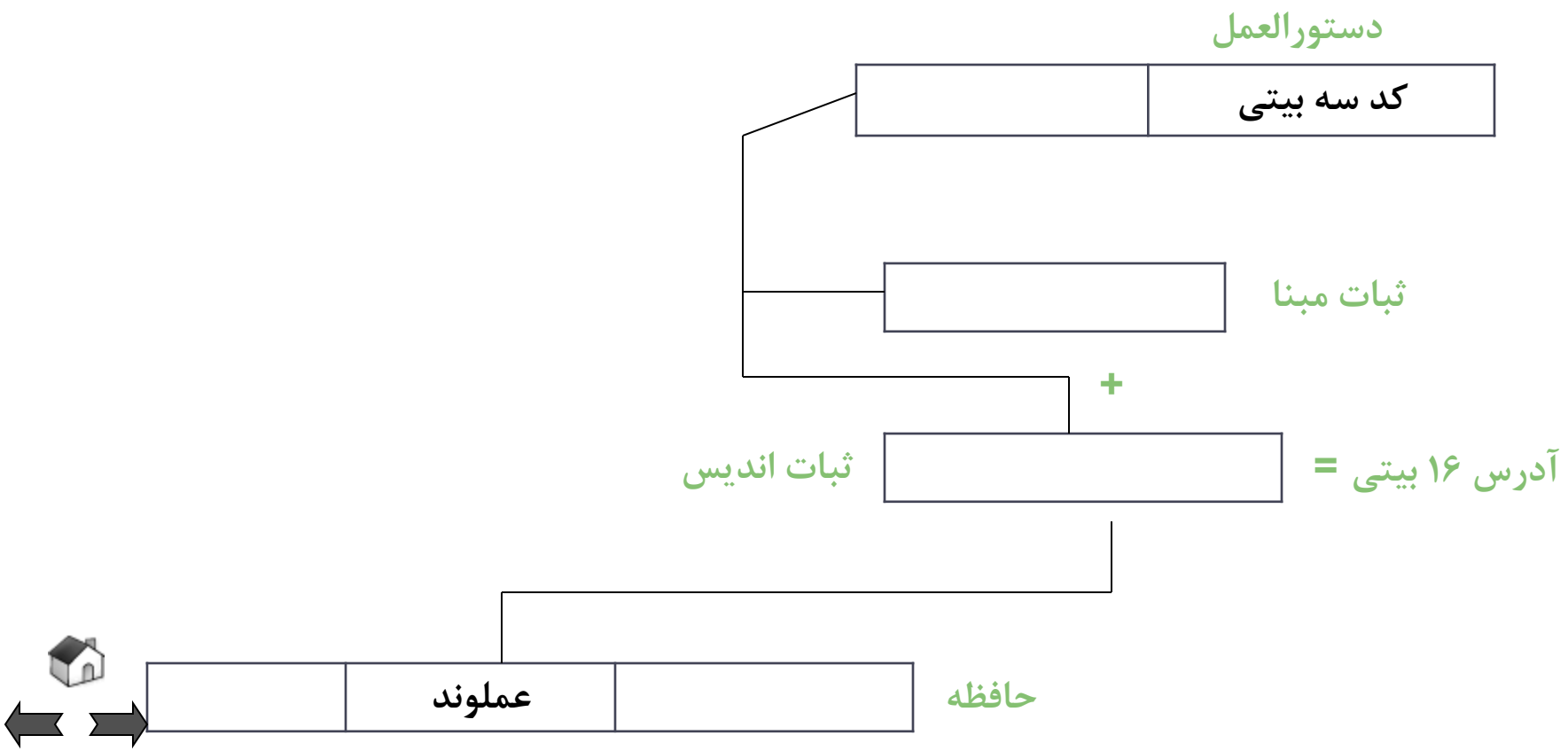
حالتهای مختلف آدرس دهی

e) دارای مبنا یا دارای اندیس



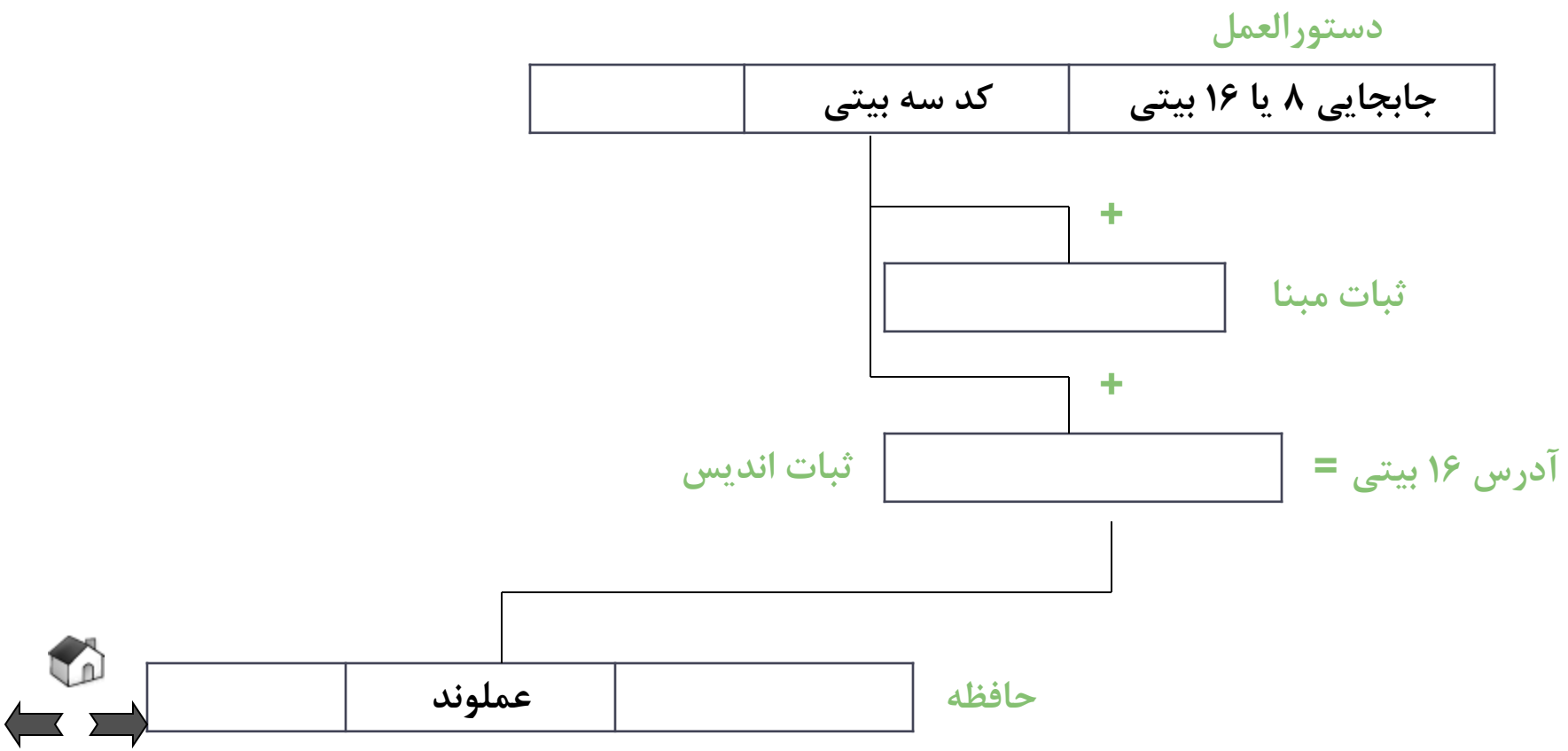
حالتهای مختلف آدرس دهی

f) دارای مبنا و اندیس (بدون جابجایی)



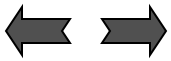
حالتهای مختلف آدرس دهی

g) دارای مبنا و اندیس (با مقدار جابجایی)



ساختارها

یک ساختار مجموعه ای از عناصر دارای انواع مختلف که نام مشترک دارند، می باشد. عناصر یک ساختار را می توان با استفاده از نام ساختار و نام فیلد عنصر مورد نظر دستیابی نمود.



مثال :

Partno	از نوع word
Description	رشته۲۰ کرکتری
Quantity	از نوع word

در اسمبلی این ساختار را می توان به صورت زیر تعریف نمود:
ساختار در حقیقت یک سگمنت کد می باشد.

```
PART        STRUC
PARTNO          DW          ?
DESCRIPTION             DB      20      DUP (?)
QUANTITY        DW          ?
PART        ENDS
```

حال می توان SPARE را بصورت زیر از نوع رکورد فوق تعریف نمود :

```
SPARE           PART
```

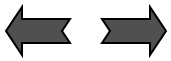
این دستورالعمل ۲۴ بایت حافظه را برای SPARE تشخیص می دهد.


فصل پانزدهم
دستکاری بیت ها

فهرست مطالب فصل پانزدهم

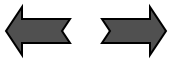
دستکاری بیتها

ریز پردازنده ۸۰۸۸ و اکثر CPU های دیگر می توانند دستورالعملهائی را اجرا نمایند که عملیات بولی را بطور همزمان بر روی چندین زوج بیت انجام می دهند. هر چند که دستورالعملهای دستکاری بیت ها خیلی اولیه هستند، ولی بطور گسترده در زبان اسمبلی مورد استفاده قرار می گیرند.



عملیات منطقی

یک کامپیوتر دارای مدارات مجتمع بسیاری می باشد که آن را قادر می سازد کارهای آنرا انجام دهد. هر تراشه حاوی چند هزار گیت (gate) منطقی می باشد که هر کدام یک مدار اولیه برای انجام عملیات بولی بر روی بیتهائی که بوسیله حالت های الکترونیکی عرضه می گردند می باشد که معمولا در یک PC , CPU پیچیده ترین مدار مجتمع می باشد.



تعاریف عملیات منطقی AND

بیت دو and بیت یک	بیت دو	بیت یک
0	۰	۰
0	۱	۰
0	۰	۱
1	۱	۱

تعاریف عملیات منطقی OR

بیت یک	بیت دو	بیت دو Or بیت یک
۰	۰	۰
۰	۱	۱
۱	۰	۱
۱	۱	۱

تعاریف عملیات منطقی XOR

بیت دو XOR بیت یک	بیت دو	بیت یک
۰	۰	۰
۱	۱	۰
۱	۰	۱
۰	۱	۱

تعاريف عمليات منطقي NOT

بيت NOT	بيت
١	٠
٠	١

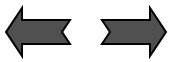
دستور العملهای منطقی

۸۰۸۸ دارای دستورالعملهای **AND ; OR ; XOR, NOT** می باشد.

فرم این دستورالعملها عبارتند از :

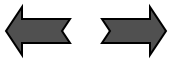
AND	منبع و مقصد
OR	منبع و مقصد
XOR	منبع و مقصد
NOT	مقصد

این دستورالعمل بیتهای صفر را به یک و بیت های یک را به صفر تبدیل می نماید و روی فلگ ها اثر دارد.



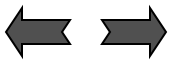
مثال :

```
AND     AX , Y
XOR     Y ,
  2BAFH
```



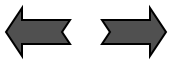
عمل پوشش (MASK)

برای تغییر مقادیر بیت های یک بایت یا WORD خاصی می توان اینکار را با استفاده از عمل MASK انجام داد. بعنوان مثال فرض کنید می خواهیم بیت های فرد ثبات AL را به یک تبدیل نمائیم. برای اینکار یک بایت بعنوان MASK ایجاد نموده که بیت های شماره فرد آن یک و بیت های شماره زوج آن صفر باشد. سپس عمل MASK را روی AL انجام می دهیم.



مثال :

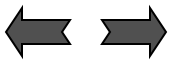
```
MOV   MASK , 10101010B
OR      AL , MASK
```



استفاده از mask

با ایجاد یک mask می توان بصورت زیر عمل نمود :

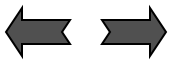
- **برای صفر کردن بیت های یک عملوند از نوع بایت یا Word بایستی عمل AND بین آن عملوند و mask انجام داد.**
- **برای یک کردن بیت های یک عملوند از نوع بایت یا Word بایستی عمل OR بین آن عملوند و mask انجام داد.**
- **برای معکوس نمودن بیت های یک عملوند از نوع بایت یا Word بایستی عمـل XOR بین آن عملوند و mask انجام داد.**



مثال :

برای معکوس نمودن بیت های ۲ , ۵ , ۶ ثبات AL بصورت زیر عمل می نمائیم :

```
MOV     MASK , 01100100B
XOR     AL , MASK
```



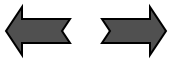
دستورالعمل TEST

دستورالعمل TEST مانند دستورالعمل AND عمل نموده با این تفاوت که هیچ کدام از عملوند ها را تغییر نمی دهد.

شکل کلی این دستورالعمل بصورت زیر می باشد :

مثال:

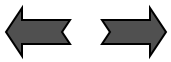
```
TEST    AX , Y
TEST    AL , DH
```



تست محتوی یک عملوند

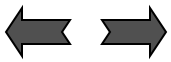
به منظور تست اینکه آیا بیت های شماره ۷ , ۶ , ۳ , ۰ ثبات AL برابر یک می باشند یا خیر می توان بصورت زیر عمل نمائیم.

```
NOT       AL
MOV       MAKS , 11001001B
TEST      AL , MAKS
JZ        YES
.
.
.
YES:
    .
    .
    .
```



دستورالعملهای شیفت

دستورالعملهای شیفت، بیت های واقع در موقیعت داده شده بوسیله عملوند مقصد را بطرف چپ یـا راست حرکت می دهند. جهت شیفت می تواند از آخرین کرکتر نام دستورالعمل شیفت تشخیص داده شود. R برای راست و L برای چپ. عملوند می تواند از نوع بایت یا WORD باشد.



نکته :

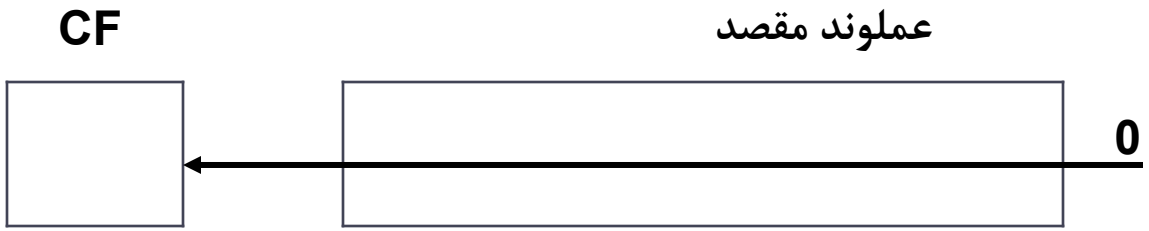
چنانچه فقط یک بیت شیفت داده شود خود مقدار یک و در صورتیکه بیش از یک بیت شیفت داده شود، تعداد را در ثبات CL قرار می دهیم.

دستورالعمل SHL

از این دستورالعمل برای شیفت منطقی بیت ها به سمت چپ استفاده می گردد.

شکل کلی بصورت زیر می باشد :

مثال :




```
MOV         CL , 3
MOV         DL , 4BH
SHL         DL , CL
```

چنانچه محتوی CF برابر با یک باشد نتیجه اجرای دستورالعملهای فوق بصورت زیر می باشد.

```
CL          3
DL          58H
CF          0
```

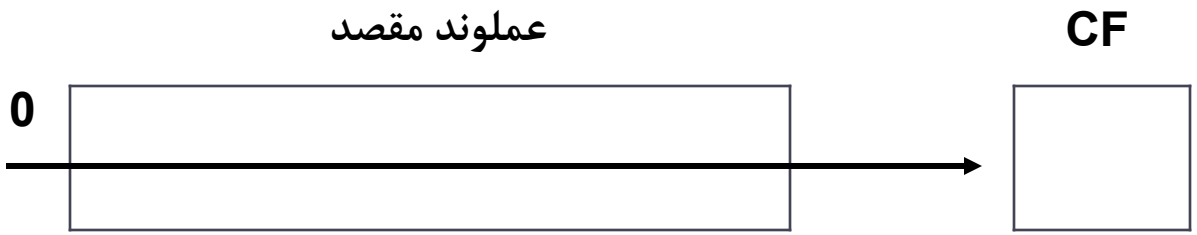
دستورالعمل SHR

ازا ین دستورالعمل برای شیفت منطقی بیت ها به سمت راست استفاده می شوند.

شکل کلی بصورت زیر می باشد.

SHR تعداد و مقصد

مثال :




```
MOV         CL , 3
MOV         DL , 4BH
SHR         DL , CL
```

چنانچه محتوی CF برابر با یک باشد پس از اجرای دستورالعمل های فوق
مقدار ثبات ها برابر است با

```
CF          0
CL          3
DL          09H
```

دستورالعمل SAL

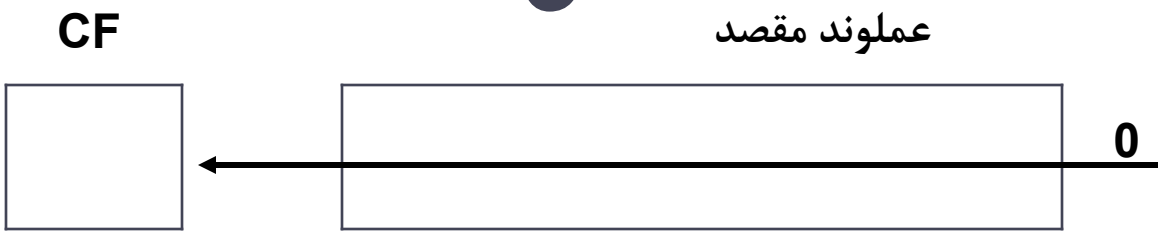
از این دستورالعمل برای شیفت محاسباتی بیت ها به سمت چپ استفاده می گردد.

شکل کلی عبارتست از :

SAL	**تعداد و مقصد**

کار این دستورالعمل شبیه دستورالعمل SHL می باشد.

مثال :




```
MOV     CL , 3
MOV     DL , 4BH
SAL     DL , CL
```

چنانچه مقدار CF برابر با یک باشد ،
مقادیر ثباتها پس از اجرای دستورالعمل ها بصورت زیر می باشد.

```
CL      3
DL      58H
CF      0
```

دستورالعمل SAR

از این دستورالعمل برای شیفت محاسباتی به سمت راست استفاده می گردد.

شکل کلی عبارتست از :

این دستورالعمل شبیه دستورالعمل SHR می باشد با این تفاوت که بجای صفر از بیت MSB استفاده می گردد.

مثال :

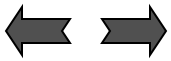

```
MOV     CL , 3
MOV     BL , 0B4H
SAR     BL , CL
```

با توجه به اینکه مقدار CF=1 باشد پس از اجرای دستورالعملهای فوق مقادیر ثبات ها برابرند با

```
CL          3
BL      11110110B
CF          1
```

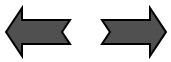
ضرب و تقسیم

دستورالعمل های شیفت دارای کاربردهای زیادی می باشند. بعضی از ریز پردازنده هـا اصـلا دارای دستورالعملهای ضرب و تقسیم نمی باشند. وقتی قرار باشد که عدد در دو ضرب شود، یک شیفت تک بیتی به طرف چپ عدد اولیه می تواند حاصل ضرب صحیح را در اختیار بگذارد. یک عمل شیفت تک بیتی به طرف راست می تواند بصورت موثری برای تقسیم کردن یک عملوند بدون علامت بر ۲ ، مورد استفاده قرار گیرد.



دستورالعملهای چرخش

دستورالعملهای چرخش خیلی شبیه دستورالعملهای شیفت هستند. در دستورالعملهای شیفت، بیت ها از یک طرف شیفت داده شدهو بدور ریخته می شوند، در حالیکه جاهای خالی از طرف دیگر با صفر (یا عددی که در شیفت ریاضی بسمت راست روی اعداد منفی) پر می شوند. ولی در دستورالعملهای چرخش، بیتهایی که از یک طرف یه بیرون شیفت داده می شوند، از طرف دیگر فضاهای خالی را پر می کند.

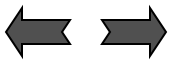


شکل کلی دستورالعمل های چرخش

شکل کلی دستورالعمل های چرخش بصورت زیر می باشد:

چرخش یک بیت ;	**۱ و مقصد**	**r - -**
چرخش باندازه محتوی CL ;	**CL و مقصد**	**r - -**

عملوند مقصد می تواند از نوع بایت یا از نوع WORD باشد. وقتی که یک بیت از یک طرف بیرون می رود از طرف دیگر وارد می شود. علاوه بر این آخرین بیت که بطرف دیگر عملوند کپی می شود در فلگ CF نیز منعکس می گردد.



دستورالعمل ROL

این دستورالعمل به تعداد داده شده بیت به سمت چپ چرخش می دهد.

شکل کلی این دستورالعمل بصورت زیر میباشد:

ROL	۱ و مقصد
ROL	CL و مقصد

مثال :

```
MOV     DX , 0D25EH
ROL     DX , 1
```

پس از اجرای دستورالعملهای فوق مقادیر عبارتند از :

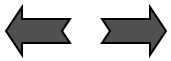
```
DX      1010010010111110113
CF      1
```

دستورالعمل ROR

این دستورالعمل به تعداد داده شده بیت به سمت راست چرخش می دهد.

شکل کلی این دستورالعمل به شکل زیر می باشد :

ROR	۱ و مقصد
ROR	CL و مقصد



مثال :

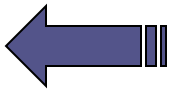

```
MOV     CL , 3
MOV     AL , 0B5H
ROR     AL , CL
```

پس از اجرای دستورالعملهای فوق نتایج عبارتند از

```
CL      3
CF      1
AL      B6H
```

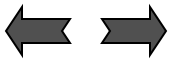

فصل شانزدهم
وقفه های ورودی / خروجی

فهرست مطالب فصل شانزدهم



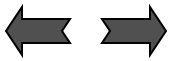
وقفه ها و ورودی / خروجی

سیستم های عامل معمولا سرویسهای گوناگونی را ارائه می دهند که می توان در برنامه های اسمبلی از آنها استفاده نمود. بعضی از این سرویس ها برای ورودی و خروجی و برخی برای اهداف دیگر می باشند. همچنین می توان ورودی و خروجی بدون استفاده از سرویسهای DOS یعنی استفاده از دستورالعملهائی که در گاههای I /O یک PC را بطور مستقیم دستیابی می کنند ، انجام داد.



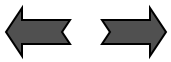
تله یا استثناء

برای استفاده ازسرویسهای سیستم عامل، برنامه نویس می تواند از یک روال یا ماکروی (یک ماکرو معمولا توسط اسمبلر بدنباله ای از دستورالعملهایی که شامل فراخوانی یک روال مـی باشـد، بسـط و توسعه می یابد) استفاده نماید. بعضی مواقع از فراخوانی های معمولی روال ها استفاده می شود. ولـی DOS و بسیاری از سیستم عامل های دیگر از نوع بخصوص فراخوانی روال ها استفاده مـی کننـد ; یک وقفه نرمافزاری که بعضی مواقع آن را یک تله یا یک استثناء می گویند.



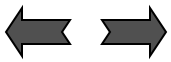
دستورالعمل int و جدول بردار وقفه

برای فعال کردن یک وقفه می توان از دستورالعمل int پردازنده ۸۰۸۸ استفاده نمود. بعلاوه، بعضی وقفه ها توسط خود سخت افزار PC تولید می شوند.



وقفه

سیستم ۸۰۸۸ می تواند شامل ۲۵۶ وقفه مختلف باشد. یک وقفه عملا دارای یک پردازنده وقفه می باشد که بلوکی از کد می باشد که تقریبا مانند یک روال معمولی می باشد. یک پردازنده وقفه بجای دستورالعمل **Call** توسط دستورالعمل**int** فراخوانی می شود. برای برگشتن از یک پردازنده وقفه، بجای استفاده از دستورالعمل **ret** برای یک روال معمولی از دستورالعمل **(interrupt return) iret** استفاده می شود.

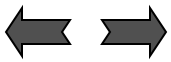


دستورالعمل int

دستورالعمل int دارای قالب زیر است :

int interrupt _ type

interrupt _ type برابر عدد صحیح از ۰ تا ۲۵۵ می باشد. کد هدف دستورالعمل int به طول ۲ بایت می باشد.

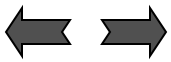


اجرای دستورالعمل int

وقتی دستورالعمل int اجرا می شود، ابتدا محتویات ثبات نشانه ها را روی پشته اضافه کرده و سپس نشانه های IF (نشانه فعال سازی وقفه) و TF (نشانه تله) خاموش می گردد. (اگر نشانه IF برابر ۰ باشد ، در این صورت پردازنده ۸۰۸۸ تمام وقفه های سخت افزاری بجز وقفه های پوشش ناپذیر (nonmaskable interrupt) را صرفه نظر می کند. اگر نشانه TF برابر ۱ باشد در این صورت پردازنده ۸۰۸۸ در حالت تک گام عمل می کند.پاک کردن نشانه TF این حالت اشکال زدائی را غیر فعال می کند.)

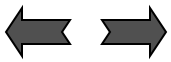
اجرای دستورالعمل int

مراحل نهائی اجرای یک دستورالعمل int مانند فراخوانی یک روال دور می باشد- شماره سگمنت واقع در ثبات CS روی پشته اضافه شده و شماره سگمنت کد جدید بداخل ثبات CS بار شده و آفست دستورالعمل بعدی که در ثبات IP قرار دارد روی پشته اضافه شده و آفست دستورالعمل جدید بداخل ثبات IP بار می گردد. به غیر از نشانه های IF و TF نشانه های دیگری توسط دستورالعمل int تغییر پیدا نمی کنند.



دستورالعمل برگشت از وقفه

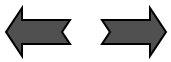
دستورالعمل برگشت از وقفه یعنی iret بدون عملوند می باشد. این دستورالعمل بطول یک بایت دارای کد عمل CF بوده و تعداد سیکلهای زمانی اجرای آن برابر ۴۴ می باشد. این دستورالعمل ابتدا مانند یک دستورالعمل برگشت دور عمل می کنند یعنی این که ثبات های IP , CS را از روی پشته برداشته و سپس مقادیر نشانه ها را از روی پشته بر می دارد در نتیجه تمام نشانه ها برابر مقادیری توسط دستورالعمل int روی پشته ذخیره شده بودند، می گردد.



از وقفه در موارد متعددی استفاده می شود .
وقفه صفر بطور اتوماتیک در صورت بوجود آمدن خطای تقسیم بر صفر توسط پردازنده ۸۰۸۸ فراخوانی می شود.

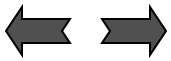
در خواست توابع DOS

سیستم عامل DOS با استفاده از وقفه نوع 21h و 10hتوابع متعددی را در اختیار استفاده کننده قرار می دهد . اغلب این توابع در مورد ورودی /خروجی دستگاههای مختلف می باشد. بسیاری در مورد عملیات مربوط به فایل های دیسک می باشند. بعضی از این توابع برای اهداف ویژه های می باشند (مانند تعیین تاریخ سیستم).



در خواست توابع DOS

تمام توابع فراخوانی شده توسط int 21h یک پردازنده وقفه معینی را فراخوانی می کنند. تابع مورد نظر با قرار دادن شماره تابع مربوطه در ثبات AH انتخاب می شود. در اغلب توابع ، داده های دیگری به روال پردازنده وقفه ارسال شده و یا اطلاعاتی برگردانده می شوند.

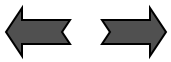


مثال :

برای پایان بخشیدن به اجرای برنامه از تابع $4C_{16}$ سیستم DOS استفاده می شود.

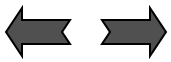
```
Quit:   mov    al,0          ;return    code   0
        mov    ah,4ch        ;DOS function  to  return
        int    21h           ;interrupt for  DOS  services
```

تابع $4C_{16}$ یک فرایند (process) را خاتمه داده (برنامه نمونه ای از یک فرآیند است) ، تمام فایلهائی که فرآیند مزبور باز کرده بسته ، کنترل اجرا را به فرآیند پدر انتقال داده (اگر فرآیند مزبور یک برنامه باشد، کنترل اجرا به سیستم DOS انتقال می یابد) و یک کد خروج یا کد برگشت به فرآیند پدر برگردانده می شو



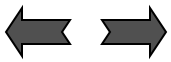
توابع ساده DOS

شماره تابع	عمل	پارامترهای انتقالی	پارامترهای برگشتی
1	گرفتن یک کاراکتر (با ظاهر شدن روی صفحه نمایش)	به DOS بدون پارامتر	از DOS کاراکتر خوانده شده در AL
2	نمایش دادن یک کاراکتر	کاراکتر مورد نظر در DL	بدون پارامتر
5	چاپ یک کاراکتر	کاراکتر در DL	بدون پارامتر

توابع ساده DOS

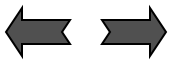
شماره تابع	عمل	پارامترهای انتقالی	پارامترهای برگشتی
8	گرفتن یک کاراکتر (بدون ظاهر شدن روی صفحه نمایش)	بدون پارامتر	کاراکتر در AL
9	نمایش دادن یک رشته	DS:DX برابر آدرس رشته	بدون پارامتر
$0A_{16}$	خواندن یک رشته	DS:DX برابر آدرس بافر ماکزیمم تعداد کاراکترها در بایت	رشته مورد نظر در بافر تعداد واقعی کاراکترها در بایت دوم بافر
$4C_{16}$	پایان دادن به یک فرآیند	کد بازگشتی در AL	بدون پارامتر



ورودی / خروجی فابلهای پیاپی با استفاده از DOS

اکثر توابع DOS که از طریق int 21h فراخوانی می شوند، در مورد عملیات روی فایلهای دیسک می باشند. از جمله توابع زیر می باشند :

- **تشکیل دادن یک فایل جدید.**
- **باز کردن یک فایل (آماده ساختن یک فایل برای خواندن یا نوشتن)**
- **بستن یک فایل (بافرهای حافظه را روی دیسک نوشته و دایر کتوری فایل را بروز رسانی می کند)**
- **خواندن داده از یک فایل**
- **نوشتن داده روی یک فایل**
- **حذف کردن یک فایل**



توابع فایل DOS

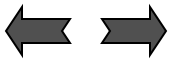
```
mov cx, 0 ; normal - no attributes. mov cx, 1 ; read-only. mov cx, 2 ; hidden. mov cx, 4 ;
system mov cx, 7 ; hidden, system and read-only! mov cx, 16
```

شماره تابع	عمل	پارامترهای انتقالی	پارامترهای برگشتی
$3C_{16}$	تشکیل دادن یک فایل جدید	افست نام فایل در DS : DX صفات مشخصه فایل در ثبات CX	if CF=1 then AX error code else AX file handle
$3D_{16}$	باز کرد ن یک فایل	افست نام فایل در DS : DX کد حالت دستیابی فایل (خواندن یا نوشتن) در ثبات AL	if CF=1 then AX error code else AX file handle

توابع فایل DOS

شماره تابع	عمل	پارامترهای انتقالی	پارامترهای برگشتی
$3E_{16}$	بستن یک فایل	ثبات **BX** شامل هندل فایل	**if CF=1 then** **AX error code**

$3F_{16}$	خواندن از یک فایل	ثبات **BX** شامل هندل فایل
		ثبات **CX** شامل بایت های مورد نظر برای خواندن و آدرس بافر مقصد در **DS:DX**
			if CF=1 then
			AX error code
			else
			AX bytes read

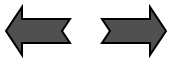


توابع فایل DOS

شماره تابع	عمل	پارامترهای انتقالی	پارامترهای برگشتی

40_{16}	نوشتن روی یک فایل	ثبات **BX** شامل هندل فایل
		ثبات **CX** شامل بایت های مورد نظر برای نوشتن و آدرس بافر منبع در **DS:DX**
			if CF=1 then
			AX error code
			else
			AX bytes written

41_{16}	حذف کردن یک فایل	افست نام فایل در **DS : DX**	**if CF=1 then** **AX error code**



دستورالعملهای IN , OUT

خیلی شبیه دستورالعملهای MOV می باشند. هیچگونه فلگی را تغییر نمی دهند. با این دستورالعملها می توان یک بایت یا یک کلمه را انتقال داد. منبع دستورالعمل out بایستی ثبات AX یا AL باشد. بهمین ترتیب مقصد دستورالعمل in بایستی ثبات AX یا AL باشد.

نکته :

از فرم آدرس حقیقی دستورالعملهای **in , out** زمانی می توان استفاده نمود که آدرس ها از ۰ تا ۲۵۵ باشد.

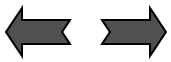
مثال :

دستورالعمل :

```
IN    AX , 07 CH
```

معادل دستورالعملهای زیر می باشد :

```
MOV    DX , 07CH
IN       AX , DX
```



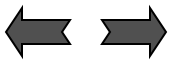

فصل هفدهم
پردازش اسمبلی

فهرست مطالب فصل هفدهم

- اسمبلی دو گذری
- کد های ثبات ها در دستورالعمل های 8088
- کدگذاری آدرس موثر در 8088
- دستوراسمبلر ASSUME
- مقدار دهی ثبات های سگمنت
- دستور اسمبلر TITLE

اسمبلی دو گذری

ماکرو اسمبلی مایکروسافت یک اسمبلی دو گذری است. این به آن معنی است که یک برنامه منبع زبان اسمبلی دو بار بوسیله MASM پویش می شود تا فایل کد هدف آن ایجاد شود. می توان یک اسمبلی را بصورت یک گذری طرح کرد و بعضی اسمبلی ها برنامه منبع را سه بار یا بیشتر پویش می کنند.

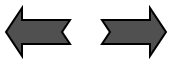


کدهای ثبات ها در دستورالعمل های ۸۰۸۸

کد	ثبات ۱۶ بیتی	ثبات ۸ بیتی	ثبات سگمنت
000	AX	AL	ES
001	CX	CL	CS
010	DX	DL	SS
011	BX	BL	DS
100	SP	AH	
101	BP	CH	
110	SI	DH	
111	DI	BH	

کد گذاری آدرس موثر در ۸۰۸۸

r/m	mod=00	mod=10 یا mod=01
000	[BX+SI]	[مقدار جابجایی + BX+SI]
001	[BX+DI]	[مقدار جابجایی+BX+DI]
010	[BP+SI]	[مقدار جابجایی+BP+SI]
011	[BP+DI]	[مقدار جابجایی+BP+DI]
100	[SI]	[مقدار جابجایی+SI]
101	[DI]	[مقدار جابجایی+DI]
110	(حالت مستقیم)	[مقدار جابجایی+ BP]
111	[BX]	[مقدار جابجایی+ BX]

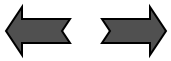


دستور اسمبلر ASSUME

دستور اسمبلر ASSUME معین می نماید که کدام یک از این سگمنت ها سگمنت کد (که شماره آن در CS قرار می گیرد) و کدامیک سگمنت داده (که شماره آن در DS قرار می گیرد) و ..... می باشد. دستور اسمبلر ASSUME دارای فرم زیر می باشد :

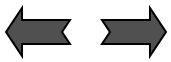
```
ASSUME  Segment_register : segment_name , ……
```

segment_register می تواند هر کدام از ثبات های CS , DS , ES و یا SS بوده و Segment_name بر چسب دستور اسمبلر segment است.



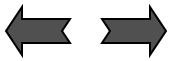
مقدار دهی ثباتهای سگمنت

بطور خاص بایستی گفت که CS بوسیله DOS مقدار دهی می گردد. همچنین DOS ثبات SS را مقدار دهی می نماید. ولی مقدار دهی ثبات های DS , ES توسط برنامه نویس انجام می شود.



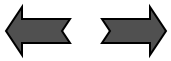
مثال :

```
MOV     AX , data
MOV     DX , AX
```



دستور اسمبلر TITLE

ماکرو اسمبلر مایکرو سافت دارای دو دستور اسمبلی است که می توانند در فایل لیست تیترهایی را قرار دهند. هر فایل منبع می تواند دارای یک دستور اسمبلر TITLE باشد.

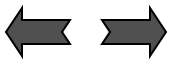


دستور اسمبلر TITLE

این دستور دارای ساختار زیر است:

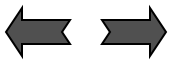
```
TITLE  text
```

که در آن text هر رشته ای از کاراکتر ها تا حد اکثر ۶۰ کاراکتر است. رشته ای که بوسیله این دستور اسمبلر مشخص می شود در دومین خط هر صفحه از فایل لیست اسمبلی نوشته می شود.



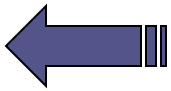
دستور اسمبلر TITLE

یک فایل منبع اسمبلی می تواند دارای تعدادی دستور اسمبلر SUBTTL باشد. ساختار دستور اسمبلر SUBTTL شبیه دستور اسمبلر TITLE است ، متنی که بوسیله آخرین دستور اسمبلر SUBTTL مشخص شده است در سومین خط هر صفحه فایل لیست نوشته می شود.



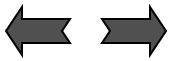

فصل هجدهم
ماکروها و اسمبلی شرطی

فهرست مطالب فصل هجدهم



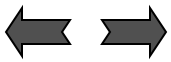
ماکروها

بعضی مواقع لازم است که اشکال نسبتا مختلفی از یک برنامه اسمبلی تولید گردد. ماکرو اسمبلر مایکرو سافت می تواند شرایط گوناگونی را در زمان اسمبلی امتحان کرده و طبق این شرایط، نحوه اسمبل کردن برنامه مورد نظر را انتخاب می کند.



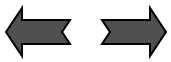
بسط دادن ماکروها

اسمبلر یک ماکرو را به دستورالعملهای تشکیل دهنده ماکروی مزبور بسط داده و سپس این دستورالعملهای جدید را اسمبل می کند. تعریف یک ماکرو شبیه تعریف یک روال در یک زبان سطح بالا می باشد.



بسط دادن ماکروها

خط اول، نام ماکروی مورد نظر و لیست پارامتر ها را ذکر می کند، قسمت اصلی تعریف یک ماکرو متشکل از دستورالعملهائی است که طرز عمل ماکروی مربوط را بر حسب پارامتر های آن بیان می کند. یک ماکرو همچنین مانند یک روال زبانهای سطح بالا فراخوانی می شود



تعریف ماکرو

تعریف یک ماکرو در بین دستورات MACRO , ENDM قرار داده می شوند. شکل تعریف یک ماکرو بصورت زیر می باشد :

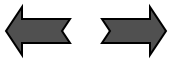
نام MACRO لیست پارامترها

دستورالعملهای زبان اسمبلی

ENDM

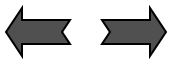
تعریف ماکرو

پارامتر ها در دستور MACRO ، نمادهای معمولی هستند که بوسیله علامت کاما (،) از یکدیگر جدا می شوند. دستورالعملهای اسمبلی در تعریف یک ماکرو می توانند از پارامترهای آن ، ثباتها، عملوندهای بلاواسطه یا نمادهای تعریف شده در بیرون ماکروی مزبور ، استفاده کنند.



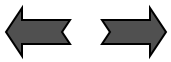
نکته :

تعریف یک ماکرو می تواند در هر جای برنامه اسمبلی ذکر شود بشرط اینکه این تعریف قبل از فراخوانی های آن بیاید. ولی بهتر است تعریف ماکروها در اوائل برنامه اسمبلی ذکر شوند.



ماکروی Pause

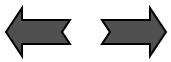
```
Pause   MACRO
; prompt  user  and  wait  for  key  to  be  pressed
        mov  dx,  OFFSET  wait_msg      ; ; “press  any  key ….”
        mov  ah,  09h                   ; ; display  string  function
        int  21h                        ; ; call  DOS
        mov  ah,  08h                   ; ; input  character  function
        int  21h                        ; ; call DOS
        ENDM
```



ماکروی جمع کردن دو عدد صحیح

ماکروی **add2** که مجموع دو پارامتر را پیدا کرده و آنرا در ثبات **AX** قرار می دهد :

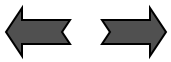
```
Add2        MACRO   nbr1,  nbr2
; ;  put  sum   of  two  word_size  parameters  in  AX

            mov     ax,  nbr1        ; ; first  number
            add     ax,  nbr2        ; ; second  number

            ENDM
```



ماکروی پیدا کردن مینیمم دو مقدار

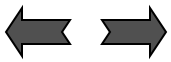
ماکروی min ، مینیمم دو عدد صحیح را پیدا نموده و نتیجه را در AX قرار می دهد.

```
Min2    MACRO  first,  second
        LOCAL   end_if
; ;  put  smaller  of  two  words  in  the  AX register
        mov      ax,  first      ; ;  first   value
        cmp      ax, second      ; ;  first       second  ?
        jle      end_if          ; ;  exit  if  so
        mov      ax,  second     ; ;  otherwise  load  second  value
end_if :
         ENDM
```



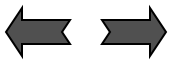
دستور LOCAL در ماکرو ها

دستور LOCAL تنها در تعریف یک ماکرو استفاده می شود و بایستی بلافاصله بعد از دستور MACRO ذکر شود (حتی بین دستورات MACRO , LOCAL نبایستی یک دستورالعمل ملاحظات وجود داشته باشند.) در دستورالعمل LOCAL یک یا چند نماد که با کاراکتر کاما (،) از یکدیگر جدا می شوند، ذکر شده و این نمادها تنها در داخل تعریف ماکروی مربوطه استفاده می شوند.



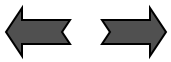
دستور LOCAL در ماکرو ها

هر بار که ماکروی مزبور بسط داده شده و یکی از این نمادها مورد استفاده قرار می گیرد، نماد مربوطه با نمادی که با دو علامت سوال شروع شده و به چهار رقم شانزده شانزدهی ختم می شود ??0000 , ??0001 و غیره) ، جایگزین می گردد. در هر فراخوانی یک ماکرو ، یک نماد ثابت ??dddd جایگزین یک نماد محلی ثابت می گردد.



اسمبلی شرطی

بعضی مواقع برنامه نویس می خواهد که اشکال نسبتا متفاوتی از یک برنامه یا یک روال را تولید نماید. این می تواند در صورتی که برنامه نویس بخواهد در سطح زبان ماشین عملیات ورودی یا خروجی را انجام دهد که تنها آدرس درگاههای مورد استفاده در ماشینهای مختلف تغییر کند، اتفاق بیافتد(یا آدرس درگاههای آدرس های مختلف متصل به یک ماشین تغییر کند.)

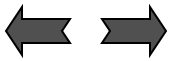


اسمبلی شرطی

مورد دیگر زمانی است که بسط یک ماکرو بر حسب تعداد و نوع آرگومانها تغییر پیـدا مـی کنـد. در ماکرو اسمبلر مایکرو سافت می توان کد منبعی نوشت بطوری که تحت این شرایط یا شـرایط دیگـر ، باشکال مختلف اسمبل شوند.

ماکروی add_all با استفاده از اسمبلی شرطی

```
add_all     MACRO  nbr1, nbr2, nbr3, nbr4, nbr5
; ; add up to 5 word_size integers, putting sum in AX
             mov    ax, nbr1    ; first operand

              IFNB   <nbr2>
             Add     ax, nbr2   ; second operand
             ENDIF

             IFNB   <nbr3>
             Add     ax, nbr3   ; third operand
             ENDIF

             IFNB    <nbr4>
             Add     ax, nbr4   ; fourth operand
             ENDIF

             IFNB     <nbr5>
             Add      ax, nbr5  ; fifth operand
             ENDIF
            ENDM
```



فراخوانی ماکروی add_all

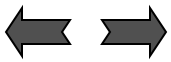
```
Mov  ax , bx              ; first  operand
Add  ax , cx               ; Second  operand
Add  ax , dx               ; third operand
Add  ax , number             ; fourth operand
Add  ax , 1               ; fifth operand
```

فراخوانی

```
Add_all  bx , cx , 45
```

بسط آن بصورت زیر خواهد بود:

```
Mov  ax , bx               ; first  operand
Add  ax , cx               ; Second  operand
Add  ax , 45               ; third operand
```



بلوکهای اسمبلی شرطی

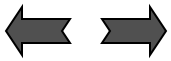
بطور کلی ، بلوکهای اسمبلی شرطی بصورت زیر می باشند:

```
IF . . .  [ operands ]
Statements
ELSE
Statements
ENDIF
```

نکته :

نوع عملوند ها با انواع دستورات IF تغییر کرده و با تمام انواع آنها بکار نمی روند. دستور ELSE و دستورالعملهای بعد از آن اختیاری می باشد.

دستورالعملهای بعد از IF یا ELSE می توانند شامل دستور IF دیگر باشند، به این معنی که بلوک های اسمبلی شرطی می توانند تو در تو باشند.

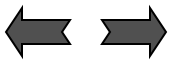


نکته :

با استفاده از دستور EXITM می توان نوشتن و فهمیدن تعریف ماکروها را آسانتر ساخت. زمانی که اسمبلر فراخوانی یک ماکروئی را پردازش کرده و دستور EXITM را پیدا می کند، بلافاصله به عمل بسط ماکرو خاتمه داده و از تمام دستورالعملهائی که در تعریف ماکرو بعد از دستور EXITM قرار دارند صرف نظر می کند.

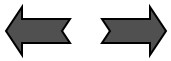
تعریف ماکروی بازگشتی

یادآوری می کنیم که تعریف یک ماکرو می تواند شامل فراخوانی ماکرو باشد. در حقیقت، تعریف یک ماکرو می تواند شامل فراخوانی خودش باشد. باین معنی که تعریف یک ماکرو می تواند بازگشتی باشد.



مثال :

```
Add_all    MACRO    nbr1,  nbr2,  nbr3,  nbr4,  nbr5
 ; ;  add  up  to 5 word_size  integers,  putting  sum  in  AX
            IFB    <nbr1>
            Mov    ax, 0          ; ;  initialize  sum
            ELSE
            Add_all    nbr2, nbr3, nbr4, nbr5   ; ;  add  remaining  arguments
            Add    ax, nbr1      ; ;  add  first  argument
            ENDIF
          ENDM
```




فصل نوزدهم
مثال های نمونه

فهرست مطالب فصل نوزدهم

مشخص کردن فلگ ها

بعد از جمع کردن دو مقدار 5439 ,456A در مبنای ۱۶ مقادیر فلگ ها عبارتند از :

Pf = 1
Az=1
Zf=0
Sf=1
Of =1
CF=0

زیرا

0100 0101 0110 1010 +

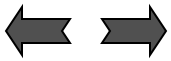
0101 0100 0011 1001

1001 1001 1010 0011

محاسبات روی مقادیر double word

قطعه برنامه زیر مقدار عبارت x+y +30-z را که در آن x,y,z از نوع double word بوده محاسبه نموده نتیجه را در w قرار دهد.

```
MOV     AX , X
MOV     AX , X+2
ADD     AX , Y
ADC     DX , Y+2
ADD     AX , 30
ADC     DX ,0
SUB     AX , Z
SBB     DX , Z+2
MOV     W , AX
MOV     W+2 , DX
```



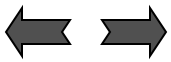
ضرب دو مقدار با علامت

```
MOV    AL , 0B4H
MOV    BL , 11H
IMUL   BL
BL    00010001
AL    10110100
```

چون از IMUL استفاده شده و مقدار BL برابر است با 17 و مقدار AL برابر است با -76

17 *-76 = -1292

محتوی AX برابر است با -1292



ضرب دو مقدار بدون علامت

```
MOV     AL , 0B4H
MOV     BL , 11H
MUL     BL
BL    0001001
AL   10110100
MUL   BL
BL    00010001
AL    10110100
```

چون از MUL استفاده گردید مقدار BL برابر است با 17 و مقدار AL برابر است با 180

17 * 180 = 3060

محتوی AX برابر با 3060

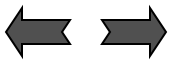
دستور العمل CBW

این دستور العمل محتوی AL را که از نوع بایت می باشد به WORD تبدیل نموده و مقدار بدست آمده را در AX قرار می دهد . از این دستور العمل می توان برای تبدیل مقداری از نوع بایت به WORD استفاده نمود.

مثال :

برای ضرب محتوی AL در محتوی BX بایستی بصورت زیر عمل استفاده نماییم .

```
CBW
IMUL    BX
```



دستور العمل CWD

از این دستور العمل برای تبدیل یک مقدار از نوع **WORD** به **double word** استفاده می گردد.
برای اینکار مقدار را بایستی در رجیستر **ax** قرار داد.آنگاه نتیجه تبدبل در ریجیسترهای **DX:AX** قرار می گیرد.

مثال، برای تقسیم محتوی **AX** بر **BX** بایسی بصورت زیر عمل نمود.

```
CWD
IDIV  BX
```

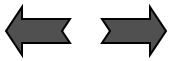
تغییر بیت های یک ثبات

با استفاده از دستورالعمل **AND** , **OR** , **XOR** می توان مقادیر بیت های یک ثبات را تغییر داد.

برای یک کردن بیت ها **OR**

برای مکمل کردن بیت ها **XOR**

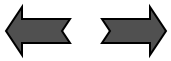
برای صفر کردن بیت ها **AND**



ماکرو WAIT

از ماکروی زیر جهت ایجاد تاخیر می توان استفاده نمود.

```
WAIT MACRO COUNT
   LOCAL NEXT
   PUSH CX
   MOV CX , COUNT
   NEXT : LOOP NEXT
   POP CX
ENDM
```



مرتب سازی حبابی

برنامه زیر N مقدار از نوع WORD را گرفته به روش حبابی به صورت صعودی مرتب می نماید.

```
MOV   CX,N
DEC   CX
LOOP1 :  MOV  DI,CX
         MOV  BX,0
LOOP2 :  MOV  AX,A[BX]
         CMP  AX,A[BX+2]
        JGE  CONTINUE
       XCHG  AX , A[BX+2]
        MOV   A[BX] , AX

CONTINUE :
                  ADD  BX,2
                  LOOP LOOP2
                  MOV CX , DI
                  LOOP LOOP1
```