فصل ۳– اصول و قواعد برنامه نویسی در اسمبلی:

الف) قالب دستورات در اسمبلی طبق الگوی زیر می باشد:

Label : opcode opreand1 , opreand2 , ... ; Comment **توضیحات; ..... , عملوند۲ , عملوند۱ کد اجرا : برچسب**

به عنوان مثال :

Main: mov AH,0 ; move zero to AH

نکته ۱ : برچسب ها و توضیحات اختیاری هستند.

نکته ۲: کد اجرا معمولا ۳ یا ۴ حرفی است. مانند **MOV, ADD, SUB** و ...

نکته ۳: عملوندها می توانند آدرس حافظه , نام ثبات و یا عدد ثابت باشد. بعضی از دستورات اصلا عملوندی ندارند.(مثل دستور **Nop**)

ب) روش نامگذاری آدرس اطلاعات و برچسب دستور:

Label : opcode opreand , address memory , . . . ; Comment **توضیحات; ..... , آدرس حافظه , عملوند کد اجرا : برچسب**

به عنوان مثال :

Mov R0,[1500]

یعنی محتوای خانه حافظه که آدرس آن ۱۵۰۰ است را در ثبات **R0** قرار بده

البته میتوانیم به جای آدرس عددی حافظه از یک نام برچسب سمبولیک مثلا **sum** استفاده کنیم که قبلا این اسم را به آن آدرس اختصاص داده ایم.

Mov R0, sum

ج) انواع روشهای آدرس دهی اطلاعات یا عملوندها در دستورات اسمبلی:

۱) روش آدرس دهی ضمنی(implied addressing)

در این روش به آدرس خاصی اشاره نمی شود و خود دستور تعیین کننده آدرس است.

مثال: دستور CLc یا دستور STC که در هر دو دستور بیت پرچم CF مقدار دهی می شود.(به ترتیب آن بیت را صفر و یک میکنند)

۲) روش آدرس دهی بلافاصله یا بلادرنگ(immediate addressing)

در این روش اطلاعات یا عملوند در خود دستور قرار دارند و نیازی نیست که از حافظه یا ثبات آورده شود.

مثال: ADD R1,4000

یعنی عدد ثابت ۴۰۰۰ را با ثبات R1 جمع کن و نتیجه را در ثبات R1 قرار بده.

۳) روش آدرس دهی ثبات ها (register addressing)

در این روش نام ثبات ها در دستورات قرار دارند.

مثال: ADD R1,R2

در این روش چون ثباتها در داخل پروسسور قراردارند پس سرعت اجرای دستورات بالاست.

۴) روش آدرس دهی مستقیم(direct addressing)

در این روش آدرس اطلاعات حافظه در دستور وجود دارد.

مثال : ADD [8000],R1

یعنی محنوای ثبات R1 را با محتوای خانه حافظه به آدرس ۸۰۰۰ جمع کن و نتیجه را در خانه ۸۰۰۰ حافظه قراربده.(وقت زیادی برای ارتباط پروسسور و حافظه تلف می شود)

۵) روش آدرس دهی غیر مستقیم (Indirect addressing)

در این روش ثبات اشاره گر در دستور قرار دارد که محتوای آن ثبات , آدرس خود اطلاعات در حافظه می باشد.

مثال : ADD R0,(R1)

یعنی محتوای خانه حافظه ای که آدرس آن در ثبات R1 قرار دارد را با ثبات R0 جمع کن و نتیجه را در R0 قرار بده.

از محاسن این روش این است که میتوان در طول برنامه محتوای ثبات اشاره گر را عوض کرد و بالطبع به نقاط مختلفی از حافظه دستیابی داشت.

۶) روش آدرس دهی با ثبات پایه و نسبی (relative and base register Addressing)

در روش آدرس دهی با ثبات از دو قسمت تشکیل شده است: یکی آدرس پایه است که آدرس شروع یک ناحیه از حافظه می باشد که در یک ثبات قرار می گیرد و قسمت دوم آدرس جابجایی d یا آفست است که که در دستور نوشته می شود.

در روش آدرس دهی نسبی , اشاره گر دستور IP به عنوان ثبات پایه در نظر گرفته می شود و آدرس جابجایی d در دستور نوشته می شود به مقدار اشاره گر IP اضافه می شود و مقدار جدید آدرس شروع دستور خواهد بود.

کاربرد آن در دستور پرش JUMP یا انشعاب Branch می باشد.

مثال: JUMP 1003 یعنی از آدرس فعلی به آدرس ۱۰۰۳ پرش کن

۷)روش آدرس دهی ایندکس یا شاخص(Index addressing)

در این روش بیشتر برای دستیابی مستقیم به اجزاء یک جدول یا یک ماتریس بکار می رود.

مثال : ADD R2,20(R1)

یعنی محتوای آدرس خانه حافظه ای که آدرس آن ۲۰ واحد بیشتر از آدرس R1 است را با محتوای ثبات R2 جمع کن و در R2 قرار بده.

۸) روش آدرس دهی افزایش و کاهش خودکار

در این روش آدرس داخل ثبات اشاره گر یک واحد اضافه و یا یک واحد کم میشود.

مثال: Mov R1,(R2)+

یعنی محتوای حافظه به آدرس R2 را به ثبات R1 منتقل کن و سپس ثبات R2 را یک واحد اضافه کن تا برای دستور بعدی آماده باشد.

****

نحوه ترجه برنامه اسمبلی توسط مترجم اسمبلر:

در مرور اول اسمبلر تمام برنامه اسمبلی را می خواند , سپس تعداد بایت های هر دستور را مشخص کرده و یک جدول سمبول ها از برچسب دستورات و آدرس آن ها و نام آن ها درست میکند. در مرور دوم برنامه اسمبلی توسط مترجم اسمبلر با استفاده از جدول درست شده ترجمه برنامه را کامل می کند و کدهای باینری برنامه را به زبان ماشین تبدیل می نماید.

یکی دیگر از وظایف مهم اسمبلر , بررسی اشتباهات احتمالی در برنامه اسمبلی است. مثل بررسی کد اجرایی غیر معتبر و یا تشخیص آدرس نامعتبر.

طریقه اجرای برنامه به زبان اسمبلی:

ابتدا باید دستورات را در یک محیط ویرایشگر مثل Notepad تایپ کرد و تحت یک نام و با پسوند ASM ذخیره کرد.

سپس باید برنامه را توسط مترجم اسمبلر به زبان ماشین ترجمه کرد که نام این برنامه اسمبلر MASM (Macro assembler) یا TASM (Turbo assembler)است و

با اجرای این مترجم ها و معرفی اسم برنامه ذخیره شده , عملیات ترجمه آغاز می شود و در نهایت برنامه ترجمه شده با یک اسم دلخواه اما با پسوند OBJ ذخیره می گردد.

در نهایت توسط یک برنامه پیوند دهنده یا linker و تحت نظارت سیستم عامل فایل اجرایی برنامه با پسوند EXE تولید می شود.

فصل ۴ – ساختار اصولی و ساده شده برنامه نویسی اسمبلی

معمولاً هر برنامه زبان اسمبلی از تعدادی دستورالعمل ساخته شده است که بیانگر عملیاتی است که بایستی انجام شود . مجموعه دستور
العملهای یک میکروپروسسور لیست تمام فرمانها یی است که CPU می تواند تشخیص دهد و اجرا کند.

شبه دستورات : اسمبلر دارای فرمانهایی است که کاربر را در کنترل ترجمه و تهیه لیست های برنامه یاری می کند . این فرمان ها به شبه
دستورات معروف هستند که کد زبان ماشین تولید نمی کنند.

۱- **شبه دستور PAGE** :جهت تعیین مشخصات صفحه

PAGE [length **] , [** width **]**

Length : بیانگر تعداد خطوط در صفحه است

Width : بیانگر تعداد حروف در هر خط است.

مثال : PAGE 40,80

در این مثال زمانی که length به عدد ۴۰ رسید ، کنترل لیست به بالای صفحه بعد منتقل و یکی به شماره صفحه اضافه میشود . اگر جلوی PAGE عددی ننویسیم لیست اسمبل شده بطور خودکار پس از مواجه شدن با PAGE به صفحه جدید منتقل می شود.

۲- **شبه دستورTITLE**: جهت قرار دادن یک نام برای برنامه استفاده می شود (. حداکثر ٦١ حرف)

TITLE Text [COMMENT]

مثال: TITLE 'ZEINAB.asm'

۳- **شبه دستور END** :**نقطه انتهای یک برنامه یا یک سگمنت را برای اسمبلر مشخص می کند.**

END [label]

مثال:

```
Start:  .....
        .....
        .....
        END Start
```

٤- **شبه دستور تعریف قطعه یا سگمنت (SEGMENT و ENDS)**

هر برنامه در اسمبلی از یک یا چند بخش یا سگمنت تشکیل شده است. مثل سگمنت پشته , سگمنت داده , سگمنت کد.

لذا الگوی نوشتن سگمنتها به شکل زیر است:

NAME SEGMENT [Align][combine][CLASS]

دستورات یا داده ها

NAME ENDS

با این شبه دستور یک سگمنت را تعریف می کنیم و ابتدا و انتهای آن را مشخص می سازیم.(حداکثر ظرفیت هر سگمنت ۶۴ کیلوبایت است)

مثال :

```
DATASG    SEGMENT ;
DATA1     DB  100;
DATA2     DB  120;
DATA3     DB  ? ;
DATASG    ENDS
```

در این مثال یک سگمنت داده با نام DATASG معرفی شده و دارای ۳ داده به نامهای DATA1,DATA2,DATA3 می باشد.

نوشتن نام قبل از شبه دستورات SEGMENT و ENDS اختیاری است.

پارامتر ALIGN: به معنی همترازی می باشد که اختیاری بوده و جهت انطباق شروع یک سگمنت با آدرس حافظه بکار می رود.

پارامتر COMBINE: به معنی ترکیب بوده و چگونگی ترکیب سگمنت را با سگمنتهای همنام در سایر برنامه ها مشخص میکند.

این پارامتر شامل دو حالت PUBLICوSTACK می باشد. حالت اول دو سگمنت را به هم پیوند زده و در موقع پیوند در حافظه تشکیل یک سگمنت بزرگتری را می دهند. حالت دوم یعنی STACK فقط در سگمنتهای پشته بکار می رود و نوشتن آن در آنجا اجباری است و در موقع پیوند بین پشته برنامه و پشته سیستم عامل ترکیب میشوند.

پارامتر CLASS: کلاسها برای ترکیب سگمنت های از یک نوع بکار می رود . به عنوان مثال سگمنت داده از یک برنامه با سگمنت داده از برنامه دیگر و یا سگمنت کد از یک برنامه با سگمنت کد از برنامه دیگری قرار است با هم ترکیب شوند. برای این منظور از عبارتهای 'CODE' و'DATA' و'STACK' در داخل علائم تک کوتیشن استفاده میکنیم که به ترتیب بیانگر سگمنت کد , داده و پشته می باشند.

بنابراین شبه دستور SEGMENT برای هر سه حالت فوق به شکل زیر نوشته می شود.

تعریف سگمنت پشته

```
نام سگمنت    SEGMENT    'STACK'

نام سگمنت    ENDS
```

تعریف سگمنت داده

```
نام سگمنت    SEGMENT    'DATA'

نام سگمنت    ENDS
```

تعریف سگمنت کد

```
نام سگمنت    SEGMENT    'CODE'

نام سگمنت    ENDS
```

۵- **شبه دستورات تعریف روال شامل PROC , ENDP , FAR , NEAR**

در سگمنت کد که حاوی دستورات برنامه است حداقل یک یا چند روال یا پروسیجر دارد که با شبه دستور **PROC** تعریف می شود و پایان آن با شبه دستور **ENDP** بیان می گردد. در ضمن جلوی دستور **PROC** یکی از صفتهای **FAR** و یا **NEAR** میتواند نوشته شود(اگر صفتی بیان نشود پیش فرض **NEAR** در نظر گرفته می شود.)

NEAR به معنی داخلی بودن روال و فقط داخل همین سگمنت قابل فراخوانی است.

FAR به معنی خارجی بودن یا دور بودن و می تواند از سگمنتهای دیگری هم قابل فراخوانی شود.

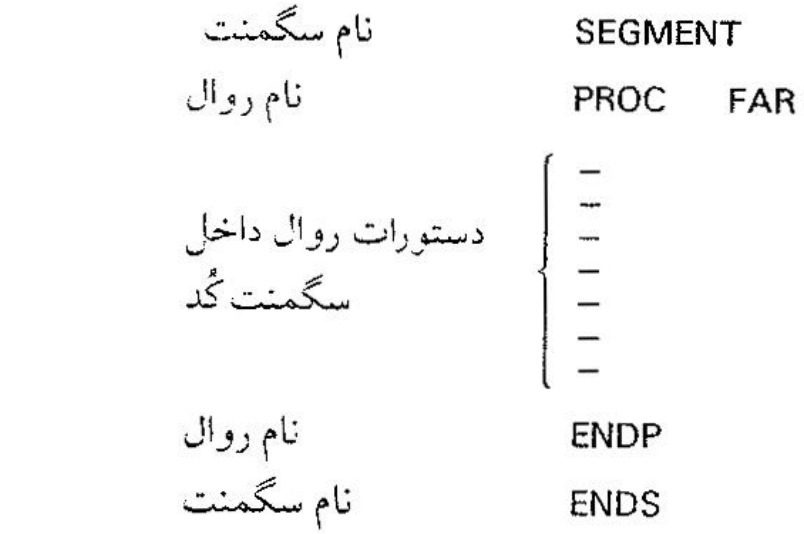
الگو:

```
نام روال    PROC [صفتها]

دستورات برنامه

نام روال    ENDP
```

مثال :




```
CODESG   SEGMENT 'CODE'
MAIN     PROC FAR
         ---
         ---
MAIN     ENDP
CODESG   ENDS
         END  MAIN
```

```
        PAGE 110,100
TITLE   'INTE-DIS.ASM'  procedures & interrupts
;-----------------------------------------------
;               Defining Segment of Program
;               ---------------------------
CODESG  SEGMENT 'CODE'
        ASSUME SS:CODESG,DS:CODESG,CS:CODESG
        ORG 100H
START:  JMP MAIN          ;1-Jump to instructions
;-----------------------------------------------
CHAR    DB 00
;-----------------------------------------------
;               Main procedure
;               --------------
MAIN    PROC NEAR
        CALL CLRMON       ;2-Clear monitor
        CALL SETCUR       ;3-Set cursor
        CALL CHARA        ;4-Display characters
        MOV AX,4C00H      ;5-End of
        INT 21H           ;6-processing
MAIN    ENDP              ;7-End procedure
;               Clear monitor
;               -------------
CLRMON  PROC NEAR         ;8-Begin of procedure
        MOV AX,0700H      ;9-Scroll down completely
        MOV BH,07H        ;10-White on black
        MOV CX,0000       ;11-Upper left location
        MOV DX,184FH      ;12-Lower right location
        INT 10H           ;13-Call BIOS
        RET               ;14-Return to caller
CLRMON  ENDP              ;15-End of procedure
;               Set cursor at 10,00
;               -------------------
SETCUR  PROC NEAR         ;16-Begin of procedure
        MOV AH,02H        ;17-Request set cursor
        MOV DH,10         ;18-Row 10
        MOV DL,00         ;19-Column 00
        INT 10H           ;20-Call BIOS
        RET               ;21-Return to caller
SETCUR  ENDP              ;22-End of procedure
;               Display characters
;               ------------------
CHARA   PROC NEAR         ;23-Begin of procedure
        MOV CX,256        ;24-  256 iterations
        MOV AL,CHAR       ;25-  00 for ASCII
BACK1:  MOV AH,0EH          ;26-Display ASCII
        INT 10H             ;27-characters
        INC AL              ;28-Next character
        LOOP BACK1          ;29-Loop nonzero
        RET                 ;30-Return to caller
CHARA   ENDP                ;31-End of procedure
CODESG  ENDS                ;32-End of segment
        END START           ;33-End of program
```

شکل (۶-۱) نمونه‌ای از برنامه اسمبلی با توضیحات هر قسمت

نکته۱:

برای نوشتن توضیحات در برنامه از علامت سمیکالن (;) در ابتدای هر سطر استفاده می شود و کشیدن خط هم برای جداکردن قسمتهای برنامه از یکدیگر است.

نکته ۲:

در مثال فوق از دستورات CALL وRET استفاده شده که وظیفه آنها فراخوانی و خروج از یک زیر برنامه است.

نکته ۳ :

دودستور زیر باعث می شود کنترل از برنامه به سیستم عامل بر می گردد و کامپیوتر متوقف می شود:

```
MOV   AX,4C00H
INT   21
```

٦- شبه دستور ASSUME

این شبه دستور ارتباط بین نام هر سگمنت و ثباتهای آن سگمنت را برقرار می کند.

ثبات SS : جهت معرفی آدرس شروع سگمنت پشته

ثبات DS : جهت معرفی آدرس شروع سگمنت داده

ثبات CS : جهت معرفی آدرس شروع سگمنت کد

الگوی این دستور:

```
ASSUME    SS:نام سگمنت پشته , DS:نام سگمنت داده , CS : نام سگمنت کد
```

مثال :

```
ASSUME    SS:STACKSG , DS: DATASG , CS: CODESG
```

در انتهای این بحث قالب کلی برنامه نویسی به زبان اسمبلی را به شرح زیر بیان میکنیم:

```
Stksg    segment    'Stack'
.
.
Stksg    Ends

Datasg   segment    'data'
.
.
Datasg       Ends

Codsg    segment    'code'
Proc_name    proc    far or near
.
.
Proc_name      Endp
Codsg    Ends
.
{تعریف سایر زیر برنامه ها که اختیاری است}
.
End      Proc_name
```

ابتدا تعریف سگمنت پشته

بعد تعریف سگمنت داده

بعد تعریف سگمنت کد

پروسیجر اصلی اینجا تعریف می شود

پایان کل برنامه

مثال کتاب : نمونه ای از برنامه اسمبلی برای جمع دو مقدار

```
        PAGE 110,100
TITLE   'ADDNUMBE.ASM'  a sample of EXE program
;-----------------------------------------
;
;               1- Define stack segment
;               -----------------------
STACKSG SEGMENT STACK 'STACK'
        DW 32H DUP(0)    ; Define 32H word stack
STACKSG ENDS             ; End of segment
;-----------------------------------------
;               2- Define data segment
;               ----------------------
DATASG  SEGMENT 'DATA'
DATA1   DB 100           ; Define data
DATA2   DB 120
DATA3   DB  ?
DATASG  ENDS             ; End of segment
;-----------------------------------------
;                3- Define code segment
;                ----------------------
;
CODESG  SEGMENT 'CODE'
        ASSUME SS:STACKSG,DS:DATASG,CS:CODESG
MAIN    PROC FAR
        MOV AX,DATASG    ; 1-Initialize DS
        MOV DS,AX        ; 2-register
;
        MOV AL,DATA1     ; 3-Move 100 to AL
        ADD AL,DATA2     ; 4-Add 120 to AL
        MOV DATA3,AL     ; 5-Store sum in DATA3
;
;       Come back to operating system
;       ---------------------------------
        MOV AX,4C00H     ; 6-End of
        INT 21H          ; 7-processing
MAIN    ENDP             ; End of procedure
CODESG  ENDS             ; End of segment
        END MAIN         ; End of program
```

شکل (۶-۳) نمونه‌ای از برنامه اسمبلی برای جمع دو مقدار