أ/ عبد القادر البعداني abdalkader_2015@hotmail.com

محاضرات بلغة التجميع (مدعومة بكثير من البرامج)

اليمن – اب

جامعة الجزيرة

للعام الجامعي ٢٠١٠-٢٠١١م

مدخل إلي لغة التجميع

تعليمات لغة التجميع :-

يتم تحويل برنامج لغة التجميع للغة الآلة بواسطة برنامج يسمى Assembler وبالتالي يجب كتابة التعليمات بصوره محدده حتى يتعرف عليها الـ Assembler، وفى هذا الجزء سنتناول الشكل العام للأوامر المستخدمة.

يتكون البرنامج من مجموعه من التعليمات أو الأوامر بحيث يحتوى كل سطر على أمر واحد فقط كما أن هنالك نوعين من التعليمات.

الأوامر أو التعليمات Instructions والتي يقوم الـ Assembler بتحويلها إلي لغة الآلة والإيعازات Assembler-Directives وهى إيعازات للـ Assembler للقيام ببعض العمليات المحددة مثل تخصيص جزء من الذاكرة لمتغير محدد وتوليد برنامج فرعى.

كل الأوامر في لغة التجميع تأخذ الصورة

```
OPERATION        OPERAND(S)        COMMENT
```

- **يتم الفصل بين الحقول بواسطة مفتاح الـ TAB أو المسطرة(SPACE) أي يكون هناك فراغ واحد على الأقل بين كل حقل والحقل التالي.**
- **الحقل Operation يحتوى على الأمر المطلوب تنفيذه.**
- **الحقل Operation(s) يحتوى على المعامل أو المعاملات المطلوب تنفيذها بواسطة الأمر المحدد ويعتمد على نوع الأمر. (لاحظ أن هناك بعض الأوامر لا تتطلب وجود هذا الحقل).**
- **حقل الملحوظات الـ Comments يستخدم عادة للتعليق على الأمر الحالي وهو يستخدم لتوثيق البرنامج.**

كمثال للتعليمات

```
MOV CX , 5 ; initialize counter
```

الأمر المستخدم MOV والمعاملات هي CX والرقم ٥ ومعنى ذلك هو وضع الرقم ٥ في المسجل CX وحقل الملاحظات يوضح أن ٥ هي القيمة الابتدائية للعداد.

ومثال للإيعازات:

```
Main Proc
```

وهذا الإيعاز يقوم بتعريف برنامج فرعي (إجراء) باسم Main.

١ – الأمر MOV

يستخدم الأمر MOV في نقل البيانات من مكان لآخر وهذه الأماكن هي المسجلات العامة أو المسجلات الخاصة أو المتغيرات في الذاكرة أو حتى في نقل (وضع) قيمة ثابتة في مكان محدد من الذاكرة أو علي مسجل. والصورة العامة للأمر هي

```
MOV  Destination , Source
```

حيث يتم نقل محتويات المصدر Source إلي المستودع Destination ولا تتأثر قيمة المصدر بعد تنفيذ الأمر مثلاً

```
MOV AX , Word1
```

حيث يتم نسخ محتويات (قيمة) المتغير Word1 إلي المسجل AX. وبالطبع يتم فقد القيمة الأولية للمسجل AX بعد تنفيذ الأمر. كذلك الأمر

```
MOV AL, 'A'
```

يقوم بوضع الرقم 041h (وهو الرقم المناظر للحرف A في جدول الـ ASCII) في المسجل AL.

الجدول التالي يوضح قيود استدام الأمر MOV

	المستودع			
المصدر	مسجل عام	مسجل مقطع	متغير (موقع في الذاكرة)	ثابت
مسجل عام	مسموح	مسموح	مسموح	غير مسموح
مسجل مقطع	مسموح	غير مسموح	مسموح	غير مسموح
متغير (موقع في الذاكرة)	مسموح	مسموح	غير مسموح	غير مسموح
ثابت	مسموح	غير مسموح	مسموح	غير مسموح

الشكل العام للبرنامج:-

```
.MODEL SMALL
.STACK 100H
.DATA
هنا يكون تعريف المتغيرات والثوابت ;
.CODE
MAIN PROC
التعليمات والأوامر داخل الإجراء ;
MAIN ENDP
بقية الإجراءات تكتب هنا;
END MAIN
```

آخر سطر في البرنامج يحوى كلمة نهاية البرنامج END متبوعة باسم الإجراء الرئيسي في البرنامج

شرح الشكل العام :

ويتم كتابة هذا السطر قبل تعريف أي نقطة ويوجد لدينا اكثر من نموذج للذاكرة سوف يتم توضيحها في الجدول التالي ولكن عموماً إذا لم يكن حجم البيانات كبيراً يتم غالباً استخدام النموذج SMALL وهذا هو الحال في اغلب البرامج التي سنتطرق لها. ويتم كتابة السطر على الصورة التالية:. MODEL SMALL

الجدول التالي يوضح أسماء موديلات الذاكرة المختلفة وتوضيح خصائص كل منها

الموديل MODEL	الوصف
SMALL	الكود في مقطع واحد والبيانات في مقطع واحد
MEDIUM	الكود في أكثر من مقطع والبيانات في مقطع واحد
COMPACT	الكود في مقطع واحد والبيانات في أكثر من مقطع
LARGE	الكود في أكثر من مقطع والبيانات في أكثر من مقطع ولكن غير مسموح بتعريف مصفوف اكبر من 64k BYTE
HUGE	الكود في أكثر من مقطع والبيانات في أكثر من مقطع ولكن يمكن أن يكون هناك مصفوف بطول اكبر من 64k BYTE

مقطع المكدس Stack Segment:

الغرض من مقطع المكدس هو حجز جزء من الذاكرة ليتم استخدامه في عملية تكديس البيانات أثناء تنفيذ البرنامج. ويجب أن يكون هذا الحجم كافي لتخزين كل المكدس في أقصي حالاته (لتخزين كل القيم المطلوب تكديسها أثناء عمل البرنامج).

ويتم تعريف مقطع المكدس باستخدام التعريف:.Stack Size

حيث size يمثل عدداً اختيارياً هو حجم المكدس بالوحدات bytes. والمثال التالي يقوم بتعريف المكدس بحجم 100h

```
.Stack 100h
```

إذا لم يتم تعريف الحجم يتم افتراض الحجم 1KB بواسطة الـ Assembler.

مقطـــع البرنامج Code Segment:

يحتوى هذا المقطع على الأوامر والتعليمات المستخدمة داخل البرنامج

حيث Name هو اسم الإجراء، أما Proc و Endp فهما إيعازات Pseudo_Ops

تعليمات الإدخال والإخراج INPUT &OUTPUT INSTRUCTIONS

يتعامل المعالج الدقيق مع الأجهزة الخارجية باستخدام موانئ الإدخال والإخراج وذلك باستخدام الأوامر IN للقراءة وفى ميناء إدخال والأوامر OUT للكتابة في ميناء إخراج.

يوجد نوعان في روتينات الخدمة المستخدمة في التعامل مع الموانئ يسمى الأول BIOS (BASIC INPUT /OUTPUT SYSTEM) والثاني باستخدام الـ DOS. روتينات الـ BIOS يتم تخزينها في ذاكرة القراءة فقط (الـ ROM) ويتعامل مباشرة مع موانئ الإدخال والإخراج بينما خدمات الـ DOS تقوم بتنفيذ عمليات أكثر تعقيداً مثلاً طباعة سلسلة حروف وهي تقوم عادة باستخدام الـ BIOS في تنفيذ عمليات إدخال/إخراج مباشرة.

يتم نداء الـ BIOS أو الـ DOS لتنفيذ عملية محددة باستخدام نداء مقاطعة INT (INTERRUPT) والنداء على هذه الصورة

```
INT INTERRUPT_NUMBER
```

توضيح:

المقاطعة هي عبارة عن إجراء خاص أو روتين خاص مبرمج سلفا ومخزون في مكان معروف من الذاكرة ويستدعى لغرض إنجاز مهمة معينة .

- **وتستخدم لبرمجة عمليات الادخال والاخراج المختلفة إي نقل البيانات من وحدات الادخال و الاخراج الي و حدة المعالجة المركزية و بالعكس**

<u>نداء المقاطع رقم 21H (INT 21H)</u>

- **يتم استخدام هذا النداء لتنفيذ مجموعة كبيرة من الخدمات التي يقدمها نظام التشغيل DOS حيث يتم وضع رقم الخدمة المطلوبة في المسجل AH وقد يتطلب الأمر وضع بعض القيم في مسجلات أخرى وذلك حسب نوع الخدمة المطلوبة وبعد ذلك يتم نداء طلب المقاطعة 21H. وقد يتطلب الأمر استقبال قيم محددة في نداء المقاطعة حيث يتم وضعها في المسجلات. يتم وضع الخدمات المختلفة في جدول كبير يوضح وظيفة كل خدمة والمدخلات إليها والمخرجات منها.**

الجدول التالي يوضح ثلاثة فقط من الخدمات التي يخدمها النظام

رقم الخدمة	الوصف (الروتين)
1	قراءة قيمة واحدة من لوحة المفاتيح
2	كتابة حرف واحد في الشاشة
9	كتابة مجموعة من الحروف في الشاشة

في الجزء التالي ستناول بعض هذه الخدمات

الخدمة رقم 1: قراءة حرف من لوحة المفاتيح

المدخلات: وضع الرقم ١ في المسجلAH

المخرجات: المسجل AL يحتوي علي كود ال ASCII للحرف الذي تم الضغط عليه في لوحة المفاتيح أو 0 في حالة الضغط على مفتاح غير حرفي NON CHARACHTER KEY

(مثلا المفاتيح F1-F10).

لتنفيذ هذه الخدمة تتم كتابة الآتي:-

```
MOV AH, 01
INT 21H
```

تقوم هذه الخدمة بانتظار المستخدم إلى حين الضغط على لوحة المفاتيح. عند الضغط على أي مفتاح يتم الحصول على كود الـ ASCII للمفتاح من المسجل AL كما يتم عرض الحرف الذي تم الضغط عليه في لوحة المفاتيح علي الشاشة. ولا تقوم هذه الخدمة بإرسال رسالة إلي المستخدم فهي فقط تنتظر حتى يتم الضغط على مفتاح. إذا تم ضغط بعض المفاتيح الخاصة مثل F1-F10 فسوف يحتوي المسجل AL علي القيمة صفر. التعليمات التي تلي INT 21h تستطيع فحص المسجل AL و تتخذ الفعل المناسب.

2- الخدمة رقم 2: عرض حرف على الشاشة أو تنفيذ وظيفة تحكم.

المدخلات : وضع الرقم 02 في المسجل AH.

وضع شفرة الـ ASCII كود للحرف المطلوب عرضه في المسجل DL.

المخرجات : الكود الـ ASCII للحرف الذي تم عرضه يتم وضعه في المسجل AL.

مثال: الأوامر التالية تعرض علامة استفهام علي الشاشة

```
MOV  AH , 02H
MOV  DL , '?'
INT   21H
```

بعد طباعة الحرف على الشاشة يتحرك المؤشر إلي الموضع التالي (إذا كان الوضع الحالي هو نهاية السطر يتحرك المؤشر إلي بداية السطر الجديد).

يتم استخدام هذه الخدمة لطباعة حرف التحكم Control Character أيضاً والجدول التالي يوضح بعض حروف التحكم)

الكود ASCII	الـرمــز	الوظيـفة
7	BEL (Beep)	إصدار صوت
8	BS (Back space)	مسافة للخلف (Back Space)
9	HT (Tab)	تحرك بمقدار Tab
A	LF (Line Feed)	سطر جديد
D	CR (Carriage return)	بداية السطر الحالي

بعد التنفيذ يحصل المسجل AL علي شفرة ASCII لحرف التحكم

<u>البرنامج الأول:</u>

برنامجنا الأول سيقوم بقراءة حرف من لوحة المفاتيح ثم طباعة الحرف الذي تم إدخاله في بداية السطر التالي ثم إنهاء البرنامج.

يتكون البرنامج من الأجزاء التالية:

١- إظهار علامة الاستفهام "؟" على الشاشة

```
MOV  AH,2
MOV  DL,'?'
INT   21h
```

٢- قراءة حرف من لوحة المفاتيح

```
MOV   AH,1
INT    21h
```

٣- حفظ الحرف الذي تم إدخاله في مسجل آخر BL مثلاً و ذلك لأننا سنستخدم المسجل DL في تحريك المؤشر إلي بداية السطر الجديد وسيؤدي ذلك لتغيير محتويات المسجل AL (لاحظ أن الخدمة ٢ تقوم باستقبال الحرف المطلوب طباعته في المسجل DL وتقوم بإعادة الحرف المطبوع في المسجل AL مما يجعلنا نفقد القيمة المسجلة فيه) وبالتالي يجب تخزين محتوياته في مسجل آخر مثل BL

```
MOV   BL , AL
```

٤- لتحريك المسجل إلي بداية السطر الجديد يجب طباعة حرف التحكم Carriage Return و Line Feed ويتم ذلك كالآتي

```
MOV   AH,2
MOV   DL,0dh ; Carriage Return
INT    21h
MOV   DL,0ah ; Line Feed
INT    21h
```

٥- طباعة الحرف الذي تم إدخاله (لاحظ انه تم تخزينه في المسجل BL في الخطوة (٣)

```
MOV   DL , BL
INT    21h
```

٦- إنهاء البرنامج و العودة الى نظام التشغيل ويتم ذلك بوضع الرقم 4Ch في المسجل AH واستدعاء نداء المقاطعة رقم 21h.

```
MOV   AH,4CH
INT    21h
```

و على ذلك يصبح البرنامج على الصورة التالية:

```
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
        اظهار علامة التعجب ;
        MOV AH,2    ; طباعة حرف
        MOV DL,'?'  ; الحرف المطلوب طباعته
        INT 21H
        قراءة حرف من لوحة المفاتيح;
        MOV AH,01  ; قراءة حرف
        INT 21H
        MOV BL,AL  ; تخزين الحرف
        الذهاب إلى سطر جديد ;
        MOV AH,02
        MOV DL,0DH ; carriage return
        INT 21H
        MOV DL,0AH ; line feed
        INT 21H
        طباعة الحرف الذي تم إدخاله ;
        MOV DL,BL  ; ; إحضار الحرف من المسجل
        INT 21H
        العودة إلى نظام التشغيل DOS ;
        MOV AH,4CH
        INT 21H
```

```
MAIN ENDP

END MAIN
```

لاحظ أنه عندما يتوقف البرنامج فإنه يحول التحكم للـ DOS بتنفيذ INT 21h الوظيفة 4Ch

إنشاء وتشغيل البرنامج:-

في هذا الجزء سنوضح طريقة إنشاء و تجهيز البرنامج للتشغيل حيث يتضمن ذلك الخطوات التالية:-

١- استخدام أي برنامج Text Editor لكتابة البرنامج الموضح في المثال السابق. (ملف برنامج المصدر)

٢- استخدام الـ ASSEMBLER لتوليد الملف المسمى OBJECT FILE.

٣- استخدام برنامج الربط LINKER لربط ملفات الـ OBJECTلتوليد ملف التشغيل EXECUTABLE FILE.

٤- تشغيل البرنامج.

فيما يلي توضيح بالتفصيل كل خطوة من الخطوات السابقة:-

١- إنشاء ملف البرامجSOURCE FILE:-

يتم استخدام أي محرر نصوص Editor لكتابة البرنامج ويمكن استخدام أي محرر ينتج ملف نصي عادى Text Editor مثل EDIT يتم عادة تخزين الملف بامتداد (Extention) ASM مثلا المثال السابق نحفظ الملف بالاسم FIRST.ASM.

٢- تجميع البرنامجASSEMBLE THE PROGRAM:-

ويتم هذا عن طريق معالجة البرنامج بواسطة أحد الـ Assembler مثل MASM(Microsoft Macro Assembler) أو TASM(Turbo Assembler) و التي تقوم بتحويل الملف الأصلي الذي يحتوى على البرنامج المكتوبة بلغة التجميع إلى ملف اقرب إلى لغة الآلة يسمى(OBJECT FILE). وأثناء هذه العملية يتم التعامل مع الملف والتأكد من عدم وجود أي خطأ في كتابة البرنامج حيث يتم الرجوع إلي الخطوة (1) وتحديد الأخطاء و تصحيحها حتى نحصل على رسالة بعدم وجود أخطاء في البرنامج.

واستخدام البرنامج TASM أوMASM يتم على النحو التالي:

```
TASM FILENAME;

أو MASM FILENAME;
```

في هذا الجزء سنستخدم برنامج TASM والجزء التالي يوضح هذه العملية:-

```
>TASM FIRST.asm
TURBO ASSEMBLER VERSION 3.1 COPYRGHT(C)1988,1992BRLAND
INTERNATIONAL
ASSEMBLING FILE: FIRST.SAM
ERROR MESSAGE: NONE
WARNING MESSAGE:NONE
PASSES: 1
```

السطر الأول يوضح نوع الـASSEMBLER والسطر الثاني يوضح اسم الملف يليه سطرين بالأخطاء التي توجد في البرنامج.

لاحظ أنه إذا كان هناك أي خطأ في البرنامج الأصلي يتم إظهار رسالة تحوي رقم السطر ونبذة سريعة عن الخطأ حيث يجب فتح الملف الأصلي first.asm وتصحيح الخطأ ثم العودة مرة أخرى وإعادة هذه الخطوة حتى نحصل على الملف first.obj.

٣-ربط البرنامج Linking the program

الملف الذي تم إنشاؤه في الخطوة السابقة هو ملف بلغة الآلة Machine Language ولكنه غير قابل للتنفيذ لأنه لا يحتوي على الشكل المناسب للبرامج القابلة للتنفيذ وذلك للأسباب التالية:

أ- عدم تعريف مكان تحميل الملف في الذاكرة وبالتالي فإن عمليه العنونة داخل البرنامج لا يمكن تنفيذها.

ب- بعض الأسماء والعناوين داخل البرنامج تكون غير معرفة بالذات في حالة ربط أكثر من برنامج حيث يتم من أحد البرامج نداء برامج فرعيه أخرى مكتوب في ملف آخر.

برنامج الربط Link Program يقوم بإجراء عملية الربط بين الـ Object Files المختلفة وتحديد العناوين داخل البرنامج ويقوم بعد ذلك بإنتاج ملف قابل للتنفيذ EXE . (Executable File) على النحو التالي:

```
> TLINK First
```

```
Turbo Link Version 2.0 Copyright (c) 1987 Borland International.
```

٤ – تنفيذ البرنامج Run The Program

لتشغيل البرنامج يتم فقط كتابة اسمه من محث الـDOS

```
C:\tasm > first
?t
t
C:\ASM >
```

يقوم البرنامج بطباعة الحرف "؟" والانتظار إلي حين الضغط علي مفتاح من لوحة المفاتيح. يقوم البرنامج بالذهاب إلي بداية السطر الجديد وطباعة الحرف الذي تم الضغط عليه ثم الانتهاء والعودة إلي نظام التشغيل.

البيانات المستخدمة في البرنامج Program Data

يقوم البرنامج بالتعامل مع البيانات في صورة أرقام ثنائية وفي برامج لغة التجميع يتم التعامل مع الأرقام في الصورة الثنائية أو السداسية عشر أو العشرية أو حتى في صورة حروف.

الأعداد Numbers

* يتم كتابة الأرقام الثنائية في صورة ٠ و١ وتنتهي الحرف B أو b للدلالة علي أن الرقم ثنائي Binary
* مثل 01010111B أو 11100011b
* الأرقام العشرية يتم كتابتها في الصورة المعتادة وبدون حرف في النهاية، كما يمكن أن تنتهي بالحرف D أو الحرف d دلالة علي أنها عشرية Decimal مثل 1234 و 1345d و –234D.
* الأرقام السداسية عشر يجب أن تبدأ برقم وتنتهي بالحرف H أو الحرف h للدلالة علي أنها سداسية عشر Hexadecimal مثل 0abh أو 56H. (السبب في استعمال 0 في المثال الأول لتوضيح أن المطلوب هو الرقم السداسي عشر ab وليس المتغير المسمى ab).

الجدول التالي يوضح بعض الأمثلة

الرقم	ملحوظات
10011	عشري
10011b	ثنائي
6455	عشري
-456h	سداسي عشر
FFFFh	خطأ (لا يبدأ برقم)
1,234	خطأ (يحتوي على حرف غير رقمي)
0ab	خطأ (لم ينتهي بالحرف h أو H)

الحروف Characters

يتم وضع الحروف والجمل داخل علامات التنصيص مثلاً 'A' أو 'SUDAN' ويتم داخلياً تحويل الحروف إلي الأرقام المناظرة في كود الـ ASCII بواسطة الـ Assembler وبالتالي تخزينها في الذاكرة وعلى ذلك لا يوجد فرق بين الحرف 'A' والرقم 41h (وهو الرقم المناظر للحرف A في الجدول) وذلك داخل البرنامج أو من ناحية التخزين في الذاكرة.

المتغيـرات VARIABLES

تلعب المتغيرات في لغة التجميع نفس الدور الذي تلعبه في البرامج باللغات ذات المستوى العالي High Level Programming Languages مثل لغة الباسكال والسي. وعلى ذلك يجب تحديد أسماء المتغيرات المستخدمة في البرنامج ونوع كل متغير حيث سيتم حجز مكان في الذاكرة لكل متغير وبطول يتناسب مع نوع المتغير وذلك بمجرد تعريف المتغير. ويتم استخدام الجدول التالي لتعريف المتغيرات في لغة التجميع حيث يشير كل إيعاز لنوع المتغير المطلوب تعريفه.

الايعـــاز	المعـــــــنـى
DB (Define Byte)	لتعريف متغير حرفي يشغل خانة واحدة في الذاكرة
DW (Define Word)	لتعريف متغير كلمة يشغل خانتين متتاليتين في الذاكرة
DD (Define Double Word)	لتعريف متغير يشغل أربعة خانات متتالية في الذاكرة
DQ (Define Quad Word)	لتعريف متغير يشغل ثمان خانات متتالية في الذاكرة
DT (Define Ten Bytes)	لتعريف متغير يشغل عشر خانات متتالية في الذاكرة

في هذا الجزء سنقوم بالتعامل مع المتغيرات من النوع DB و DW.

المتغيرات الحرفيـة Byte Variables:

يتم تعر يف المتغيرات الحرفية بالصورة التالية:

```
Name        DB    Initial_Value
```

مثلاً

```
Alpha       DB    4
```

يقوم هذا الإيعاز بتعريف متغير يشغل خانه واحدة في الذاكرة واسمه Alpha ويتم وضع قيمه ابتدائيــة مقدارها ٤ في هذا المتغير.

يتم استعمال علامة الاستفهام (؟) في حالة عدم وجود قيمه ابتدائية للمتغيــر.

مثال: Byte DB ?

القيــم التي يمكن تخزينها في هذا المتغير تتراوح بين ٠ و ٢٥٥ في حالة الأرقام التي يتم تخزينها بدون إشارةUnsigned Numbers و بين ١٢٨- و ١٢٧+ في حالة الأرقام التي يتــم تخزينها بإشارة Signed Numbers.

متغيرات الجمل Word Variables

يتم تعريف المتغير علي أنه من النوع Word ويتم تخزينه في خانتين من الذاكرة Two Bytes وذلك باستخدام الصيغة

name DW initial_value

مثلاً التعريف التالي

-2 DW WRD

يتم فيه تعريف متغير باسم WRD ووضع قيمة ابتدائية (الرقم –٢) فيه

كما في حالة المتغيرات الحرفية يتم وضع العلامة ؟ في حالة عدم وجود قيمة ابتدائية للمتغير.

يمكن للمتغير من النوع word تخزين أرقام تتراوح بين ٠ و ٦٥٥٣٥ (2^{16} –1) في حالة الأرقام بدون إشارة (الموجبة فقط)Unsigned Numbers

ويمكن تخزين الأرقام من –٣٢٧٦٨ (2^{15}-) وحتى ٣٢٧٦٧ (2^{15} - 1) في حالة الأرقام بإشارة

(الموجبة والسالبة)Signed Numbers.

المصفوفات Arrays

في لغة التجميع نتعامل مع المصفوفات علي أنها مجموعة من الحروف أو الكلمات المتراصة في الذاكرة في عناوين متتالية. فمثلاً لتعريف مصفوفة تحتوي علي ثلاثة أرقام من النوع الحرفي 3Bytes بقيم ابتدائية 10h و 20h و 30h علي الترتيب يتم استخدام التعريف التالي:

```
10h, 20h, 30h     DB     B_ARRAY
```

الاسم B_ARRAY يشير إلي العنصر الأول في المصفوف(العدد 10h) والاسم B_ARRAY + 1 يشير إلي العنصر الثاني والاسم B_ARRAY + 2 يشير إلي العنصر الثالث. فمثلاً إذا تم تخصيص عنوان الإزاحة 0200h للمتغير B_ARRAY يكون شكل الذاكرة كما يلي:

المحتوي	العنوان	الاسم (الرمز Symbol)
10h	٠200h	B_ARRAY
20h	0201h	B_ARRAY + 1
30h	0202h	B_ARRAY + 2

وبنفس الطريقة يتم تعريف مصفوف مكون من كلمات فمثلاً التعريف

```
W_ARRAY     DW     1000h, 2000h, 3000h
```

يقوم بتعريف مصفوف يحتوي علي ثلاثة عناصر بقيم ابتدائية 1000h و 2000h و 3000h علي الترتيب. يتم تخرين القيمة الأولى (1000h)في العنوان W_ARRAY والقيمة الثانية في العنوان W_ARRAY +2 والقيمة الثالثة في العنوان W_ARRAY + 4 وهكذا. فمثلاً لو تم تخزين المصفوف في الذاكرة بدءاً من العنوان 300h يكون شكل الذاكرة كما يلي:

المحتوي	العنوان	الاسم (الرمز Symbol)
1000h	0300h	W_ARRAY
2000h	0302h	W_ARRAY + 2
3000h	0304h	W_ARRAY + 4

لاحظ أن للمتغيرات من هذا النوع يتم تخزينها في الذاكرة في خانتين حيث يتم تخزين الخانة ذات الوزن الأقل Low Byte في الخانة الأولي والخانة ذات الوزن الأكبر High Byte في العنوان التالي مباشرة. فمثلاً إذا كان لدينا التعريف: Word1 DW 1234h

يتم تخزين الرقم 34h (الذي يمثل الخانة ذات الوزن الأقل) في العنوان word1 والرقم 12h (الذي يمثل الخانة ذات الوزن الأكبر) في العنوان word1 + 1.

الرسائل والنصوصCharacter Strings

يتم تخزين النصوص علي أنها سلسلة من الحروف ويتم وضع القيمة الابتدائية في صورة حروف أو القيم المناظرة للحروف في جدول الحروف ASCII Table فمثلاً التعريفان التاليان يؤديان إلي نفس النتيجة وهي تعريف متغير اسمه Letters ووضع القيمة الابتدائية "ABC' فيه

```
1 - Letters    db     'ABC'
2 – Letters    db     41h, 42h,43h
```

ويمكن دمج القيمة الابتدائية لتحوي الحروف والقيم المناظرة لها كما في المثال التالي

```
msg   db     0dh,0ah,'Sudan$'
```

ويتم هنا بالطبع التفرقة بين الحروف الكبيرة Capital Letters والحروف الصغيرة Small Letters.

الثوابت

يتم عادة استخدام الثوابت لجعل البرنامج أسهل من حيث القراءة والفهم وذلك بتعريف الثوابت المختلفة المستخدمة في البرنامج. يتم استخدام الإيعاز (EQU (EQUate لتعريف الثوابت علي النحو التالي:

```
name  EQU  Constant
```

حيث name هو اسم الثابت. مثلاً لتعريف ثابت يسمى LF بقيمة ابتدائية 0Ah نكتب

```
0Ah   EQU     LF
```

وبالتالي يمكن استخدام الثابت LF بدلاً عن الرقم 0Ah كالآتي MOV AL , LF بدلاً عن استخدام الآتي MOV AL,0Ah. حيث يقوم الـ Assembler بتحويل الثابت LF داخل البرنامج إلي الرقم 0Ah.

كذلك يمكننا استخدام المثال التالي

```
Prompt          EQU  'Type your Name'
Msg             DB   prompt
```

لاحظ أن EQU عبارة عن إيعاز وليس تعليمه أو أمر وبالتالي لا ينتج عنه تعريف متغير ووضعه في الذاكرة.

مثال١:

برنامج يقوم بوضع القيمه 5h في المتغير Xمن نوع DB وطباعتها ؟

ولأن هذا البرنامج يحتوي علي مقطع بيانات فإننا نحتاج إلي تجهيز المسجل DS لكي يشير إلي مقطع البيانات.

بادئة مقطع البرنامج (PSP (Program Segment Prefix

عندما يتم تحميل البرنامج في الذاكرة يقوم نظام التشغيل بتخصيص ٢٥٦ خانة للبرنامج وهي تسمي PSP. يحتوي الـ PSP علي معلومات عن البرنامج وعلي ذلك يستطيع البرنامج التعامل مع هذه المعلومات. يقوم نظام التشغيل DOS بوضع عنوان المقطع الخاص به في كل من المسجلين DS و ES قبل تنفيذ البرنامج ونتيجة لذلك فإن مسجل مقطع البيانات DS لا يحتوي علي عنوان مقطع البيانات الخاص بالبرنامج ولعلاج هذه المشكلة فإن أي برنامج يحتوي علي مقطع بيانات يجب أن يبدأ بتجهيز مسجل مقطع البيانات ليشير إلي مقطع البيانات الخاص بالبرنامج علي النحو التالي

```
MOV  AX, @DATA
MOV  DS, AX
```

حيث @DATA هو عنوان مقطع البيانات الخاص بالبرنامج والمعرف بـ.DATA حيث يقوم الـ ASSEMBLER بتحويل الاسم @DATA إلي رقم يمثل عنوان المقطع ولأننا لا نستطيع تخزين النتيجة في المسجل DS مباشرة فقد استعنا بمسجل عام AX كمسجل وسيط يتم وضع القيمة فيه أولاً وبعد ذلك يتم نقلها إلي المسجل DS.

```
.MODEL SMALL

.STACK 100H

.DATA

 X DB 35h

.CODE

MAIN PROC

 ; initialize DS

 MOV AX,@DATA

 MOV DS,AX


MOV AH,02H

mov dl,x

 INT 21H

 ;return to DOS

 MOV AH,4CH

 INT 21H

 MAIN ENDP

END MAIN
```

إظهار رسالة على الشاشة Display String

في البرنامج السابق تم استخدام الوظيفة رقم ١ من نداء المقاطعة رقم 21h وهي تستخدم لاستقبال حرف من لوحة المفاتيح وكذلك الوظيفة رقم ٢ وهي لطباعة حرف علي الشاشة.

في هذا المثال ولإظهار رسالة كاملة علي الشاشة يتم استخدام الخدمة رقم ٩

خدمة رقم ٩ : إظهار رسالة علي الشاشة

المدخلات : عنوان الإزاحة Offset لبداية الرسالة يتم وضعه في المسجل DX

(يجب أن تنتهي الرسالة بالحرف "$")

الحرف "$" في نهاية الرسالة لا تتم طباعته علي الشاشة. وإذا احتوت الرسالة علي أي حرف تحكم Control Character فإنه يتم تنفيذه أثناء الطباعة.

لتوضيح هذه العملية سنقوم بكتابة برنامج يقوم بإظهار الرسالة 'Hello!' في الشاشة. يتم تعريف هذه الرسالة في مقطع البيانات بالطرقة التالية

```
msg   db     'HELLO!$'
```

الأمر LEA

تحتاج الخدمة رقم ٩ في نداء المقاطعة INT 21h إلي تجهيز عنوان إزاحة الرسالة في المسجل DX ولعمل ذلك يتم تنفيذ الأمر LEA (Load Effective Address)

```
Destination , Source    LEA
```

حيث المستودع هو أحد المسجلات العامة والمصدر هو اسم المتغير الحرفي (موقع في الذاكرة). يقوم الأمر بوضع عنوان الإزاحة للمتغير المصدر في المسجل المستودع. فمثلاً الأمر

```
DX, MSG    LEA
```

يقوم بوضع قيمة الإزاحة لعنوان المتغير msg في المسجل DX.

```
.MODEL SMALL
.STACK 100H
.DATA
 MSG DB 'HELLO!$'
.CODE
MAIN PROC
 ; initialize DS
 MOV AX,@DATA
 MOV DS,AX
 ;display message
 LEA DX,MSG              ; احصل علي الرسالة
 MOV AH,09H              ; وظيفة عرض السلسلة
 INT 21H
 ;return to DOS
 MOV AH,4CH
 INT 21H                 ; الخروج الي نظام التشغيل
MAIN ENDP
END MAIN
```

بعض الأوامر الأساسية

في هذه المحاضرة سيتم التعرف على بعض الاوامر الأساسية حيث سيتم دراستها في محاضرات مستقلة بشكل اوسع مع أوامر آخري

١- الأمر XCHG (Exchange)

يستخدم الأمر XCHG لاستبدال قيمة مسجلين أو لاستبدال قيمة مسجل مع موقع محدد في الذاكرة (متغير). والصيغة العامة للأمر هي:

```
XCHG        Destination, Source
```

مثال:

```
XCHG AH, BL
```

حيث يتم تبادل قيم المسجلين AH, BL (تصبح قيمة AH تساوى قيمةBL وBL تساوى AH).

مثال:

الأمر التالي يقوم باستبدال قيمة المسجل AX مع المتغير WORD1

```
XCHG AX, WORD1
```

الجدول التالي يوضح قيود استخدام الأمر XCHG

	المستودع	
المصدر	**مسجل عام**	**موقع في الذاكرة**
مسجل عام	**مسموح**	**مسموح**
موقع في الذاكرة	**مسموح**	**غير مسموح**

لاحظ عدم السماح للتعليمتين MOV أو XCHG بالتعامل مع موقعين في الذاكرة في أمر واحد مثل MOV Word1,Word2 ولكن يمكن تفادي هذا القيد باستخدام مسجل وسيط فيصبح الأمر كما يلي:

```
Mov    AX , Word2
Mov    Word1 , AX
```

برنامج بلغة التجميع استبدال قيمة المتغير Xمع المتغير Y؟

```
model small.
stack 100h.
.data
x db 41h
y db 42h
"$msg db " x replace Y
cr  equ 0dh
lf equ 0ah
.code

main proc
mov ax,@DATA
mov ds,ax
mov al,x

xchg al,y
mov x,al
Lea dX,msg
mov ah,09h
int 21h

mov ah,2h
mov dl,cr
int 21h

mov ah,2h
mov dl,lf
int 21h

mov ah,2h
mov dl,x
int 21h

mov ah,2h
mov dl,lf
int 21h

mov dl,y
int 21h

mov ah,4ch
int 21h

main Endp
end main
```

٣ - العمليات الحسابية ADD, SUB, INC, DEC, NEG:

يتم استخدام الأمرين ADD و SUB لجمع أو طرح محتويات مسجلين أو مسجل وموقع في الذاكرة أو موقع في الذاكرة مع مسجل أو مسجل مع موقع في الذاكرة والصيغة العامة للأمرين هي:-

```
ADD  Destination, Source
SUB  Destination, Source
```

مثلاً الأمر

```
ADD WORD1, AX
```

يقوم بجمع محتويات المسجل AX إلي قيمة المتغير WORD1 ويتم تخزين النتيجة في المتغير WORD1 (لا يتم تغيير قيمة محتويات المسجل AX بعد تنفيذ الأمر) كذلك الأمر

```
sub AX, DX
```

حيث يتم طرح محتويات المسجل DX من المسجل AX ويتم تخزين النتيجة في المسجل AX (لاحظ أن محتويات المسجل DX لا تتغير بعد تنفيذ الأمر)

الجدول التالي يبين قيود استعمال الأمرين ADD و SUB

	المستودع	
المصدر	**مسجل عام**	**موقع في الذاكرة**
مسجل عام	**مسموح**	**مسموح**
موقع في الذاكرة	**مسموح**	**غير مسموح**
ثابت	**مسموح**	**مسموح**

لاحظ أنه غير مسموح بالجمع أو الطرح المباشر بين مواقع في الذاكرة في أمر واحد وبالتالي فإن الأمر ADD BYTE1, BYTE2 غير مسموح به ولكن يمكن إعادة كتابته على الصورة:

MOV AL, BYTE2 ; حيث يتم قيمة المتغير إلى مسجل قبل عملية الجمع

ADD BYTE1, AL

الأمر ADD BL,5 يقوم بجمع الرقم ٥ إلى محتويات المسجل BL وتخزين النتيجة في المسجل BL.

كملاحظة عامه نجد انه يجب أن يكون المتغيرين لهما نفس الطول بمعني أن الأمر التالي غير مقبول

```
MOV AX ,BYTE1
```

وذلك لأن طول المتغير BYTE هو خانه واحدة أما المسجل AX فان طوله هو خانتين -2 BYTE. (أي أن المتغيرات (المعاملات) يجب أن تكون من نفس النوع)

بينما نجد الـ ASEMBLER يستقبل الأمر

```
MOV AH, 'A' ( مادام AH بايت فإن المصدر يجب أن يكون كذلك بايت )
```

حيث يتم وضع الرقم 41h في المسجلAH ويقوم أيضا بتقبل الأمر

```
MOV AX ,'A' ( مادام AX كلمة فإن المصدر يجب أن يكون كذلك كلمة )
```

حيث سيتم وضع الرقمh0041 في المسجل AX.

الأوامر INC (Increment) , DEC (Decrement) , NEG

أما الأمرين INC ,DEC يتم فيها زيادة أو نقصان قيمه مسجل أو موقع في الذاكرة بمقدار ١ والصيغة العامة لها هي:

```
INC    Destination  ; Destination = Destination +1

DEC   Destination  ; Destination = Destination - 1
```

فمثلا الأمرINC WORD1 يقوم بجمع ١ إلى محتويات المتغيرWORD1

بينما الأمرDEC WORD2 يقوم بإنقاص الرقم ١ من محتويات المتغيرWORD2.

أخيراً نتحدث عن الأمر(NEG(Negate والذي يستعمل لتحويل إشارة الرقم الموجب إلي رقم سالب والرقم السالب يتم تحويله إلي رقم موجب وذلك بتحويله إلى المكمل لاثنين

2'S Complement والصيغة العامة للأمر هي:

NEG Destination

حيث يتم التعامل مع أحد المسجلات أو موقع في الذاكرة

مثـــال:

```
NEG BX      ; BX = -BX
NEG BYTE    ; BYTE = -BYTE.
```

كيف تؤثر العمليات على البيارق:

عندما يقوم المعالج بتنفيذ أي أمر يتم رفع البيارق المناسبة لتوضيح النتيجة . وعموماً هناك أوامر لا تؤثر في كل البيارق وإنما تؤثر في بعضها فقط إذ قد تترك كل البيارق دون تأثير . وعموماً فإن عملية تفرع البرنامج باستخدام أوامر التفرع JUMP INSTRUCTIONS تعتمد عملياً على قيم البيارق المختلفة كما سنري فيما بعد .

الأمر	البيارق المتأثرة
MOV / XCHG	لا تتأثر أي من البيارق
ADD / SUB	تتأثر كل البيارق
INC / DEC	تتأثر كل البيارق عدا بيرق المحمول (CF)
NEG	تتأثر البيارق (CF=1 إلا إذا كانت النتيجة تساوي ٠ ، 0F=1 إذا كان المعامل هو الرقم 800H في حالة WORD أو 80h في حالة المعامل Byte)

برنامج تحويل حالة الحروف A Case Conversion Program:

في هذا المثال سنقوم بسؤال المستخدم ليقوم بإدخال حرف صغير lower-case letter يقوم البرنامج بإظهار رسالة تطبع الحرف الذي تم إدخاله بعد تحويله إلى صورة حرف كبير upper-case letter مثلاً

Enter A Lower Case Letter: a

In Upper Case It Is: A

سيتم في هذا البرنامج استخدام الإيعاز EQU لتعريف كل من CR,LF

```
CR    EQU  0DH
LF    EQU  0AH
```

بينما يتم تعريف الرسائل على النحو التالي

```
MSG1        DB    'Enter A Lower Case Letter:$'
MSG2        DB    CR,LF,' In Upper Case It Is: '
Char  DB    ? ,'$'
```

عند تعريف المتغير char تم تعريفه بعد الرسالة MSG2 مباشرة وذلك لأن البرنامج سيقوم بإظهار الرسالة msg2 متبوعة مباشرة بالحرف char (وهو الحرف الذي تم إدخاله بعد تحويله إلى - Upper case ويتم ذلك بطريقة طرح الرقم 20h من الحرف الذي تم إدخاله)

تم تعريف حروف التحكم CR,LF قبل الرسالة msg2 بهدف جعل الرسالة تبدأ من بداية السطر الجديد.

ولأن الرسالة msg2 لا تنتهي بعلامة نهاية الرسالة '$' فإنه سيتم الاستمرار في الطباعة وطباعة الحرف char في الشاشة (لاحظ أن العلامة '$' توجد في نهاية المتغير char مباشرة).

يبدأ البرنامج بإظهار الرسالة msg1 ثم قراءة الحرف من لوحة المفاتيح

```
LEA   DX ,msg1
MOV   AH ,9
INT   21h
MOV   AH ,1
INT   21h
```

بعد ذلك يتم تحويل الحرف إلى حرف كبير upper-case وذلك بطرح العدد 20h من الحرف (وذلك لأن الفرق بين الحروف الكبيرة والصغيرة في جدول ASCII هو العدد 20h حيث تبدأ الحروف الكبيرة ابتداءً من 41hبينما تبدأ الحروف الصغيرة ابتداءً من 61h) ويتم تخزين النتيجة في المتغير char

```
SUB   AL,20h        ; حوله الي حرف كبير
MOV   char ,AL      ; ثم خزِنهُ في المتغير
```

بعد ذلك يقوم البرنامج بإظهار الرسالة الثانية **msg2** وتطبع متبوعة بالمتغير **char** كما ذكرنا سابقاً. وفيما يلي نص البرنامج:

```
.MODEL SMALL
.STACK 100H
.DATA
        CR     EQU   0DH
        LF     EQU   0AH
        MSG1 DB      'ENTER A LOWER CASE LETTER: $'
        MSG2 DB      CR,LF,'IN UPPER CASE IT IS:'
        CHAR DB      ?,'$'
.CODE
MAIN PROC
        ; initialize DS
        MOV   AX,@DATA
        MOV   DS,AX
        ;print user prompt
        LEA   DX,MSG1
        MOV   AH,09H
        INT    21H
        ;input character and convert to lower case
        MOV   AH,01H
        INT    21H
        SUB   AL,20H
```

```
        MOV   CHAR,AL
        ;display on the next line
        LEA   DX,MSG2
        MOV   AH,09H
        INT   21H
        ;return to DOS
        MOV   AH,4CH
        INT   21H
MAIN    ENDP
        END   MAIN
```

التفرع وتعليمات ضبط الانسياب Flow Control Instructions

لكي نكتب برنامج يقوم بعمل محدد غالبا ما يتم استخدام أوامر التفرع التي تجعل المبرمج قادراً علي اتخاذ قرارات محددة وتؤدي أوامر التفرع والتكرار إلي تنفيذ برامج فرعية ويعتمد هذا التفرع أو التكرار عادة علي قيم محددة للمسجلات وذلك عن طريق بيارق الحالة Status Flags والتي تتأثر دائماً بآخر عملية تم تنفيذها.

التفرع المشروط CONDITIONAL JUMP

الأمر JNZ السابق هو مثال لأوامر التفرع المشروط. و يكون أمر التفرع المشروط على الصورة

Jxxx destination-Label

فإذا تحقق الشرط المحدد يتم تفرع البرنامج إلى العنوان الموضح كمعامل للأمر، ويكون الأمر التالي هو الأمر الموجود في العنوان المحدد .أما إذا لم يتحقق الشرط يتم الاستمرار كالمعتاد إلى الأمر التالي مباشرة .

في حالة التفرع يجب أن يكون العنوان الذي سيتم التفرع عليه على بعد ١٢٦ قبل العنوان الحالي أو ١٢٧ بعد العنوان الحالي وسنرى فيما بعد كيفية التفرع إلى أماكن أبعد من هذا المدى .

كيف يقوم المعالج بتنفيذ عملية التفرع المشروط ؟

يقوم المعالج باستخدام البيارق لتحديد عملية التفرع . حيث أن البيارق تعكس الحالة بعد تنفيذ آخر عملية وبالتالي فإن أوامر التفرع يجب أن تعتمد على بيرق محدد أو بيارق محددة حيث يتم التفرع إذا تم رفع هذه البيارق .

إذا تحقق التفرع يقوم المعالج بتحميل مؤشر التعليمات IP بالقيمة المحددة بالعنوان الموجود في أمر التفرع .أما إذا لم يتم تحقق الشرط فإن مؤشر التعليمات يواصل إلى العنوان التالي مباشرة .

نجد الأمر

JNZ PRINT-LOOP

وهذا يعني أنه إذا كان بيرق الصفر لا يساوي واحد ZF= 0 فإنه يتم التفرع إلى العنوان PRINT-LOOP وذلك بتحميل مؤشر التعليمات بالعنوان . أما إذا كانت النتيجة تساوي الصفر (ZF= 1) فإن البرنامج يواصل إلى الخطوة التالية.

تنقسم أوامر التفرع المشروط إلى ثلاثة مجموعات :

* المجموعة الأولى التفرع بالإشارة Signed Jumps وتستخدم في حالة استخدام الأرقام بالإشارة Singed Numbers
* المجموعة الثانية التفرع بدون إشارة Unsigned Jumps وتستخدم في حالة استخدام الأرقام بدون إشارة Unsigned Numbers .
* التفرع ببيرق واحد Single Flag Jumps والتي تعتمد على بيرق محدد .

الجداول التالية توضح أوامر التفرع المختلفة . لاحظ أن الأمر قد يأخذ أكثر من اسم مثلا JG و JNLE حيث تعني تفرع إذا كانت النتيجة أكبر JG أو تفرع إذا كانت النتيجة ليست أصغر من أو تساوي . ويمكن استخدام أي من الأمرين لأنهما يؤديان إلى نفس النتيجة .

١-التفرع بالإشارة Signed Jumps

الأمر	الوصف	شرط التفرع
JG / JNLE	تفرع في حالة أكبر من (ليس أصغر من أو يساوي)	ZF=0 & SF=OF
JGE / JNL	تفرع في حالة أكبر من أو يساوي (ليس أصغر من)	SF=OF
JL / JNGE	تفرع في حالة أقل من (ليس أكبر من أو يساوي)	SF<>OF
JLE / JNG	تفرع في حالة أقل من أو يساوي (ليس أكبر من)	ZF=1 OR SF<>OF

٢-التفرع بدون إشارة Unsigned Jumps

الأمر	الوصف	شرط التفرع
JA / JNBE	تفرع في حالة أكبر من (ليس أصغر من أو يساوي)	CF=0 & ZF=0
JAE / JNB	تفرع في حالة أكبر من أو يساوي (ليس أصغر من)	CF=0
JB / JNAE	تفرع في حالة أقل من (ليس أكبر من أو يساوي)	CF=1
JBE / JNA	تفرع في حالة أقل من أو يساوي (ليس أكبر من)	CF=1 OR ZF=1

٣-التفرع ببيرق واحد Single Flag Jumps

الأمر	الوصف	شرط التفرع
JE / JZ	تفرع في حالة التساوي أو الصفر	ZF=1
JNE / JNZ	تفرع في حالة عدم التساوي (لا يساوي الصفر)	ZF=0
JC	تفرع في حالة محمول Carry	CF=1
JNC	تفرع في حالة عدم وجود محمول Carry	CF=0
JO	تفرع في حالة الفيضان	OF=1
JNO	تفرع في حالة عدم حدوث الفيضان	OF=0
JS	تفرع في حالة النتيجة سالبة	SF=1
JNS	تفرع في حالة النتيجة موجبة	SF=0
JP / JPE	تفرع في حالة التطابق الزوجي	PF=1
JNP / JPO	تفرع في حالة التطابق الفردي	PF=0

الأمر CMP

الأمر Compare(CMP) يستخدم لمقارنة رقمين ويأخذ الصيغة :

```
CMP   Destination , Source
```

يقوم البرنامج بعملية المقارنة عن طريق طرح المصدر source من المستودع destination ولا يتم تخزين النتيجة ولكن البيارق تتأثر ، لا يقوم الأمر CMP بمقارنة موضعين في الذاكرة كما أن المستودع destination لا يمكن أن يكون رقم ثابت .

لاحظ أن الأمر CMP يماثل تماما الأمر SUB فيما عدا أن النتيجة لا يتم تخزينها .

افترض أن البرنامج يحتوي على التالي:

```
CMP   Ax , Bx
JG    Below
```

حيث AX=7FFFh، BX=0001h فان نتيجة الأمر CMP Ax,Bx هي:

```
7FFFh - 0001h = 7FFEh
```

والتفرع هنا يتم حيث أن البيارق تكون zf = sf = of =0 والأمر JG يتطلب أن تكون

Zf = 0 و كذلك Sf = Of وعلى هذا يتم التفرع إلى العنوان المحدد Below .

في حالة التفرع المشروط ورغم أن عملية التفرع تتم حسب حالة البيارق المختلفة فان المبرمج ينظر إلى الأمر بدون تفاصيل البيارق فمثلا :

```
CMP     AX,BX
JG      Below
```

إذا كان الرقم الموجود في المسجل AX أكبر من الرقم الموجود في المسجل BX فان البرنامج يتفرع إلى العنوان Below .

<u>التفرع بإشارة والتفرع بدون إشارة:</u>

كل أمر تفرع بإشارة يناظره أمر تفرع بدون إشارة ، مثلا الأمر JG يناظره الأمر JA واستخدام أي منهما يعتمد على طريقة التعامل مع الأرقام داخل البرنامج. حيث أن الجدول السابق قام بتوضيح أن كل عملية من هذه العمليات تعتمد على بيارق محددة حيث أن التفرع بإشارة يتعامل مع البيارق zf , sf , of بينما التفرع بدون إشارة يعتمد على البيارق zf , cf واستخدام الأمر غير المناسب قد يؤدي إلى نتائج غير صحيحة .

مثلا إذا استخدمنا الأرقام بإشارة وكان المسجل Ax يحتوي على الرقم 7fffhوالمسجل Bx يحتوي على الرقم 8000h وتم تنفيذ الأوامر التالية :

```
CMP   AX,BX
JA    Below
```

فبالرغم من أن 7EFF > 8000h في حالة الأرقام بإشارة فان البرنامج لن يقوم بالتفرع إلى العنوان Below وذلك لأن 7FFFh < 8000h في حالة الأرقام بإشارة ونحن نستعمل الأمر JA الذي يتعامل مع الأرقام بدون إشارة .

<u>التفرع الغير مشروط Unconditional Jump</u>

يستخدم الأمر JMP للتفرع إلي عنوان محدد وذلك بدون أي شروط حيث الصيغة العامة للأمر هي:

```
Jmp   Destination
```

ويكون العنوان الذي سيتم التفرع إليه داخل مقطع البرنامج الحالي وعلي ذلك فإن المدى الذي يمكن التفرع

إليه أكبر من حالة التفرع المشروط.

هيكلية البرنامج

ذكرنا أن عمليات التفرع يمكن استخدامها في التفرع والتكرار ولأن أوامر التفرع بسيطة سنتطرق في هذا الجزء لكيفية كتابة أوامر التكرار والتفرع والمستخدمة في لغات البرمجة الراقية High Level Programming Languages .

أوامر التفرع

الأمر IF……..Then…….

الشكل العام لعبارة If..Then… هو

```
IF  condition is True then
        Execute True branch statements
End_IF
```

أي إذا تحقق الشرط يتم تنفيذ الأوامر وإذا لم يتحقق لا يتم تنفيذ شيء

بلغة التجميع تصبح

```
Cmp op1,op2
Jcondition THEN
Jmp ENDIF
THEN:
Statement1
EndIF
```

مثال إذاكان x=0eh و y=0ah اطبع الرسالة "x is greater" إذا كان xأكبر من y؟

```
If x>y then
        Cout<<"x is greater"        End_IF
```

بلغة التجميع تصبح

```
.model small
.stack 100h
.data
x db 0eh
y db 0ah
msg db " x is greater$"
.code
main proc
mov ax,@DATA
mov ds,ax
mov al,x
cmp al,y
jg TTHEn
jmp ENDIFF
TTHen:
lea dx,msg
mov ah,09h
int 21h
ENDIFF:
mov ah,4ch
int 21h
main Endp
end main
```

٢ – عبارة IF…THEN…..ELSE…..ENDIF

وهي تكون علي الصورة

```
IF  Condition is True then
        Execute True_Branch statements
ELSE
        Execute False_Branch statements
End_IF
```

بلغة التجميع :

```
Cmp op1,op2
Jcondition THEN
Jmp  ELSE

THEN:
Statement1
Jmp ENDIF
ELSE:
Statement2
EndIF
```

مثال:

إذا كان x=0eh و y=0ah اطبع الرسالة "x is greater" إذا كان xأكبر أو يساوي y

وإلا اطبع الرسالة "Y is greater" ؟

```
. model small
.stack 100h
.data
x db 09h
y db 0ah
msg1 db " x is greater$ "
msg2 db " y is greater$"

.code
main proc
mov ax,@DATA
mov ds,ax

mov al,x

cmp al,y
jge TTHEn

jmp eelse

TTHen:
lea dx,msg1
mov ah,09h
int 21h
jmp ENDIFF
```

```
eelse:
lea dx,msg2
mov ah,09h
int 21h

ENDIFF:
mov ah,4ch
int 21h

main Endp
end main
```

إذا تحقق الشرط يتم تنفيذ مجموعة من الأوامر وإذا لم يتحقق يتم تنفيذ مجموعة أخرى من الأوامر

مثال:-

٣- عبارة CASE

في حالة عبارة CASE يوجد أكثر من مسار يمكن أن يتبعه البرنامج والشكل العام للأمر هو :

```
switch(Selection){
Case value1: Statement1 break;
Case value2: Statement2 break;
Case value3: Statement3 break;
Default:Statement5;
}
```

في لغة التجميع :

```
Cmp select,select1

Je caswvalue1

Cmp select,select2

Je caswvalue1

Cmp select,select3

Je caswvalue1
```

مثال اكتب برنامج يستقبل رقم من ١ الى ٥ و طباعة كتابة؟

```
.model small
.stack 100h
.data
msg1 db "one"$
msg2 db "two"$
msg3 db "three"$
msg4 db "four"$
msg5 db "five"$
msg6 db "ERROR"$

.code
main proc
mov Ax,@DATA
mov DS,AX

mov ah,01h
int 21h

sub al,30h

cmp  al,1
je VALUE1
cmp  al,2
je VALUE2

cmp  al,3
je VALUE3
```

```
cmp  al,4
je VALUE4

cmp  al,5
je VALUE5

jmp DEFAULTT

VALUE1:
LEA DX,msg1
jmp ENDSWITCH

VALUE2:
LEA DX,msg2
jmp ENDSWITCH

VALUE3:
LEA DX,msg3
jmp ENDSWITCH

VALUE4:
LEA DX,msg4
jmp ENDSWITCH

VALUE5:
LEA DX,msg5
jmp ENDSWITCH

DEFAULTT:
LEA DX,msg6
jmp ENDSWITCH

ENDSWITCH:


mov ah,09h
int 21h

mov ah,4ch
int 21h

main Endp
end main
```

٤ التفرع بشروط مركبة Compound Conditions

في بعض الأحيان يتم استعمال شرط مركب لعملية التفرع مثل

IF condition1 AND condition2

أو IF condition1 OR condition2

حيث في الحالة الأولي تم استخدام الشرط "و" AND وفي الحالة الثانية تم استخدام الشرط "أو" OR

الشرط "أو" OR Condition

يتحقق الشرط "أو" إذا تحقق أي من الشرطين أو كلاهما

بلغة التجميع :

```
Cmp op1,op2
J condition1  OR_TRUE
ORR:
Cmp op3,op4
J condition2  OR_TRUE
Jmp OR_FALSE


OR_TRUE :
Instruction when true
OR_FALSE:
```

مثال : حول البرنامج التالي الي لغة التجميع؟

```
If ( x> y || z>y  )
Cout<<" compound  conditions satisfied";
```

حيث أن :X=09h,Y=0ah,Z=0eh

```
.model small
.stack 100h
.data
```

```
x db 09h
y db 0ah
z db 0eh
msg1 db "compound  conditions satisfied"$
.code
main proc
mov Ax,@DATA
mov DS,AX

mov al,y
cmp x,al
jg OR_TRUE

ORR:
cmp z,al
jg OR_TRUE

jmp ENDIFF

 OR_TRUE:
 LEA DX,msg1
mov ah,09h
int 21h

ENDIFF:
```

```
mov ah,4ch
int 21h


main Endp
end main
```

مثال : حول البرنامج التالي إلي لغة التجميع؟

```
If ( x> y || z>y  )
Cout<<" compound  conditions satisfied";
Else
Cout<<" compound  conditions  not satisfied";
```

حيث أن :X=09h,Y=0ah,Z=03h

```
.model small
.stack 100h
.data
x db 09h
y db 0ah
z db 03h
msg1 db "compound  conditions satisfied$  "
msg2 db "compound  conditions not satisfied$ "
.code
main proc
mov Ax,@DATA
mov DS,AX
mov al,y
```

```
cmp x,al
jg OR_TRUE

ORR:
cmp z,al
jg OR_TRUE
jmp OR_ELSE

 OR_TRUE :
LEA DX,msg1
mov ah,09h
int 21h
jmp ENDIFF

OR_ELSE:
 LEA DX,msg2
mov ah,09h
int 21h
ENDIFF:
mov ah,4ch
int 21h
main Endp
end main
```

الشرط "أو" AND Condition

تكون نتيجة الشرط "و" صحيحة إذا تحقق كل من الشرطين في آن واحد

بلغة التجميع :

```
Cmp op1,op2
J condition1  ANDD
OR_FALSE:
ANDD:
Cmp op3,op4
J condition2  AND_TRUE
Jmp AND_FALSE


AND_TRUE :
Instruction when true
AND_FALSE:
```

مثال : حول البرنامج التالي الي لغة التجميع؟

```
If ( x> y && z>y  )
Cout<<" compound  conditions satisfied";
```

حيث أن : X=0Dh,Y=09h,Z=0eh

```
. model small
.stack 100h
.data
x db 0Dh
y db 09h
z db 0eh
msg1 db "compound  conditions satisfied$"
.code
main proc
mov Ax,@DATA
mov DS,AX

mov al,y
cmp x,al
jg ANDD
jmp ENDIFF
ANDD:
cmp z,al
jg AND_TRUE
```

```
jmp ENDIFF


 AND_TRUE:
 LEA DX,msg1
mov ah,09h
int 21h


ENDIFF:
mov ah,4ch
int 21h


main Endp
end main
```

مثال : حول البرنامج التالي إلي لغة التجميع؟

```
If ( x> y && z>y  )
Cout<<" compound  conditions satisfied";
Else
Cout<<" compound  conditions  not satisfied";
```

حيث أن :X=09h,Y=0Dh,Z=0Eh

```
.model small
.stack 100h
.data
x db 09h
```

```
y db 0Dh
z db 0eh
msg1 db "compound  conditions satisfied$"
msg2 db "compound  conditions not satisfied$"

.code
main proc
mov Ax,@DATA
mov DS,AX

mov al,y
cmp x,al
jg ANDD
jmp AND_ELSE
ANDD:
cmp z,al
jg AND_TRUE

jmp AND_ELSE


 AND_TRUE:
 LEA DX,msg1
mov ah,09h
int 21h
```

```
jmp ENDIFF
AND_ELSE:
LEA DX,msg2
mov ah,09h
int 21h
ENDIFF:
mov ah,4ch
int 21h


main Endp
end main
```

برنامج DEBUG :

يمكن باستخدام برنامج DEBUG متابعة تنفيذ البرنامج خطوة_خطوة وإظهار النتيجة وتأثر المسجلات بعد كل خطوة كما يمكن كتابة برنامج بلغة التجميع حيث يقوم بتحويله إلي لغة الآلة مباشرة وتخزينها في الذاكرة ولاستعمال برنامج الـ DEBUG نقوم بكتابة برنامج بلغة التجميع وتجهيزه حتى نحصل علي الملف القابل للتنفيذ EXCUTABLE FILE بعد ذلك يمكننا تحميل البرنامج بواسطة الأمر

```
C:\DOS\DEBUG TEST.EXE
```

يقوم البرنامج بالرد بالإشارة "-" دليل علي أنه في حالة انتظار لأحد الأوامر وهنا توضيح لبعض الأوامر الهامة :-

١. الأمر R وهو يوضح محتويات المسجلات . ولوضع قيمة محددة في أحد المسجلات يتم كتابة الأمر R متبوعاً باسم المسجل (مثلاً R IP).
٢. الأمر T (TRACE) وهو يؤدي إلي تنفيذ الخطوة الحالية فقط من البرنامج .
٣. الأمر G (GO) يؤدي إلي تنفيذ البرنامج .
٤. الأمر Q (QUIT) يؤدي إلي الخروج من البرنامج .
٥. الأمرA ASSEMBLE يتيح فرصة كتابة برنامج .
٦. الأمر U لرؤية جزء من الذاكرة .
٧. الأمر D DUMB يؤدي إلي إظهار جزء من الذاكرة .

```
MODEL   SMALL
        .STACK 100H
        .CODE
        MAIN  PROC
              MOV   AX , 4000H          ;ax = 4000h
              ADD   AX , AX             ;ax = 8000h
              SUB   AX , 0FFFFH         ;ax = 8001h
              NEG   AX                  ;ax = 7fffh
              INC   AX                  ;ax = 8000h
              MOV   AH , 4CH
              INT   21H                 ;DOS exit
        MAIN  ENDP
              END   MAIN
```

بعد كتابة البرنامج السابق وليكن اسمه test.asm وتوليد الملف القابل للتنفيذ Executable file والذي سيحمل الاسم Test.exe يتم نداء برنامج Debug وتحميل البرنامج وذلك بتنفيذ الأمر التالي من محث الـ DOS:

```
c:\asm> DEBUG TEST.EXE
```

يقوم البرنامج بالتحميل وإظهار المؤشر "-" والذي تشير للاستعداد لتلقي الأوامر.

نبدأ بتجربة الأمر R وذلك لإظهار محتويات المسجلات المختلفة وتكون المخرجات علي الصورة التالية:

```
- R
AX=0000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=EE6 IP=0000 NV UP DI PL NZ NA PO NC
0EE6:0000       B80040  MOV AX , 4000
```

يقوم البرنامج بإظهار محتويات المسجلات المختلفة وفي السطر الثالث يوضح عنوان الأمر التالي (المطلوب تنفيذه - لاحظ قيمة العنوان ومحتويات المسجلين CS:IP) متبوعاً بكود الآلة للأمر Machine Code وهو الرقم B80040 وبعد ذلك نجد الأمر مكتوباً بلغة التجميع.

عند تشغيل البرنامج ستجد أرقام مختلفة عن الأرقام الموضحة في هذا المثال وبالذات محتويات المسجلات المختلفة.

في نهاية السطر الثاني يوجد عدد ٨ أزواج حروف عل الصورة NV UP DI PL NZ NA PO NC توضح محتويات البيارق المختلفة وذلك حسب الجدول التالي:

البيرق	في حالة رفع البيرق Set	في حالة عدم رفع البيرق Clear
CF (CarryFlag)	CY (CarrY)	NC (No Carry)
PF (Parity Flag)	PE (Parity Even)	PO (Parity Odd)
AF (Auxiliary Flag)	AC (Auxiliary Carry)	NA (No Auxiliary carry)
ZF (Zero Flag)	ZR (ZeRo)	NZ (NonZero)
SF (Sign Flag)	NG (NeGative)	PL (Plus)
OF (Overflow Flag)	OV (OVerflow)	NV (No oVerflow)
بيارق التحكم Control Flags		
DF (Direction Flag)	DN (DowN)	UP (UP)
IF (Interrupt Flag)	EI (Enable Interrupt)	DI (Disable Interrupt)

لبداية تشغيل البرنامج نصدر الأمر T أي Trace للتنفيذ خطوة خطوة فيكون التسلسل التالي للأوامر :

في البداية كانت المسجلات علي النحو التالي (سنكرر الشاشة السابقة حتى نتابع التنفيذ بالتفصيل

```
- R
AX=0000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=EE6 IP=0000 NV UP DI PL NZ NA PO NC
0EE6:0000       B80040  MOV AX , 4000
```

ثم نبدأ التنفيذ: الأمر الأول MOV AX , 4000h

```
- T
AX=4000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=EE6 IP=0003 NV UP DI PL NZ NA PO NC
0EE6:0003       03C0    ADD AX , AX
```

التنفيذ يضع 4000h في المسجل AX

لاحظ أن المسجل AX أصبح به الرقم 4000H ولم يتم تغيير محتويات البيارق وأن الأمر التالي أصبح الأمر

ADD AX,AX

```
- T
AX=8000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=EE6 IP=0005 OV UP DI NG NZ NA PE NC
0EE6:0005       2DFFFF  SUB AX , FFFF
```

لاحظ أن المسجل AX أصبح به الرقم 8000H وأن النتيجة السابقة أثرت في البيارق حيث تم رفع بيرق الفيضان ليشير إلي حدوث فيضان بإشارة وبيرق الإشارة ليشير إلي أن النتيجة سالبة وكذلك بيرق التطابق لأن الخانة الأصغر من المسجل AX (أي AL) تحتوي علي عدد زوجي من الخانات التي بها الرقم ١ . والآن نتابع تنفيذ البرنامج حيث الأمر التالي هو الأمر SUB AX,FFFFh

```
- T
AX=8001 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=EE6 IP=0008 NV UP DI NG NZ AC PO CY
0EE6:0008       F7D8    NEG AX
```

```
- T
AX=7FFF BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=EE6 IP=000A NV UP DI PL NZ AC PE CY
0EE6:000A       40      INC AX
```

```
-T
AX=8000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=EE6 IP=000B OV UP DI NG NZ AC PE CY
0EE6:000B       B44C    MOV AH , 4C
```

```
- G
PROGRAM TERMINATED NORMALLY
-Q
C:\>
```

التكرارات

التكرار هو عملية تنفيذ مجموعة من الأوامر لأكثر من مرة وقد يكون التكرار لعدد محدد من المرات أو قد يكون التكرار حتى حدوث حدث محدد.

١- For Loop

```
For (index=star t; index condition stop ;index++)
Do Statement
End /For
```

بلغة التجميع :

```
Mov index,start

FORR: cmp index,stop
JCondition DO
Jmp ENDFORR
DO:
Statement
Inc index
Jmp FORR
ENDFORR
```

مثال :اكتب برنامج يقوم بطباعة الاعداد من 0 الى 5 ؟

```
.model small
.stack 100h
.data
x db ?

.code
main proc
mov Ax,@DATA
```

```
mov DS,AX


mov bl,0
mov x,30h


FORR:cmp bl,5
Jle DO
jmp endForr


DO:
add x,bl


mov ah,2h
mov dl,x
int 21h


sub x,bl
inc bl
jmp FORR
endforr:
mov ah,4ch
int 21h


main endp
end main
```

مثال :- اكتب برنامج يستخدم حلقة التكرار FOR وذلك لطباعة 20 نجمة"* "

```
.model small
.stack 100h
.code
main proc
mov CX,20

mov ah,02h
mov dl,'*'

FORR:
int 21h
JCXZ ENDFORR
Dec CX
jmp FORR

ENDFORR:
mov ah,04ch
int 21h
main endp
end main
```

المثال السابق يوضح إمكانية استخدام المسجل CX كعداد وكذلك إمكانية عدم استخدام إحدى العوامل كعداد.

والتعليمة JCXZ تختبر المسجل CX يقفز إذا كان يساوي الصفر.

٢- حلقة WHILE

يتم تكرار هذه الحلقة حتى حدوث شرط محدد حيث الشكل العام لها علي النحو التالي

```
While Condition DO
        Statements
End_while
```

يتم اختبار الشرط في بداية الحلقة فإذا تحقق الشرط يتم تنفيذ جسم الحلقة وإذا لم يتحقق يتم الخروج من الحلقة وتنفيذ الأوامر التالية في البرنامج.

لاحظ أن الشرط قد لا يتحقق من البداية وبالتالي لا يتم الدخول أصلاً في جسم الحلقة مما يؤدي إلي إمكانية عدم تنفيذ جسم الحلقة علي الإطلاق. لاحظ أيضاً أن جسم الحلقة يقوم دائماً بتغيير أحد معاملات شرط الحلقة حتى يتحقق شرط إنهاء الحلقة (في حالة عدم تغيير معاملات الشرط تكون الحلقة لانهائية)

```
WHILEE: cmp opr1,opr2
JCondition DO
Jmp ENDWHILE
DO:
Statement
Inc/dec opr1
Jmp WHILEE
        ENDWHILE
```

اكتب برنامج يقوم بطباعة الاعداد من 0 الى 4 ؟

```
        .model small
        .stack 100h
        .data
        x db ?
```

```
.code
main proc
mov Ax,@DATA
mov DS,AX
mov bl,0
mov x,30h

WHILEE:cmp bl,5
Jl DO
jmp ENDWHILE

DO:
add x,bl
mov ah,2h
mov dl,x
int 21h
sub x,bl
inc bl
jmp WHILEE
ENDWHILE:
mov ah,4ch
int 21h
main endp
end main
```

3- حلقة Do While Lop

وهى حلقة أخري تقوم بالتكرار حتى حدوث شرط محدد.والشكل العام لها يكون على الصورة

```
DO
        STATEMENT
While   CONDITION
```

وهنا يتم تنفيذ جسم الحلقة ثم بعد ذلك يتم اختبار الشرط .فإذا تحقق الشرط يتم الخروج من الحلقة أما إذا لم يتحقق يتم تكرار الحلقة .

```
DO:
        STATEMENT
Inc/dec opr
WHILEE: cmp opr,opr2
JCondition DO
ENDWHILEE:
```

اكتب برنامج يقوم بطباعة الاعداد من 0 الى 4 ؟

```
.model small
.stack 100h
.data
x db ?
.code
main proc
mov Ax,@DATA
mov DS,AX
mov bl,0
```

```
mov x,30h


DO:
add x,bl
mov ah,2h
mov dl,x
int 21h


sub x,bl
inc bl
WHILEE:cmp bl,5
Jl DO
ENDWHILE:
mov ah,4ch
int 21h


main endp
end main
```

المكدس

يتم استخدام مقطع المكدس للتخزين المؤقت للعناوين والبيانات أثناء عمل البرنامج

يعتبر المكدس كمصفوف أحادي في الذاكرة ويتم التعامل مع طرف واحد فقط منه حيث يتم إضافة العنصر في قمة المكدس ويتم أخذ آخر عنصر في عملية السحب التالية بمعني انه يعمل بطريقة آخر مدخل هو أول مخرج LIFO (Last In first out) يجب على كل برنامج أن يقوم بتحديد منطقة في الذاكرة وتعمل كمكدس كما ذكرنا في الفصول السابقة وذلك باستخدام الأمر.

```
STACK  100h
```

حيث يشير مسجل مقطع المكدس SS إلى مقطع المكدس في المثال السابق ويحتوى مؤشر المكدس SP على القيمة 100h وهى تشير إلى مكدس خالي وعند وضع قيم فيه يتم إنقاص هذه القيمة.

وضع قيم في المكدس الامر , PUSH:

يتم استخدام الأمر PUSH لإدخال قيمة في المكدس وصيغته

```
PUSH        SOURCE
```

حيث المصدر هو مسجل أو موقع في الذاكرة بطول 16 خانة . مثلاً

```
PUSH        AX
```

ويتم في هذه العملية الآتي:

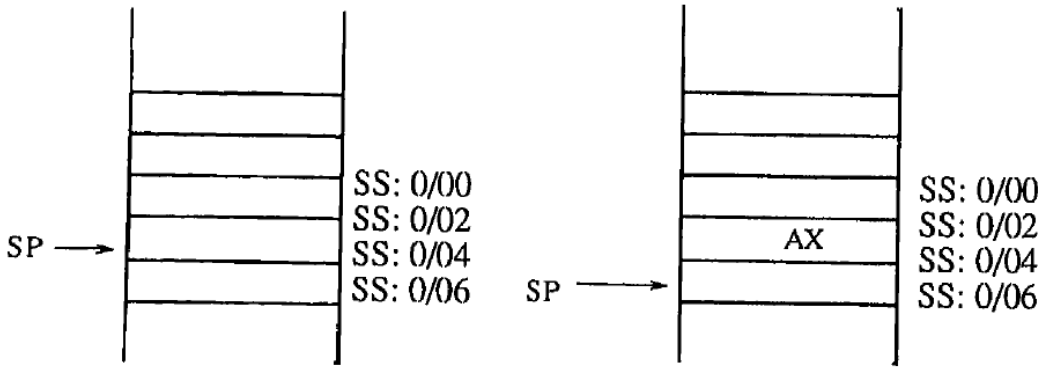
1- إنقاص قيمة مؤشر المكدس SP بقيمة 2

22- يتم وضع نسخة من المصدر في الذاكرة في العنوان SS:SP

لاحظ أن محتويات المصدر لا يتم تغييرها.

مثال :

PUSH AX



سحب قيمة من المكدس الامر , POP:

لسحب قيمة من المكدس يتم استخدام الأمر POP وصيغته

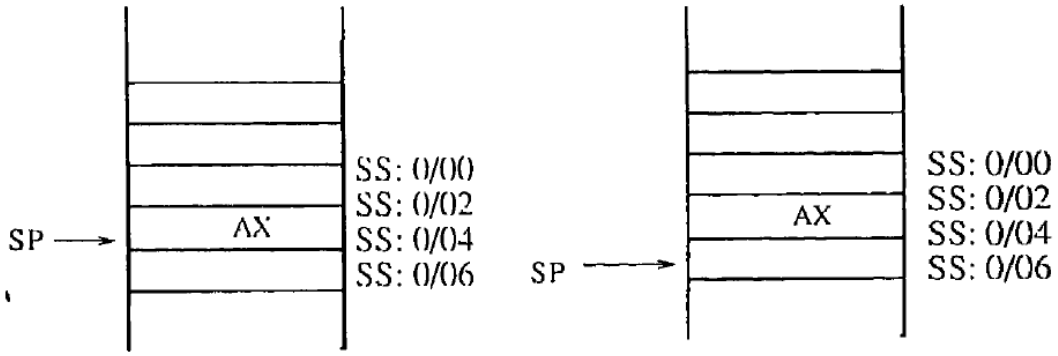
POP Destination

حيث المستودع عبارة عن مسجل 16 خانة (ماعدا المسجل IP) أو خانة في الذاكرة مثلاً

POP AX وتنفيذ الأمر POP يتضمن التالي:

1- نسخ محتويات الذاكرة من العنوان SS:SP الى المستودع

2- زيادة قيمة مؤشر المكدس SP بالقيمة 2



ل أنها تتعامل مع متغيرات بطول 16 خانة ولا تتعامل مع 8 خانات. فمثلاً الأمر التالي غير صحيح

Push AL ; ILLEGAL

مثال يقوم بقراءة جملة من لوحة المفاتيح. يقوم البرنامج في السطر التالي بطباعة الجملة بصورة عكسية مثال للتنفيذ:

? this is a test

tset a si siht

يستخدم البرنامج المسجل CX للاحتفاظ بعدد حروف الجملة التي تم إدخالها بعد الخروج من حلقة while يكون عدد الحروف الموجودة في المسجل CX وتكون كل الحروف التي تم إدخالها موجودة في المكدس بعد ذلك يتأكد البرنامج من انه قد تم إدخال حروف وذلك بالتأكد من أن المسجل CX لا يساوى صفر.

```
.MODEL      SMALL
.STACK      100H
.CODE
MAIN        PROC
            ; display user prompt
      MOV   AH,2
      MOV   DL,'?'
      INT   21H
            XOR   CX , CX
      ;read character
      MOV   AH , 1
      INT   21H
 WHILE_:
      CMP   AL , 0DH
      JE    END_WHILE
      PUSH  AX
      INC  CX
      INT   21H
      JMP   WHILE_
END_WHILE:
      MOV   AH , 2
      MOV   DL , 0DH
      INT   21H
      MOV   DL , 0AH
      INT   21H
      JCXZ  EXIT
TOP:
      POP   DX
      INT   21H
      LOOP  TOP
EXIT: MOV   AH , 4CH
      INT   21H
MAIN        ENDP
END   MAIN
```

المصفوفات ذات البعد الواحد One - Dimensional Arrays

المصفوف هو عبارة عن مجموعة من العناصر مرتبة وراء بعضها في الذاكرة وقد تكون هذه العناصر عبارة عن حروف Bytes أو جمل Words أو أي نوع آخر. فإذا كان اسم المصفوف هو A فأن عناصر المصفوف هي A[1] و A[2] و A[3] و ....A[N] حيث N هو عدد عناصر المصفوف وقد تعرفنا سابقاً على كيفية تعريف المصفوف فمثلاً لتعريف مصفوف من الحروف اسمه Msg نستخدم التعريف

```
MSG     DB  "ABCDE"
```

حيث يتم يكون MSG[1] = A و MSG[2] = (B) وهكذا .

ولتعريف مصفوف من الكلمات (كل عنصر يشغل خانتين في الذاكرة) باسم A نستخدم التعريف التالي :

```
A     DW   10,20,30,40,50,60
```

حيث يتضمن ذلك تعريف مصفوف به 5 خانات كل خانه عبارة عن كلمه Word بقيم ابتدائية A[1] = 10 و A[2] = 20 و A[3] = 30 و A[4] = 40 و A[5] = 50

يسمى عنوان المصفوف بالعنوان الأساسي للمصفوف Base Address of the array ويتم تحديد هذا العنوان عند تحميل البرنامج إلى الذاكرة فمثلاً إذا كان عنوان الإزاحة للمصفوف A هو العنوان 0200h يكون شكل المصفوف على النحو التالي:

العنوان الرمزي	قيمة الإزاحة	المحتويات في النظام العشري
A	0200h	10
A + 2h	0202h	20
A + 4h	0204h	30
A + 6h	0206h	40
A + 8h	0208h	50

المؤثر DUP (Duplicate)

يستخدم المؤثر Dup لتعريف مصفوف بعدد من العناصر تأخذ كلها نفس القيمة الابتدائية ويكون على الصورة.

Repeat_Count Dup (value)

يقوم المؤثر Dup بتكرار القيمة value عدد من المرات يساوى Repeat_count مثلاً:

GAMMA DW 100 Dup (0)

هنا يتم تعريف مصفوف باسم GAMMA يحتوى على 100 عنصر كل عنصر عبارة عن Word ووضع قيمة ابتدائية 0 في كل العناصر وكمثال آخر.

DELTA DB 60 Dup (?)

حيث يتم تعريف مصفوف باسم Delta يتكون من 60 عنصر حرفي Byte وعدم وضع أي قيمة ابتدائية للعناصر

مواقع عناصر المصفوفة

يبدأ تخزين المصفوف في الذاكرة ابتدأ من العنوان الأساسي للمصفوف وهو عنوان العنصر الأول ويكون عنوان العنصر الثاني يعتمد على نوعية عناصر المصفوف فإذا كانت Byte يكون هو الأساسي + 1 أما إذا كانت Word يكون عنوان العنصر الثاني هو العنوان الأساسي + 2 وهكذا وعموماً إذا كانت S هي طول عنصر المصفوف(S 1= إذا كانت العناصر عبارة عن Byte و2 = S إذا كانت العناصر عبارة عن Word) يكون عنوان العنصر N هو العنوان الأساسي للمصفوف + S * (N - 1) فمثلاً المصفوف A المعرف سابقاً يكون فيه عنوان العنصر N هو S (N - 1) + A

مثال: استبدل العنصرين رقم 10 ورقم 25 في المصفوف W حيث

W DW 100 Dup (?)

الحل

عنوان العنصر العاشر هو W + 18 = W + 9 x 2 = W + (10 - 1) * 2

وعنوان العنصر 25 هو W + 48 = W + 24 x 2 = W + (25 - 1) * 2

وبالتالي يكون البرنامج هو

```
MOV     AX , W + 18
XCHC    A x , W + 48
MOV     W + 18 , A x
```

في كثير من التطبيقات نحتاج للتعامل مع عناصر المصفوف كلها. مثلاً إذا أردنا إيجاد مجموع عناصر المصفوف A والذي به عدد N عنصر فإننا نحتاج لمخاطبة العناصر داخل حلقة كما في الخوارزمية التالية:

```
Sum = 0
M = 0
Repeat
    Sum = sum + A [M]
    M = M + 1
Until   M = N
```

ولعمل ذلك نحتاج لطريقة للتحرك بين عناصر المصفوف وذلك باستخدام مؤشر محدد وتغيير قيمته كل مره داخل الحلقة ولذلك سنقوم في الجزء التالي بتوضيح طرق العنونة المختلفة المستخدمة.

أنماط العنونة ADDRESSING MODES

طريقة استخدام معاملات الأمر تسمى بطرق العنونة وقد تعاملنا سابقاً مع ثلاثة أنماط مختلفة للعنونة وهى:

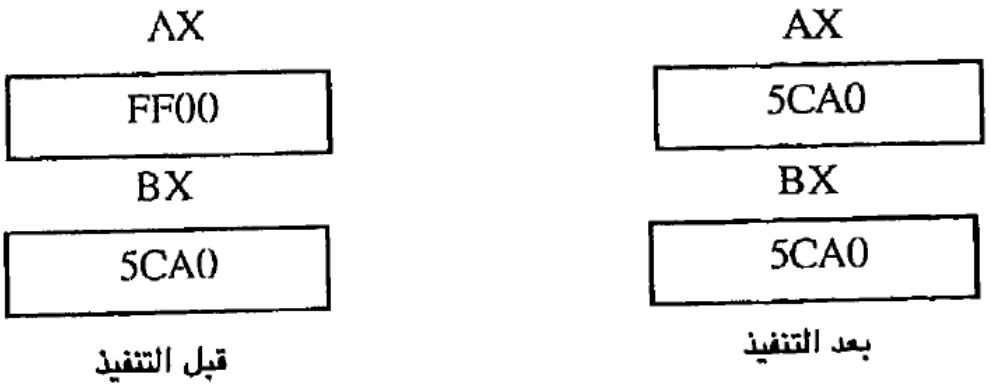
1/ نمط المسجلات Register Mode

وفيه يتم استخدام أحد المسجلات المعروفة

باستخدام هذه الطريقة يقوم المعالج بالبحث عن المعامل في أحد مسجلاته وليس هناك حاجة للرجوع للذاكرة الرئيسية .

مثال : MOV AX,BX

تقوم هذه التعليمة بنقل محتويات المسجل BX الي المسجل AX .



٢

2/ النمط اللحظى Immediate Mode

وفيه يتم استخدام الثوابت بمعاملات مثل

MOV A x , 5

هنا المعامل Ax يعتبر عنونه من النوع Register والمعامل 5 يعتبر من النمط اللحظي Immediate

3/ النمط المباشر Direct Mode

تحتوي التعليمة على العنوان الفعلى للمعامل المطلوب اجراء العمليات علية ،ويعطى العنوان في التعليمة على شكل اسم لموقع بذاكرة الذي يحتوي على المعامل .

حينما يكون المعامل أحد المتغيرات مثل MOV Ax , Words

حيث المعامل Words عبارة عن مجموعة مباشرة

هناك أربعة أنماط أخرى سنقوم بالتحدث عنها في الأجزاء التالية:

4/ نمط العنونة بالاستخدام الغير مباشر للمسجلات Register Indirect Mode.

يتم هنا تحديد عنوان الذاكرة المطلوب في أحد المسجلات SI أو BX أو DI أو BP وعلى هذا يعتبر المسجل أنه مؤشر Pointer للعنوان المطلوب مخاطبته ويتم وضع المعامل داخل الأمر على الصورة التالية:

[Register]

المسجلات DI , SI , BX تشير إلي العناوين داخل مقطع البيانات DS والمسجل BP يشير إلي العناوين داخل مقطع المكدس SS.

مثال:

إذا كان SI = 0100h والكلمة في العنوان 0100h في البيانات تحتوى على الرقم 1234h فإن الأمر

MOV AX , [SI]

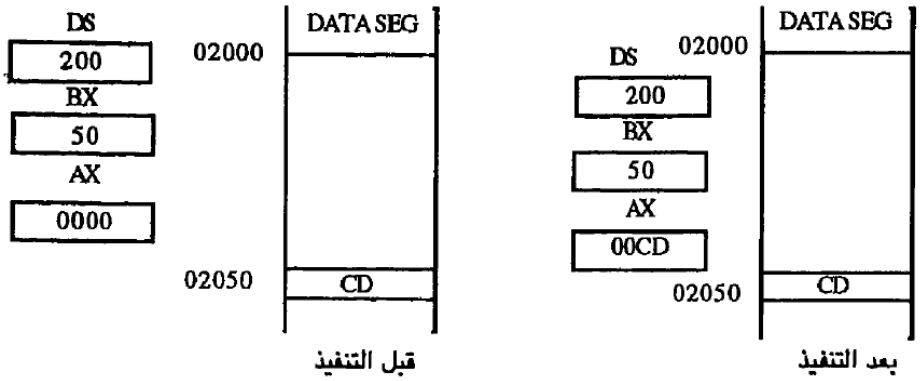
يتم أخذ القيمة 100h من المسجل SI وتحديد العنوان DS: 0100 ثم أخذ القيمة الموجودة في ذلك العنوان (الرقم 1234h) ووضعها في المسجل AX (أي AX = 1234h) وهذا بالطبع غير الأمر

MOV AX , SI

مثال :

MOV AL, [BX]

تقوم هذه التعليمة بنقل محتويات الموقع المشار له بـ BX الى AL ، ويبين الشكل تأثير هذه التعليمة على الذاكرة والمسجل AL ، BX ، بعد التنفيذ



والذي يقوم بوضع الرقم 0100h في المسجل AX

مثال:

افترض أن DI = 3000h , SI = 2000h , BX = 1000h وأن الذاكرة تحوى القيم التالية في مقطع البيانات في الازاحه 1000h يوجد الرقم 1BACH وفي

الازاحـه 2000h يوجـد الـرقم 20FEh وفـي الإزاحـة 3000h يوجـد الـرقم 031Dh حيث أن الازاحات أعلاه في مقطع البيانات Data Segment . حدد أياً من الأوامر أدناه صحيحاً. ووضع العدد الذي يتم نقله في هذه الحالة:

أ - MOV BX, [BX] ب - MOV CX, [SI]

جـ MOV BX, [AX] د - ADD [SI] [DI] هـ - INC [DI]

الحل:

أ - MOV BX, [BX] يتم وضع الرقم 1BACh في المسجل BX

ب - MOV CX, [SI] يتم وضع الرقم 20FEh في المسجل CX

جـ MOV BX, [AX]– خطأ لا يمكن استخدام المسجل AX في العنونة الغير مباشرة.

د - ADD [SI], [DI] خطأ لا يمكن جمع محتويات عنصرين في الذاكرة بأمر واحد

هـ/ INC [DI] يتم جمع الرقم واحد إلى محتويات الذاكرة في الازاحه 3000h لتصبح القيمة 031Eh الموجودة

مثال: أكتب جزء من برنامج يقوم بطباعة العناصر الخمسة للمصفوف W إذا كان

w db 30h,31h,32h,33h,34h,35h

الحل:

```
.MODEL          SMALL
.STACK  100H
.DATA
w db  30h,31h,32h,33h,34h,35h

.CODE
MAIN    PROC
mov ax,@data
mov ds,ax
  LEA SI,w
        mov bl,0

WHILE_:
        CMP    bl , 5
```

```
        JE      END_WHILE
    mov al,[SI]
        add SI,1
      INC  bl
       mov ah,2h
          mov dl,al
        INT     21H
        JMP     WHILE_
END_WHILE:
        MOV   AH , 4CH
        INT     21H
MAIN    ENDP
END     MAIN
```

5/ أنماط العنونه المنفهرسة والأساسية Indexed and Based Addressing modes

في هذه الأنماط يتم إضافة عدد يسمى بالازاحة Displacement لمحتويات المسجل وقد تكون الازاحه أحد القيم التالية حيث A عبارة عن متغير تم تعريفه.

- قيمة الازاحه لمتغير مثل A
- قيمة ثابتة مثل 2
- قيمة الازاحه لمتغير بالاضافه الى قيمة ثابتة باشارة مثل A + 2

ويأخذ هذا النمط إحدى الصور التالية :

[Register +Displacement]

[Displacement + Register]

[Register] +Displacement

Displacement + [Register]

Displacement [Register]

المسجل يجب أن يكون أحد المسجلات BX و BP و SI و DIإذا تم استخدام أحد المسجلات BX أو SI أو DI فإن المسجل DS يشير إلي المقطع المعني أما إذا تم استخدام المسجل BP فإن المسجل SS يشير إلي المقطع المعني.

إذا تم استخدام المسجل BX أو المسجل BP يسمي النمط بـ Based بينما يسمي النمط بـ Indexed إذا تم استخدام المسجل SI أو المسجل DI.

كمثال لهذا النمط إذا كان المتغير W عبارة عن مصفوف من الجمل Word Array وأن المسجل BX بـه الرقم ٤ فإن الأمر التالي يقوم بوضع العنصر الموجود في الذاكرة بالعنوان W + 4 في المسجل AX

```
MOV  AX , W [ BX]
```

وهذا هو العنصر الثالث في المصفوف، ويمكن كتابة الأمر بأحد الصور التالية والتي تؤدي نفس الغرض:

```
MOV  AX , [ W + BX ]
MOV  AX , [ BX + W ]
MOV  AX , W +[ BX ]
MOV  AX , [ BX ] + w
```

كمثال آخر افترض أن المسجل SI يحتوي علي عنوان بداية مصفوف W من الجمل Word Array. أي من الأوامر التالية يقوم بوضع محتويات العنصر الثاني والموجود بالعنوان W + 2 في المسجل AX :

```
MOV  AX , [ SI + 2 ]
MOV  AX , [ 2 + SI ]
MOV  AX , 2 + [ SI ]
MOV  AX , [ SI ] + 2
MOV  AX , 2 [ SI ]
```

مثال

أكتب (مستعملاً نم العنونة الأساسية) جزء من برنامج يقوم بجمع عناصر المصفوف W في المسجل AX إذا كان: W DW

10,20,30,40,50,60,70,80,90,100

الحل:

```
        XOR   AX , AX
        XOR   BX , BX
        MOV   CX , 10
ADDNOS:
        ADD   AX , w [ BX ]
        ADD   BX , 2
        LOOP ADDNOS
```

يـتم إضـافة الـرقم ٢ للمسـجل SI للتحـرك للعنصـر التـالي حيـث أن المصفوف به كلمات Words

مثال:

```
.MODEL   SMALL
.STACK    100H
.DATA
w db  30h,31h,32h,33h,34h,35h

.CODE
MAIN      PROC
mov ax,@data
mov ds,ax

          mov bl,0
  xor SI,SI
WHILE:_
          CMP    bl , 5
          JE     END_WHILE
    mov al,w[SI]
          add SI,1
       INC  bl
        mov ah,2h
          mov dl,al
          INT    21H
          JMP    WHILE_
END_WHILE:
          MOV    AH , 4CH
          INT    21H
MAIN      ENDP
END       MAIN
```

مثال

افترض أن المتغير Alpha معرف علي النحو التالي :

ALPHA DW 0123h, 0456h, 0789h, 0abcdh

وأن المسجلات بها القيم التالية : BX = 2 DI = 1 SI =4, وأن الذاكرة بها الرقم 1084h في الإزاحة ٠٠٠٢ وبها الرقم 2BACh في الإزاحة ٠٠٠٤.

وضح أياً من الأوامر التالية صحيح وإذا كان الأمر صحيح وضح عنوان الإزاحة للمصدر والرقم الذي تم التعامل معه في كل من الحالات التالية:

a. MOV AX , [ALPHA + BX]

b. MOV BX , [BX+ 2]

c. MOV CX , ALPHA [SI]

d. MOV AX , -2 [SI]

e. MOV BX , [ALPHA + 3 + DI]

f. MOV AX , [BX] 2

g. ADD BX , [ALPHA + AX]

الحل:

السؤال	عنوان الإزاحة	القيمة التي تم وضعها في المسجل
A	APLPHA +2	0456h
B	2 + 2 = 4	2BACh
C	ALPHA + 4	0789h
D	-2 + 4 = 2	1084h
E	ALPHA + 3 + 1	0789h
F	المصدر مكتوب بطريقة غير صحيحة	
G	لا يمكن استخدام المسجل AX هنا	

المعامل PTR والإيعاز LABEL:

ذكرنا فيما سبق أن المعاملين للأمر يجب أن يكونا من نفس النوع فمثلاً يكون المعاملان من النوع الحرفي Byte أو من النوع WORD . وإذا كان المعامل عبارة عن رقم ثابت يقوم المجمع بتفسيره حسب نوع المعامل الثاني فمثلاً يتم التعامل مع الرقم الثابت في المثال التالي على أنه عبارة عن متغير من النوع WORD .

```
MOV  AX , 1
```

بينما يتم التعامل مع الثابت التالي على انه متغير حرفي Byte

```
MOV  AL , 1
```

ولكن لا يمكن التعامل مع الأمر التالي

```
MOV  [ BX ] ,  1
```

وذلك لأن المستودع غير معرف هل هو word أم Byte .

ليتم تخزين الثابت على أنه من النوع Byte نستخدم الأمر

```
MOV   BYTE   PTR  [ BX ] , 1
```

وليتم تخزين الثابت على أنه من النوع WORD نستخدم الأمر

```
MOV   WORD   PTR  [ BX ] , 1
```

مثال: استبدل الحرف الأول في متغير يسمي MSG بالحرف "T"

الحل:

<u>الطريقة الأولى :</u>

باستخدام طريقة العنونة الغير مباشرة باستخدام المسجلات Register indirect mode

```
LEA   SI, Msg
MOV  BYTE PTR [SI], 'T'
```

```
.model small
.stack 100h
.data
msg db "kelphone$"
.code
main proc
mov ax,@data
mov ds,ax
lea SI,msg

mov BYTE ptr[SI],'T'
lea dx,msg
mov ah,09h
int 21h
mov ah,04ch
int 21h
main endp
end main
```

الطريقة الثانية: باستخدام العنونة المفهرسة Index Mod

```
XOR   SI, SI

MOV   mSG[ SI ] , 'T'
```

غير ضروري هنا استخدام المعامل PTR حيث أن Msg عبارة عن متغير حرفي

```
.model small
.stack 100h
.data
msg db "kelphone$
.code
main proc
mov ax,@data
mov ds,ax

xor si,si

mov msg[SI],'T'

lea dx,msg
mov ah,09h
int 21h
mov ah,04ch
int 21h
main endp
end main
```

استخدام PTR لإعادة تعريف متغير:

يمكن استخدام PTR لإعادة تعريف متغير تم تعريفه من قبل والصيغة العامة هي:

```
Type  PTR  Address_Expression
```

حيث Type هي Byte أو WORD أو Dword و Address_Expression هى DB أو DW أو DD

فمثلاً إذا كان لدينا التعريف التالي:

```
DOLLARS     DB    1Ah
CENTS       DB    52h
```

إذا أردنا وضع محتويات المتغير Dollars في المسجل AL والمتغير Cents في المسجل AH باستخدام أمر واحد لن نستطيع ذلك

```
MOV  CX , DOLLARS  ;  ILLEGAL
```

حيث أن المصدر عبارة عن Byte بينما المستودع عبارة Word ولكن يمكن إعادة كتابة الأمر على الصورة التالية

```
MOV  CX ,word  PTR DOLLARS  ; CL=DOLLARS , CH =Cents
```

وسيتم وضع الرقم 521Ah في المسجل CX

```
.model small
.stack 100h
.data
DOLLARS   DB   33h
CENTS     DB   ٣٢h
.code
main proc
mov ax,@data
```

```
mov ds,ax

MOV   CX ,word  PTR DOLLARS

mov ah,02h
mov dl, cl
int 21h
mov ah,02h
mov dl, ch
int 21h

mov ah,04ch
int 21h
main endp
end main
```

المعامل LABEL:

يمكن حل مشكلة اختلاف الأنواع هذه باستخدام المعامل LABEL فمثلاً يمكن استخدام الإعلان التالي:

MONEY	LABEL	WORD
DOLLARS	DB	1Ah
CENTS	DB	52h

وبالتالي يستخدم المتغير MONEY على انه من النوع Word والمتغيرين DOLLARS و CENTS عبارة عن متغيرات من النوع Byte . وبالتالي يصبح الأمر التالي صحيحاً

```
MOV          Ax , Money
```

وله نفس تأثير الأمرين

```
MOV  AL , DOLLARS

MOV  AH , CENTS
```

مثال: اعتبر الإعلانات التالية:

```
.DATA

      A      DW          1234H
```

```
B       LABEL           BYTE
        DW              4142h
C       LABEL           WORD
C1      DB              43H
C2      DB              44h
```

تكون الأوامر على النحو التالي:

الرقم	الأمر	ملحوظة	البيانات المنقولة
1	MOV AX , B	غير صحيح	تضارب الأنواع
2	MOV CH , B	صحيح	42h
3	MOV CX , C	صحيح	4443h
4	MOV BX , WORD PTR B	صحيح	4142h
5	MOV DL , BYTE PTR C	صحيح	43h
6	MOV CX , WORD PTR C1	صحيح	4443h

```
.model small
.stack 100h
.data
A       DW          1234H
        B       LABEL       BYTE
                DW          4142h
        C       LABEL       WORD
        C1      DB          43H
        C2      DB          44h
.code
main proc
mov ax,@data
mov ds,ax
;MOV AX , B
MOV ch , B
mov ah,02h
mov dl, ch
int 21h
mov dl, 0ah
int 21h

mov dl, 0dh
int 21h
MOV CX , C
mov ah,02h
mov dl, cl
```

```
int 21h
mov dl,ch
int 21h
mov dl, 0ah
int 21h
mov dl, 0dh
int 21h

MOV BX , WORD PTR B
mov ah,02h
mov dl, bl
int 21h
mov dl,bh
int 21h
mov dl, 0ah
int 21h
mov dl, 0dh
int 21h

MOV DL , BYTE PTR C
mov ah,02h
int 21h
mov dl, 0ah
int 21h

mov dl, 0dh
```

```
int 21h


MOV CX , WORD PTR C1
mov ah,02h
mov dl, cl
int 21h
mov dl,ch
int 21h
mov dl, 0ah
int 21h
mov dl, 0dh
int 21h
mov ah,04ch
int 21h
main endp
end main
```

عمليات القيمة الراجعة :

تقوم هذه العمليات بإعطاء معلومات عن المتغيرات و الأسماء الأخرى المستخدمة في البرنامج

١- عملية OFFSET

هذه العملية بحساب العنوان الفعلي للمتغير الذي يأتي بعدها

مثال :

OFFSET X العنوان الفعلي لـــــX

OFFSET X+4 العنوان الفعلي لـــــX +4

اكتب برنامج يقوم بطباعة قيمة X بتحميل المسجل DX بالعنوان الفعلي لـــــX؟

```
.model small
.stack 100h
.data
x db "hello"$

.code
main proc
mov ax,@data
mov ds,ax

mov DX,offset x

mov ah,09h
int 21h

mov ah,04ch
int 21h
main endp
end main
```

٢- عملية SEG

وظيفة هذه العملية هي إيجاد عنوان القطاع الذي يقع فيه المتغير الذي يأتي بعد كلمة SEG

مثال : SEG X عنوان القطاع للمتغير X

Mov DX,SEG X

تحميل المسجل DX بعنوان القطاع الذي يقع فيه المتغير X

٣- عملية LENGTH

تقوم بإيجاد عدد المدخلات المعرفة باستخدام المعامل DUP

```
X DW 10 DUP(?)
Mov  DX,LENGTH x
```

التعليمة mov تقوم بنقل 10 الى المسجل DX والرقم عشرة هو عدد المدخلات المعرفة بواسطة DUP

٤- عملية TYPE

تقوم بإيجاد عدد البايتات المخصصة لتعريف موقع معين :

```
FLD DB ?
TAB DW 20 DUP(?)
MOV AX ,TYPE FLD;AX=1
```

عدد البايتات المخصصة لتعريف المتغير FLD هو واحد ولهذا يتم نحويل رقم ١ الي المسجل AX

```
Mov AX,TYPE TAB
```

فتقوم نتحريك ٢ الى المسجل AX لان TAB متغير معرف ككلمة تتكون من بايت

٥- عملية SIZE

وتحدد هذه العملية عدد البايتات المخصصة لمتغير معين ويتم ذلك بضرب LENGTH في TYPE

```
SIZE X
MOV AX ,SIZE X;AX=20
```

مثال:

```
.model small
.stack 100h
.data
x db؟

TAB DW 10 dup(?)
.code
main proc
mov ax,@data
mov ds,ax

mov dl,TYPE  x
add dl,30h
mov ah,02h
int 21h

mov dl,TYPE  tab
add dl,30h
mov ah,02h
int 21h

mov ah,04ch
int 21h
main endp
end main
```

الأوامر المنطقية

الأوامر المنطقية AND,OR,XOR

تستخدم الأوامر المنطقية في التعامل مع خانة ثنائية واحدة في المسجل المحدد والشكل العام للأوامر هو:

AND DESTINATION , SOURCE

OR DESTINATION , SOURCE

XOR DESTINATION , SOURCE

وتم تخزين النتيجة في المستودع DESTINATION الذي يجب أن يكون مسجل أو موقع في الذاكرة بينما المعامل الآخر SOURCE يمكن أن يكون مسجل أو موقع في الذاكرة أو قيمة ثابتة. عموماً لا يمكن التعامل مع موقعين في الذاكرة.

يكون تأثر البيارق على النحو التالي :

PF,ZF,ZF : تعكس حالة النتيجة.

AF : غير معرفة.

CF,OF : تساوي صفر .

أحد الاستخدامات المهمة للأوامر المنطقية هو تغيير خانة محددة داخل مسجل ويتم ذلك باستخدام حجاب MASK حيث يتم بواسطته تحديد الخانة المطلوب التعامل معها ويتم الاستعانة بالخصائص التالية للأوامر المنطقية :

b AND 1 = b , b AND 0 = 0

b OR 1 = 1 , b OR 0 = b

b XOR 1 =~ b , b XOR 0 = b

وعلى هذا يمكن الآتي :

١- لوضع القيمة '0' في خانة (أو خانات) محددة Clear يتم استخدام الأمر ANDحيث يتم وضع القيمة '0' في الحجاب MASK للخانات المطلوب وضع '0' فيها بينما يتم وضع القيمة '1' في الخانات الغير مطلوب تعديلها .

٢- لوضع القيمة '1' في خانة (أو خانات) محددة SET يتم استخدام الأمر OR حيث يتم وضع القيمة '1' في الحجاب MASK للخانات المطلوب وضع '1' فيها بينما يتم وضع القيمة '0' في الخانات الغير مطلوب تعديلها.

٣-لعكس قيمة خانة (أو خانات) محددة COMPLEMENT يتم استخدام الأمر XOR حيث يتم وضع القيمة '1' في الحجاب MASK للخانات المطلوب عكس قيمتها بينما يتم وضع القيمة '0' في الخانات الغير مطلوب تعديلها .

مثال:

ضع القيمة '0' في خانة الإشارة في المسجل AL واترك باقي الخانات بدون تعديل.

الحل

يتم استخدام القيمة 0111 1111b = 7Fh كحجاب MASK ويتم استخدام الأمر AND

AND AL, 7Fh

مثال

ضع القيمة '1' Set في كل من الخانة ذات الوزن الأكبر MSB والخانة ذات الوزن الأصغر LSB في المسجل AL وأترك باقي الخانات بدون تعديل

الحل

يتم استعمال الحجاب Mask = 1000 0001b = 81h ونستخدم الأمر OR كالتالي

OR AL , 81h

مثال

غير إشارة المسجل DX

الحل

يتم استخدام الحجاب Mask التالي 1000 0000 0000 0000b = 8000h ونستخدم الأمر XOR

XOR DX , 8000h

وعموماً يتم استخدام الأوامر المنطقية في مجموعة من التطبيقات والتي سنتحدث عن بعضها في الجزء التالي

تحويل الحروف الصغيرة لحروف كبيرة

نعلم أن الحروف الصغيرة ('a' to 'z') تقع في جدول الـ ASCII ابتداء من الرقم 61h وحتى 7Ah بينما تقع الحروف الكبيرة ('A' to 'Z') في جدول الـ ASCII ابتداء من الرقم 41h وحتى 5Ah وعلي ذلك فإنه لتحويل الحرف من صغير إلي كبير نطرح الرقم 20h فمثلاً إذا كان المسجل DL يحتوي علي حرف صغير ومطلوب تحويله إلي حرف كبير نستعمل الأمر SUB DL , 20h وقد قمنا باستخدام هذه الطريقة من قبل. ونريد هنا استخدام طريقة أخري للتحويل.

إذا نظرنا للأرقام المناظرة للحروف نجد أن

الرقم المناظر للحرف 'a' هو 61h = 0110 0001

الرقم المناظر للحرف 'A' هو 41h = 0100 0001

ومن الأرقام نلاحظ تحويل الحرف من صغير إلي كبير يتطلب وضع القيمة '0' في الخانة السادسة في المسجل الذي يحوي الحرف ويتم ذلك باستخدام الحجاب Mask التالي 0DFh =1101 1111b ونستعمل الأمر AND

```
AND    DL , 0DFh
```

مثال :اكتب برنامج يقوم بتحويل الحرف المدخل من اصمول إلى جرف كبتل باستخدام الأوامر المنطقية؟

```
model small
.stack 100h
.code
main proc

mov ah,01h
int 21h

AND al,0dfh

mov ah,02h
mov dl,al
int 21h

mov ah,04ch
int 21h
main endp
end main
```

تفريغ مسجل (وضع صفر فيه) Clear Register

نعلم أنه لوضع القيمة صفر في مسجل يمكننا استخدام أحد الأمرين MOV AX,0

أو SUB AX , AX إذا أردنا استخدام أمر منطقي يمكننا الاستعانة بالأمر XOR حيث نعلم أن

0 XOR 0 = 0 و 1 XOR 1 = 0

وبالتالي يمكننا استخدام الأمر XOR للمسجل مع نفسه لنضع الرقم صفر فيه علي النحو التالي

```
XOR   AX , AX
```

اختبار وجود الرقم صفر في مسجل

لأن '0' OR '0' = '0' و '1' OR '1' = '1' فإن الأمر OR AX , AX يبدو كأنه لا يفعل شيئاً حيث لا يتم تغيير محتويات المسجل AX بعد تنفيذ الأمر، ولكن الأمر يقوم بالتأثير علي بيرق الصفر ZF و بيرق الإشارة SF فإذا كان المسجل AX يحوي الرقم صفر فسيتم رفع بيرق الصفر (ZF = 1) وبالتالي يمكن استخدام هذا الأمر بدلاً من استخدام الأمر CMP AX , 0

الأمر NOT

يقوم الأمر NOT بحساب المكمل لواحد 1's Complement (وهو تحويل الـ '0' إلي '1' والـ '1' إلي '0' أي عكس الخانات بداخل المسجل) والشكل العام للأمر هو :

NOT Destination

ومثال له الأمر NOT AX

الأمر TEST

يقوم الأمر TEST بعمل الأمر AND ولكن بدون تغيير محتويات المستودع Destination والهدف منه هو التأثير علي بيارق الحالة والشكل العام للأمر هو

TEST Destination , Source

ويقوم بالتأثير علي البيارق التالية:

البيارق PF و ZF و SF تعكس النتيجة

البيرق AF غير معرف

البيارق OF و CF تحتوي علي الرقم ٠

إختبار خانة أو خانات محددة

يستخدم الأمر TEST لاختبار محتويات خانة أو خانات محددة ومعرفة إن كان بها '1' أو '0' حيث يتم استخدام حجاب Mask ووضع الرقم '1' في الخانات المطلوب اختبارها ووضع الرقم '0' في الخانات الغير مطلوب معرفة قيمتها وذلك لأن 0 AND b = 0 و 1 AND b = b ويتم استخدام الأمر

TEST Destination , Mask

وبالتالي فإن النتيجة ستحتوي علي الرقم '1' في الخانة المراد اختبارها فقط إذا كانت هذه الخانة تحتوي علي الرقم '1'، وتكون صفر في كل الخانات الأخري.

مثال:

اختبر قيمة المسجل AL وإذا احتوى على رقم زوجي قم بالقفز إلى العنوان event_

الحل

الأرقام الزوجية تحتوي على الرقم ٠ في الخانة ذات الوزن الأصغر LSB وعلى ذلك لاختبار هذه الخانة يتم استخدام الحجاب MASK التالي ٠٠٠٠٠٠٠1b ويكون البرنامج على الصورة التالية :

```
TEST        AL , 01h

JZ          Even_No
```

```
.model small
.stack 100h
.data
msg1 db "event"$
msg2 db "odd"$
.code
main proc
mov ax,@data
mov ds,ax

mov ah,01h
int 21h

test al,01h
jz  _event

jmp _odd
_event:
LEA DX,msg1
jmp _EXIT

_odd:
LEA DX,msg2
_EXIT:
mov ah,09h
int 21h

mov ah,04ch
int 21h
main endp
end main
```

اكتب برنامج يقوم بتعبئة مصفوفة من لوحة المفاتيح ؟

```
.model small
.stack 100h
.data
maxlen db 10
actlen db ?
buffer db 10 dup (' ')
.code
.startup
mov ax,@data
mov ds,ax

mov dx,offset maxlen
mov ah,0ah
int 21h
mov dl,0ah
mov ah,02h
int 21h
mov dl,0dh
int 21h

xor bx,bx
mov dl,actlen
or dl,30h
mov ah,02h
int 21h
```

```
mov dl,0ah
int 21h
mov dl,0dh
int 21h
xor bx,bx
mov bl,actlen

mov buffer[bx],'$'
mov dx,offset buffer
mov ah,09h
int 21h
.exit
end
```

العمليات الحسابية :

١- الجمع ADD instruction

الصيغة العامة :

Add dest,src ; dest=dest+src

Adc dest,src ;dest=dest+src+carry flag

الشكل العام بالتفصيل The form in detail are :

Add reg,reg

Add reg,mem

Add mem,reg

Add reg,immediate data

Add mem,immediate data

مسجلات الحالة التي تتاثر بعملية الجمع هي :

OF,CF,PF,AF,SF,ZF

أمثلة توضح كيفية كتابة التعبير الحسابي باستخدام عملية الجمع :

مثال ١ : J=K+M

```
Mov ax,K
Add ax,M
Mov J,ax
```

مثال ٢: J=K+M+N+P

```
Mov ax,K
Add ax,M
Add ax,N
```

```
Add ax,P

Mov J,ax
```

مثال٣: J=K+J

```
Mov ax,J

Mov bx,K

Add ax,bx;j=j+k

Mov J,ax
```

OR

```
Mov ax,J

Add ax,K

Mov J,ax
```

OR

```
Mov ax,K

Add J,ax
```

مثال٤: J=J+2

```
Add j,2
```

٢- الطرح Sub instruction

Sub dest,src;dest=dest-src

Sbb dest,src;dest=dest-src-carry flag

الشكل العام :

الشكل العام بالتفصيل The form in detail are :

Sub reg,reg

Sub reg,mem

Sub mem,reg

Sub reg,immediate data

Sub mem,immediate data

مسجلات الحالة التي تتاثر بعملية الجمع هي :

OF,CF,PF,AF,SF,ZF

أمثلة توضح كيفية كتابة التعبير الحسابي باستخدام عملية الطرح :

مثال١ : J=K-J

```
Mov ax,K
Sub ax,J
Mov J,ax
```

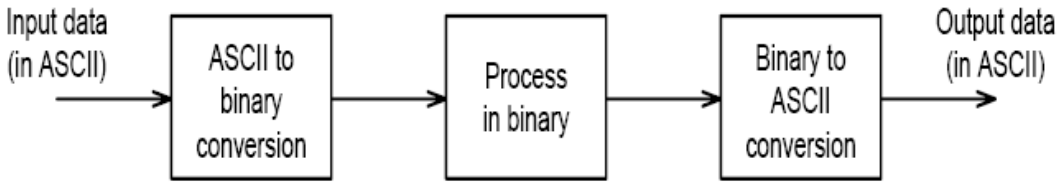
مثال٢ : J=J-(K+M)

```
Mov ax,K
Add ax,M
Sub J,ax
```

برمجة العمليات الحسابية :

مقدمة:

عند القراءة من لوحة المفاتيح ستخزن كل الأرقام في الذاكرة بحسب ترميزها في جدول ascii ،وعند عملية المعالجة تتم في الشكل الثنائي ،وعند طباعة أي حرف أو رقم يجب أن يكون في صورة ASCII ، والشكل التالي يوضح ذلك:



حساب اسكي ascii arithmetic:

بدلا من تحويل عدد مدخل (التي تخزن في رمز أسكي) إلى ثنائي ثم نقوم عملية حسابية، يمكننا القيام به في عملية حسابية مباشرة على أسكي والقيام ببعض التعديلات اللازمة في نهاية المطاف.

هي ان نقوم بأجراء العمليات الحسابية على الاعداد المدخلة وبعدها نقوم بتصحيح النتيجة بمساعدة امر معين للحصول على شكل قيمة عشرية. حتى يكون التصحيح بسيطا قاموا بانشاء كود BCD :

للتمثيل الارقام العشرية في الذاكرة

ما يميز هذا الكود هو ان:

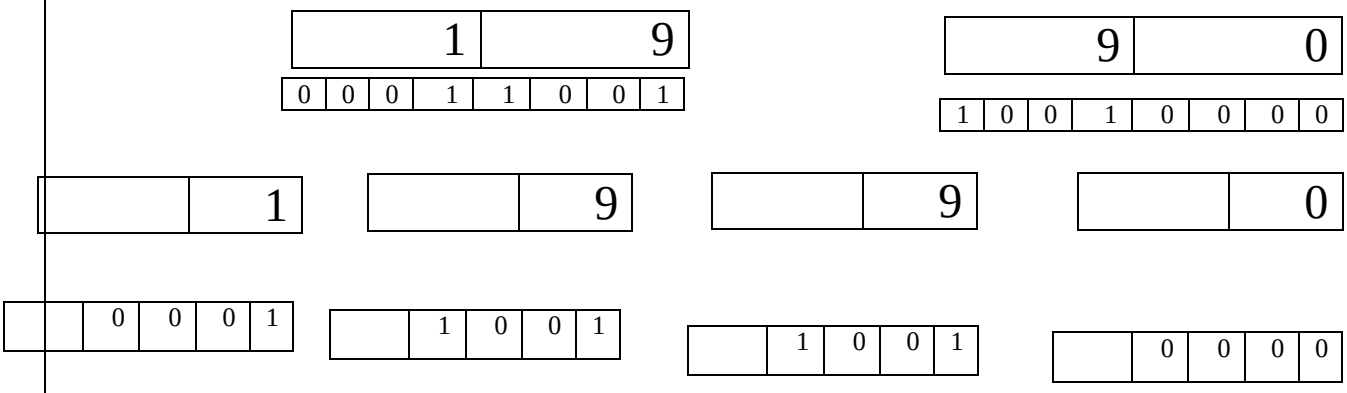
الرموز المستعملة في هذا الكود هي 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1, 0

وهناك نوعين لتمثيل لهذا الكود :

١- **الصيغة المحزومة packed BCD : في هذه الحالة كل بايت يحتوي على رقمين من BCD أي كل رقم من BCD بحاجة إلى ٤ بتات**

٢- **الصيغة المفكوكة unpacked BCD : في هذه الحالة كل بايت يحتوي على رقم واحد من BCD فقط أي رقم من BCD بحاجة الى ٨ بتات، يحتل هذا الرقم النصف الأيمن من البايت،تكون الخانات الاربع اليسرى أم صفرا في حالة القسمة و الضرب او أي قيمة في حالة الجمع والطرح .**

مثال : مثل الرقم 1990 في الصيغتين :



اليكم الجدول التالي الذي يظهر بعض الاعداد في الميزان العشري وكود BCD

الميزان العشري	Packed BCD	Unpacked BCD
0	0H	00H
1	1H	01H
2	2H	02H
3	3H	03H
4	4H	04H
5	5H	05H
6	6H	06H
7	7H	07H
8	8H	08H
9	9H	09H
11	11H	0101H
12	12H	0102H
58	58H	0508H
215	215H	020105H

عمليات تصحيح النتيجة

الان ننظر الى العمليات الحسابية التالية:

الميزان العشري	BCD-code
12	12H
+11	+11H
---	-----
23	23H

النتيجة السابقة صحيحة اذا في هذه الحالة النتيجة ليست بحاجة الى تصحيح

ننظر الى العملية التالية:

الميزان العشري	BCD-code	الثنائي
12	12H	0001 0010
+19	+19H	+
---	-----	0001 1001
31	2BH	0010 1011

اذا نظرنا الى النتيجة فان النتيجة تحتوي على رمز B وهذا الرمزغير مسموح فيه في BCD-CODE بمعنى أخر(أنها تحتوى على رقم اكبر من العدد 9 وهذا خارج نظام BCD-CODE :

و بتالي لابد من إيجاد حل لهذه المشكلة ، حيث تم إيجاد خوارزمية للتعامل معها :

١- عملية الجمع :

اولاً : بصيغة المفكوكة

يتم فحص النصف الايمن من المسجل AL:

١- فإذا احتوى على رقم يتراوح بين 0 و 9 يتم تصفير النصف الايسر من المسجل AL ووضع الرايات AF,CF في حالة الصفر.

٢- إما إذا كان النصف الايمن من المسجل AL يحتوي على قيمة أكبر من 9 أو كانت الراية AF في حالة "١" تنفذ الخطوات التالية :

a. اضافة الرقم 6 الى المسجل AL.

b. اضافة الرقم 1 الي المسجل AH.

c. وضع الرايات AF,CF في حالة "1".

d. تصفير النصف الايسر من المسجل AL.

وبهذا تكون القيمة الناتجة في المسجل AL رقماً عشرياً مفكوكاً صحيحاً.

ثانياً: بصيغة المحزومة :

تعمل هذه الصيغة نفس الصيغة السابقة غير إنها تعالج الرقمين في المسجل AL:

- **إذا كان النصف الايمن من المسجل AL يحمل قيمة أكبر من 9 أو AF=1 يضاف الرقم 6 الي المسجل AL وتوضع الراية AF في حالة واحد "١".**
- **اذ اكان النصف الايسر من المسجل AL أكبر من 9 أو كانت الراية CF في حالة واحد"١" تتم اضافة الرقم 60H الى المسجل AL وتوضع الراية CF في حالة "١".**

وبهذا يحتوى المسجل AL دائماً على رقمين عشريين صحيحين في الصيغة المحزومة .

الخوارزمية التي ندرسها للبرمجة العملية الحسابية (الصيغة المفكوكة):

```
If (al and Of >09 or AF=1)
al=al+06
ah=ah+1
AF set
CF set
END IF
```

امثلة علي جمع عددين باستخدام BCD:

مثال ١: =2+3

0000 0011+

0000 0010

0000 0101 (5)

مثال٢ : 8+7=

0000 1000(in al)+

0000 0111

0000 1111(i5 in al)

نطبق الخطوات :

0000 1111+ Al=al+6

0000 0110

0001 0101

Ah=ah+1,so ah=0000 00001

AF=1,CF=1

Al=al and 0f=0000 0101(5)

امثله على جمع عددي من ارقام اسكي كود (و هي الطريقة التي سوف ندرسها):
نطبق الخوارزمية السابقة على الامثلة الاتية لكي نعرف كيف تعمل تعليمات اللغة

مثال ١: '4'+'5'=34h+35h =

0011 0100 + in al

0011 0101

0110 1001

نلاحظ ان العدد ليس اكبر من العدد 9 وكذلك المسجل AF=0

مثال٢: '8'+'7'=38h+37h

0011 1000 +

0011 0111

0110 1111 (15)

نلاحظ البايت السفلي يحتوى على عدد اكبر من 9 لذلك نطبق علية الخوارزمية السابقة وتكون النتيجة النهائية AH=1,AL=5

مثال ٣ : '9'+'8'=39h+38h

0011 1001 +

0011 1000

0111 0001

نلاحظ ان البايت السفلى يحتوي على عدد اقل من 9 لكن مسجل الراية AF=1 لذلك نظبق علية الخوارزمية السابقة وتكون النتيجة النهائية هي : AH=1,AL=7

برنامج يقوم بجمع رقمين اسكي كود؟

```
.model small
.stack 100h
.data
 x db '8'
 y db '9'
.  code
.  startup
  mov ax,@data
  mov ds ,ax
xor ax,ax
mov al,x
add al,y
IFF:mov cl,al
and cl,0fh
cmp cl,09
jg thenn
ORR:mov cl,al
and cl,0f0h
cmp cl,60h
jg thenn
jmp finish
thenn:
add al,06
```

```
mov ah,01
stc
finish:
and al,0fh
or ax,3030h
mov cx,ax
mov dl,ch
mov ah,02h
int 21h
mov dl,cl
mov ah,02
int 21h
.exit
end
```

توضيح البرنامج :

and cl,0fh تستخدم للتصفير البايت العلوي للمسجل CL

cmp cl,09 تستخدم للمقارنة البايت السفلي يحتوى على عدد اكبر من 9

and cl,0f0h تستخدم للتصفير البايت السفلي من اجل اختار مسجل الراية AF هل يساوي واحد ام لا وذلك بتنفيذ التعليمة التالية

cmp cl,60h تعادل الشرط If(al and 0f0h)>6

- **من خلال ما سبق نلاحظ ان العملية طويلة ولذلك توفر لنا لغة التجميع تعليمة تقوم بتنفيذ الخوارزميات السابقة وهي :**
- **AAA: ليس لها معامل تستخدم للتنفيذ خوارزمية الصيغة المفكوكة للعملية الجمع**
- **DAA: ليس لها معامل تستخدم للتنفيذ خوارزمية الصيغة المحزومة للعملية الجمع**

AAA:نفس المثال السابق باستخدام التعليمة

```
.model small
.stack 100h
.data
 x db '8'
 y db '9'
.  code
.  startup
  mov ax,@data
  mov ds ,ax
xor ax,ax
mov al,x
add al,y
aaa
or ax,3030h
mov cx,ax
mov dl,ch
mov ah,02h
int 21h
mov dl,cl
mov ah,02
int 21h
.exit
End
```

مثال : جمع عددين بحيث كل عدد يحتوي على خانتين

```
.model small
.stack 100h
.data
x db '97'
y db '26'
z db 3 dup(?)
.code
.startup
mov ax,@data
mov ds,ax
xor ax,ax
mov al,x[1]
add al,y[1]
aaa
mov z[2],al
mov ah,0
mov al,x[0]
adc al,y[0]
aaa
mov z[1],al
mov z[0],ah
or z[0],30h
or z[1],30h
```

```
or z[2],30h

mov dl,z[0]

mov ah,2h

int 21h

mov dl,z[1]

int 21h

mov dl,z[2]

int 21h

.exit

end
```

مثال برنامج يطلب من المستخدم إدخال رقمين بحيث كل رقم يتكون من خانتين ويطبع مجموعهما؟

```
.model small
.stack 100h
.data
x db 3,?,3 dup(?)
y db 3,?,3 dup(?)
z db 3 dup(?)
.code
.startup
mov ax,@data
mov ds,ax

xor ax,ax

mov dx,offset x
mov ah,0ah
int 21h
mov dx,offset y
mov ah,0ah
int 21h

mov al,x[3[
add al,y[3[
aaa
```

```
mov z[2],al

mov ah,0
mov al,x[2[
adc al,y[2[
aaa
mov z[1],al
mov z[0],ah

or z[0],30h
or z[1],30h
or z[2],30h


mov dl,z[0[
mov ah,2h
int 21h

mov dl,z[1[
int 21h

mov dl,z[2[
int 21h

.exit
end
```

عملية الطرح

اولاً : بصيغة المفكوكة

يتم فحص النصف الايمن من المسجل AL:

فإذا احتوى على رقم يتراوح بين 0 و 9 يتم تصفير النصف الايسر من المسجل AL ووضع الرايات AF,CF في حالة الصفر.

إما إذا كان النصف الايمن من المسجل AL يحتوي على قيمة أكبر من 9 أو كانت الراية AF في حالة "١" تنفذ الخطوات التالية :

e. **يطرح الرقم 6 من المسجل AL.**
f. **يطرح الرقم 1 من المسجل AH.**
g. **وضع الرايات AF,CF في حالة "1".**
h. **تصفير النصف الايسر من المسجل AL.**

وبهذا تكون القيمة الناتجة في المسجل AL رقماً عشرياً مفكوكاً صحيحاً.

ولعمل ذلك نستخدم التعليمة AAS

ثانياً: بصيغة المحزومة :

تعمل هذه الصيغة نفس الصيغة السابقة غير إنها تعالج الرقمين في المسجل AL:

- **إذا كان النصف الايمن من المسجل AL يحمل قيمة أكبر من 9 أو AF=1 يطرح الرقم 6 من المسجل AL وتوضع الراية AF في حالة واحد "١".**
- **اذ اكان النصف الايسر من المسجل AL أكبر من 9 أو كانت الراية CF في حالة واحد"١" يتم طرح الرقم 60H من المسجل AL وتوضع الراية CF في حالة "١".**

وبهذا يحتوى المسجل AL دائماً على رقمين عشريين صحيحين في الصيغة المحزومة .

ولعمل ذلك نستخدم التعليمة DAS

مثال : طرح عددين بحيث كل عدد يحتوي على خانتين

```
.model small
.stack 100h
.data
 x db '73'
 y db '89'
 z db 3 dup(؟)
.  code
.  startup
  mov ax,@data
  mov ds ,ax
xor ax,ax
mov al,x

cmp al,y
jge NEXTT

mov cx, word ptr x
xchg  cx,word ptr  y
mov word ptr x,cx
mov ah,2h
mov dl,'-'
int 21h
```

```
NEXTT:

mov al,x[1]
sub al,y[1]
aas
mov z[2],al
mov ah,0
mov al,x[0]
sbb al,y[0]
aas
mov z[1],al
mov z[0],ah

or z[0],30h
or z[1],30h
or z[2],30h
mov dl,z[0]
mov ah,2h
int 21h
mov dl,z[1]
int 21h
mov dl,z[2]
int 21h
.exit
End
```

أوامر الضرب والقسمة
Multiplication and Division Instructions

في هذه المحاضرة سنقوم بتوضيح العمليات التي تقوم بعمليات الضرب والقسمة على أعداد

تختلف عمليات الضرب للأرقام بإشارة منها في حالة الأرقام بدون إشارة وكذلك عمليات القسمة وبالتالي لدينا نوعين من أوامر الضرب والقسمة أحدهما للأرقام بإشارة والأخرى للأرقام بدون إشارة وكذلك هناك صور للتعامل مع أرقام بطول 8 خانات فقط وأخرى للتعامل مع أرقام بطول 16 خانه.

عمليات الضرب MUL & IMUL

نبدأ مناقشة عمليات الضرب بالتفرقة بين الضرب بإشارة والضرب بدون إشارة فعلى سبيل المثال إذا تم ضرب الرقمين الثنائيين 10000000 و 11111111 فلدينا هنا تفسيرين للرقمين. التفسير الأول هو أن الأرقام ممثله بدون إشارة وبالتالي فإن المطلوب هو ضرب الرقم 128 في الرقم 255 ليصبح الناتج 32644. أما التفسير الثاني هو أن الأرقام عبارة عن أرقام بإشارة فإن المطلوب هو ضرب الرقم 128- في الرقم 1- لتصبح النتيجة 128 وهى نتيجة مختلفة تماماً عن النتيجة التي تم الحصول عليها في التفسير الأول (32640).

لأن عمليات الضرب للأرقام بإشارة تختلف عن عمليات الضرب للأرقام بدون إشارة يتم استخدام أمرين:

الأول: يستخدم في عمليات الضرب للأرقام بدون إشارة وهو الأمر **MUL (Multiply)**.

والثاني: يستخدم في عمليات الضرب للأرقام بإشارة وهو **IMUL (Integer Multiply)**.

تقوم هذه الأوامر بعملية الضرب لرقمين بطول 8 خانات ثنائية ليكون حاصل الضرب بطول 16 خانه ثنائية أو لضرب رقمين بطول 16 خانه ثنائية ليكون حاصل الضرب بطول 32 خانه ثنائية. والصيغة العامة للأمرين هي:

MUL Source

& IMUL Source

هنالك صورتان للتعامل مع هذه الأوامر الأولى عند ضرب أرقام بطول 8 خانات والثانية عند ضرب أرقام بطول 16 خانه

استخدام أرقام بطول 8 خانات Byte Form

حيث يتم ضرب الرقم الموجود في المسجل AL في الرقم الموجود في المصدر Source وهو إما محتويات مسجل أو موقع في الذاكرة (غير مسموح باستخدام ثوابت). يتم تخزين النتيجة (بطول 16 خانه) في المسجل AX.

استخدام أرقام بطول 16 خانات Word form

في هذه الصورة يتم ضرب الرقم الموجود في المسجل AX في الرقم الموجود في المصدر وهو إما مسجل أو موقع في الذاكرة (غير مسموح باستخدام ثوابت). يتم تخزين النتيجة (32 خانه) في المسجلين AX , DX بحيث يحوى AX على النصف السفلي و DX على النصف العلوي وتكتب النتيجة عاده على الصورة DX:AX. { النصف السفلي : النصف العلوي }

في حالة ضرب الأرقام الموجبة نحصل على نفس النتيجة عند استخدام الأمرينIMUL, MUL.

<u>تأثر البيارق بأوامر الضرب</u>

لا تتأثر بأوامر الضرب كل من البيارق SF, ZF, AF, PF

<u>أما بالنسبة للبيرقين Cf/Of :</u>

أ/ في حالة استخدام الأمر MUL

تأخذ البيارق القيمة (0) (CF/OF = 0) إذا كان النصف العلوي من النتيجة يساوى صفر وتأخذ البيارق القيمة (1) إذا لم يحدث ذلك.

ب/ في حالة استخدام الأمر IMUL

يأخذ البيرق القيمة 0 (CF/OF = 0) إذا النصف العلوي هو عبارة عن امتداد لإشارة النصف السفلي Sign Extension (أي أن كل خانات النصف العلوي

تساوى خانه الإشارة MSB من النصف السفلي) وتأخذ البيارق القيمة (1) (CF/OF = 1) إذا لم يحدث ذلك.

بالنسبة للأمرين نلاحظ أن البيارق CF/OF تأخذ القيمة (1) اذا كانت النتيجة كبيره ولا يمكن تخزينها في النصف السفلي فقط (AL في حالة ضرب رقمين بطول 8 خانات و AX في حالة ضرب رقمين بطول 16 خانه). وبالتالي يجب التعامل مع باقي النتيجة والموجود في النصف العلوي.

بعد توضيح آلية التعامل مع عملية الضرب والقسمة نبدأ بكتابة البرامج

١- ضرب الارقام الممثلة بنظام ASCII

تتلخص عملية ضرب الارقام الممثلة بنظام ASCII في الخطوات الآتية :

- تحويل الارقام المشاركة في عملية الضرب الي الصيغة المفكوكة
- تضرب الارقام باستخدام التعليمة MUL
- يعدل حاصل الضرب باستخدام التعليمة AAM
- يحول حاصل الضرب الى نظام ASCII

لنفرض ان المسجل AL يحتوى 35h ،CL يحتوى 39h تستخدم التعليمات التالية لضرب محتويات المسجلين وتحويل الناتج الى نظام ASCII.

- تحويل CL الى الرقم 9 : AND CL,0fh
- تحويل AL الى الرقم 5: AND AL,0fh

MUL CL;AL*CL

تحويل الناتج الى الصيغة المفكوكة AAM

تحويل الناتج الى نظام ASCII ; OR AX,3030h

برنامج يقوم بضرب عددين مكونة من خانة واحدة؟

```
.model small
.stack 100h
```

```
.data
 x db '8'
 y db '9'
.  code
.  startup
  mov ax,@data
  mov ds ,ax
xor ax,ax
mov al,x
AND al,0fh
mov cl,y
AND cl,0fh
mul cl
AAM
or ax,3030h
mov cx,ax
mov dl,ch
mov ah,02h
int 21h
mov dl,cl
mov ah,02
int 21h
.exit
End
```

برنامج لضرب عددين مكون الاول من خانتين والثاني خانة؟

```
.model small
.stack 100h
.data
x db '25'
y db '2'
z db 3 dup(?)
.code
.startup
mov ax,@data
mov ds,ax
xor ax,ax

mov al,x[1]
And al,0fh
mov bl,y
AND bl,0fh
mul bl
AAM

mov z[2],al
mov bh,ah
xor ax,ax
mov al,x[0]
And al,0fh
```

```
mul bl
AAM
add al,bh
mov z[1],al
mov z[0],ah
or z[0],30h
or z[1],30h
or z[2],30h
mov dl,z[0]
mov ah,2h
int 21h
mov dl,z[1]
int 21h
mov dl,z[2]
int 21h
.exit
end
```

برنامج ضرب عدد مكون كل عدد من خانتين (كلمة * كلمة)؟

```
.model small
.stack 100h
.data
x db '35'
y db '17'
z db 4 dup(؟)
k db 4 dup(0)
t db 5 dup(0)
.code
.startup
mov ax,@data
mov ds,ax
xor ax,ax

mov al,x[1]
And al,0fh
mov bl,y[1]
AND bl,0fh
mul bl
AAM

mov z[3],al


mov bh,ah
xor ax,ax
mov al,x[0]
And al,0fh
mul bl
AAM


add al,bh
mov z[2],al

mov z[1],ah

or z[0],30h
or z[1],30h
or z[2],30h
or z[3],30h

mov ah,2h
mov dl,z[0]
mov ah,2h
int 21h
mov dl,z[1]
int 21h
```

```
 mov dl,z[2]
 int 21h
mov dl,z[3]
int 21h


xor ax,ax

mov al,x[1]
And al,0fh
mov bl,y[0]
AND bl,0fh
mul bl
AAM

mov k[2],al


mov bh,ah
xor ax,ax
mov al,x[0]
And al,0fh
mul bl
AAM


add al,bh
mov k[1],al

mov k[0],ah

or k[0],30h
or k[1],30h
or k[2],30h
؛or k[3],30h
xor ax,ax
mov ah,2
mov dl,0ah
int 21h
mov dl,0dh
int 21h
mov dl,k[0]
؛mov ah,2h
int 21h
mov dl,k[1]
int 21h
 mov dl,k[2]
 int 21h
mov dl,k[3]
int 21h
```

```
mov ah,2
mov dl,0ah
int 21h
mov dl,0dh
int 21h
xor ax,ax

mov al,z[3]
add al,k[3]
aaa

mov t[4],al
mov ah,0
mov al,z[2]
adc al,k[2]
aaa
mov t[3],al
mov ah,0
mov al,z[1]
adc al,k[1]
aaa
mov t[2],al
mov ah,0
mov al,z[0]
adc al,k[0]
aaa
mov t[1],al

mov t[0],ah

or t[0],30h
or t[1],30h
or t[2],30h
or t[3],30h
or t[4],30h

mov dl,t[0[
mov ah,2h
int 21h
mov dl,t[1[
int 21h
 mov dl,t[2[
 int 21h
mov dl,t[3[
int 21h
mov dl,t[4[
int 21h
.exit
End
```

أوامر القسمة DIV , IDIV

كما في حالة عمليات الضرب فان عمليات القسمة تختلف عند التعامل مع الأرقام بإشارة عنها في حالة الأرقام بدون إشارة وعلى ذلك نستخدم

في حالة الأرقام بدون إشارة الأمر DIV (Divide)

في حالة الأرقام بإشارة الأمر IDIV (Integer Divide)

والصيغة اللغوية للأمرين كالآتي :

DIV Source

IDIV Source

عند إجراء عملية القسمة نحصل على خارج القسمة في مسجل وباقي عملية القسمة في مسجل آخر.

لدينا صورتين عند استخدام عملية القسمة إما تستخدم أرقام بطول 8 خانات أو أرقام بطول 16 خانة كما يلي:

استخدام أرقام بطول 8 خانات Byte form

في هذه الصورة تتم قسمة الرقم الموجود في المسجل AX على المصدر ويتم تخزين خارج القسمة (٨ بت) في المسجل AL وباقي القسمة (٨ بت) في المسجل AH.

استخدام أرقام بطول 16 خانة Word form

في هذه الصورة يتم قسمة الرقم الموجود في المسجلين DX , AX (على الصورة DX: AX حيث DX به النصف العلوي و AX جهة النصف السفلي) على المصدر ويتم تخزين خارج القسمة في المسجل AX وباقي القسمة في المسجل DX.

في حالة الأرقام بإشارة تكون إشارة الباقي هي نفس إشارة الرقم المقسوم. وإذا كان الرقم المقسوم والمقسوم عليه موجبين تكون النتيجة واحدة عند استخدام IDiv , Div.

بعد تنفيذ أوامر القسمة تكون البيارق كلها غير معرفه.

٢- قسمة الارقام ممثلة بنظام ASCII

تتلخص عملية قسمة الارقام الممثلة في نظام ASCII بالخطوات التالية :

- **تحويل المقسوم و المقسوم علية الى الصيغة العشرية المفكوكة**
- **تحو المقسوم ال النظام الثنائي باستخدام ADD**
- **تقسم الارقام باستخدام DIV**

لنفرض ان AX=3238 CI=37

تستخدم التعليمة التالية لقسمة الرقم 28 على 7

تحويل CL الى 7 AND CL,0FH

تحويل AX الى 0208; AND AX,0f0fH

التحويل الى النظام الثنائي AAD

قسمة AX على CL; DIV CL

برنامج قسمة عددين مكونة من خانة واحدة فقط؟

```
.model small
.stack 100h
.data
x db '4'
y db '2'

.code
.startup
mov ax,@data
mov ds,ax
xor ax,ax

mov bl,y
AND bl,0fh

mov al, x
and ax,0f0fh

AAd
DIV bl
mov cl,al
```

```
or cl,30h
mov ah,2
mov dl,cl
int 21h
.exit
end
```

برنامج قسمة عدد مكون من خانتين على عدد مكون من خانة واحدة (كلمة قسمة بايت)؟

```
.model small
.stack 100h
.data
x db '44'
y db '2'
z db 3 dup(0)
.code
.startup
mov ax,@data
mov ds,ax
xor ax,ax

mov bl,y
AND bl,0fh

mov al, x[1]
and ax,0f0fh

AAd
DIV bl
mov z[2],al

xor ax,ax
mov al, x[0]
and ax,0f0fh

AAd
DIV bl
mov z[1],al

or z[2],30h
or z[1],30h
mov ah,2
mov dl,z[1]
int 21h
mov dl,z[2]
int 21h
.exit
end
```

الماكرو

الماكرو : هو عبارة عن مجموعة اوامر لها اسم ويمكن استدعاء هذه مجموعة الاوامر عن طريق تسجيل اسم الماكرو. يمكن استدعاء الماكرو اكثر من مرة في البرنامج. فيمكن القول ان لاستخدام الماكرو عدة فوائد منها:

١.تقليل الاخطاء فيمكن تسجيل اسم الماكرو بدلا من كتابة الاوامر مرة ثانية وهذا يقلل من الوقوع في اخطاء املائية

٢.زيادة البرنامج وضوحا . فبدلا من كتابة اوامر فإننا نسجل كلمات عادية مفهومة وهي اسماء الماكرو وتدل على ما يعمله هذا الماكرو

٣.سرعة التغيير: اذا اردنا ان نضيف او نحذف اوامر معينة فبدلا من اجراء هذه التغييرات في جميع الاماكن التي تتواجد بها هذه الاوامر فعند استخدام الماكرو يكفي ان نغير في الماكرو.

٤.زيادة سرعة البرنامج لان الماكرو يزرع الاوامر التي يحتويها في المكان الذي نستدعيه فيه

ملاحظة: الماكرو يكبر حجم البرنامج وهذا يعد صفة سلبية للماكرو

انواع الماكرو

يمكن تقسيم الماكرو حسب عدة مواصفات نأخذ التقسيم الاول

١.هنالك انواع لا تأخذ برامترات

٢.هنالك انواع تأخذ برامترات

ماكرو بدون برامترات

المبنى العام للماكرو الذي لا يأخذ برامترات هوكما يلي:

```
Macro name   macro
الاوامر
Endm
```

مثال : اكتب ماكرو يقوم بمسح الشاشة:

الحل:

```
Clear_screen macro
Mov ax,3
Int 10h
Endm
```

مثال اخر

اكتب ماكرو يستقبل من لوحة المفاتيح رمزا

الحل:

```
Read_char macro
Mov ah,1
Int 21h
Endm
```

مثال اخر

اكتب ماكرو يطبع نجمة على الشاشة

الحل:

```
Print_star macro
Mov dl,'*'
Mov ah,2
Int 21h
Endm
```

ملاحظات: عليك الانتباه الى ما يلي:

١.عليك ان تحفظ قيم المسجلات التي سوف تستخدمها في الماكرو اذا كانت تحتوي على قيم مهمة بالنسبة لك وانت بحاجة اليها لان اوامر الماكرو سوف تغير القيم. هنالك عدة طرق لحفظ قيم المسجلات نذكر منها:

أ.قبل استدعاء الماكرو نقوم بتخزين القيم في مسجلات ثانية او نخزنها في الذاكرة وبعد الانتهاء من الماكرو نرجع القيم

مثال:

```
Mov bx,ax
Clear_screen
Mov ax,bx
```

نلاحظ هنا اننا قمنا بحفظ قيمة المسجل ax في المسجل bx قبل استدعاء الماكرو وبعد استدعاء الماكرو

تمت عملية ارجاع القيمة الى المسجل ax

ب.يمكن ان نحفظها في داخل الماكرو قبل القيام في الاوامر الرئيسية مثلا

```
Clear_screen macro
Push ax
Mov ax,3
Int 10h
Pop ax
Endm
```

نلاحظ في التمرين السابق اننا حفظنا الماكرو في المكدسة (stack segment) قبل ان نقوم بعملية مسح

الشاشة وقبل الخروج من الماكرو تمت عملية ارجاع القيم من المكدسة

٢.يمكن استدعاء ماكرو من داخل ماكرو اخر

مثال:

طور ماكرو يقوم بمسح الشاشة واستقبال رمز من لوحة المفاتيح ثم يطبع الرمز

الحل:

```
Print_mac macro
Clear_screen
Read_char
Mov dl,al
Mov ah,2
Int 21h
Endm
```

هنا تم استدعاء الماكرو clear_screen والماكرو readl_char

٣.عندما نستخدم LEBLE (عنوان) في الماكرو يجب ان نعرفها بمساعدة الكلمة المحفوظة LOCAL أي اننا نقول لبرنامج الترجمة بان هذا العنوان هو عبارة عن عنوان محلي معروف في الماكرو فقط .

ملاحظة: عند استخدام عنوان والتوجه اليه اكثر من مرة عن طريق اوامر الماكرو او استدعاء الماكرو اكثر من مرة فان برنامج الترجمة سوف يعلن عن خطأ

مثال:

اكتب ماكرو يطبع الارقام من ٠ الى ٩

الحل:

```
Print_numbers macro
Local t1
Mov cx,10
Mov dl,'0'
T1:mov ah,2
    Int 21h
    Inc dl
Loop t1
Endm
```

ملاحظة: اذا كان لدينا اكثر من عنوان محلي في الماكرو نسجل العناوين ونفصل بين العناوين بمساعدة فاصلة

```
Local t1,t2,t3
```

٤.للخروج من الماكرو من أي نقطة نريدها أي قبل نهاية الماكرو نستخدم الامر exitm

مثال:

```
Check macro
Local t1,t2
Mov cx,10
```

```
T1:Mov ah,1
    Int 21h
    Cmp al,'B'
    Jz t2
    Inc dh
    exitm
T2:dec dh
    loop t1
endm
```

حل اخر بدون استخدام exitm

```
Check macro
Local t1,t2,t3
Mov cx,10
T1:Mov ah,1
    Int 21h
    Cmp al,'B'
    Jz t2
    Inc dh
    Jmp t3
T2:dec dh
    loop t1
t3:nop
endm
```

٥.اذا اردنا ان نسجل ملاحظات في داخل الماكرو نستخدم الفاصلة المنقوطة مرتين

مثال:

```
Print_char macro
Mov ah,1  ;;put 1 in ah
Int 21h
Mov dl,al  ;;put the ascii_code of the char in dl
Mov ah,2
Int 21h
Endm
```

اكتب برنامجا.يطبع على الشاشة الجملة enter a char

الحل:

```
.model small
.stack 100h
.data
msg db 'enter a char$'
print_msg macro
lea dx,msg
mov ah,9
int 21h
endm
.code
.startup
  print_msg
.exit
end
```

الماكرو الذي يأخذ برامترات

المبنى العام للماكرو مع برامترات

```
Macro name macro p1,p2,….,pn
الاوامر
Endm
```

P1,p2,…pn : هي البرامترات

مثال :اكتب ماكرو يتلقى برامترين من نوع byte ويحفظ بالمسجل al البرامتر الاكبر

```
.model small
.stack 100h
.data
a db 37h
b db 32h
Max_num macro x,y
Local t1,finish
Mov al,x
Cmp y,al
Jg t1
Jmp finish
T1:mov al,y
Finish:
Endm
.code
.startup
 max_num a,b
 mov dl,al
```

```
mov ah,2h
int 21h
.exit
End
```

ملاحظة: عندما نريد استخدام الماكرو في موقع معين في البرنامج نسجل اسم الماكرو ونسجل

البرامترات المطلوبة.

مثلا في حالة استدعاء الماكرو السابق نسجل

```
Max_num DH,DL
```

هنا تم اعطاء الماكرو المسجلات DH و DL كبرامترات

```
Max_num 7,9
```

هنا تم اعطاء الماكرو القيم ٧ ، ٩ كبرامترات

```
Max_num x1,x2
```

الان ننتقل الى نوع جديد وهو الماكرو المتكرر

وهو ماكرو يقوم بتنفيذ مجموعة الاوامر الموجودة فيه عدد مرات معروفة مسبقا

يسجل هذا الماكرو بشكل عام داخل مقطع البيانات

المبنى العام لهذا الماكرو

```
Rept  n
الاوامر
Endm
```

بحيث n عبارة عن قيمة عددية او تعبيرا يرجع قيمة عددية او اسم ثابت من الثوابت المعرفة في البرنامج او امر يرجع قيمة عددية . بالاختصار المهم ان يكون قيمة عددية

مثال:

اكتب ماكرو يطبع النجمة ٥ مرات

```
Rept 5
Mov dl,'*'
Mov ah,2
Int 21h
Endm
```

سؤال: ماذا ينفذ الماكرو التالي:

```
Rept 4
Add ax,2
Endm
```

الجواب: الماكرو السابق يمكن تغييره بالاوامر التالية

```
Add ax,2
Add ax,2
Add ax,2
Add ax,2
```

أي انه يجمع ٨ للمسجل ax