الفصل الخامس عشر 15

برمجة المعالج
Intel 8086/8088
والمصحح Debugger

15-1 مقدمة

سنرى في هذا الفصل الخطوات الأولى في اتجاه كتابة برنامج بسيط بلغة التجميع (الأسمبلى) الخاصة بالمعالج 8086/8088 ، ومن ثم تنفيذه ، كل ذلك من خلال برامج بسيطة نقدمها فقط لنفهم منها مكونات برامج لغة التجميع للمعالج 8086 . بعد ذلك يعرض هذا الفصل لمجموعات أوامر هذه اللغة عرضا سريعا الغرض منه هو التعريف بأهم مفردات هذه اللغة . بالطبع سيتبقى هناك الكثير من الأوامر الأقل شيوعا ولكنها قد تفيد فى الكثير من التطبيقات ولكننا لن نتعرض لها هنا ونحيل القارئ إلى أحد الكتب المتخصصة فى لغة التجميع . إن الأمور عادة لا تأتي بكل ما يتمناه المبرمج ، حيث كثيرا ما نجد أن البرنامج يحوي العديد من الأخطاء التى تعوق تنفيذ البرنامج بالصورة المطلوبة . سنرى في هذا الفصل أيضا بإذن الله كيفية استخدام برنامج debugger أو المصحح "ديبجر" ا سـ تخراج هذه الأخطاء وا لـ تخ لص م نها .. سنستخدم التربو أسمبلر Turbo Assembler لتنفيذ كل الأمثلة التى سنسوقها فى هذا الفصل .

15-2 خطوات كتابة وتنفيذ برامج لغة التجميع

1. نبدأ بكتابة برنامج لغة التجميع مستخدمين الأوامر المختلفة لهذه اللغة كما سنرى تباعا بعد ذلك . يجب أن يتضمن البرنامج بعض الأوامر الموجهة للمجمع (الأسمبلر) لإخباره عن المتطلبات التي يحتاجها عند تحويل البرنامج إلي لغة الماكينة . وأول هذه الأوامر هو أمر إخبار المجمع عن مكان وضع البرنامج في الذاكرة مثلا ، وأيضا عن مكان وضع البيانات الناتجة عن البرنامج .

بعد الانتهاء من كتابة البرنامج يجب أن يخزن في ملف file بأي أسم مع مراعاة أن يكون امتداد هذا الملف asm. ، فمثلا يمكن تسمية الملف بأي واحد من الأسماء التالية :

Example.asm
Test.asm

2. بعد ذلك يتم استدعاء الأسمبلر وإدخال الملف الذي تمت كتابته في الخطوة 1 عليه باستخدام الأمر TASM Example.asm ، حيث سيعطينا الأسمبلر نتيجة ذلك ملف جديد بنفس الاسم السابق ولكن بامتداد مختلف ؛ هذا الملف سنسميه ملف الهدف object file وسيكون كالتالي :

example.obj

Test.obj

هذه الصورة من البرنامج تكون مكتوبة في صورة لغة الآلة ، ولكنها مازالت غير مناسبة للتنفيذ بواسطة المعالج .

3. يتم بعد ذلك إدخال الملف السابق "ملف الهدف" على البرنامج الرابط linker الذي يقوم بتجميع الأجزاء المختلفة للبرنامج ، ووضعه في صورة مناسبة قابلة للتنفيذ executable بواسطة المعالج وذلك باستخدام الأمر التالى : Tlink Example.obj . هذه الصورة الجديدة للملف ستكون بنفس الاسم ولكن بامتداد جديد وهو EXE. وذلك كما يلي :

Example .EXE
Test .EXE

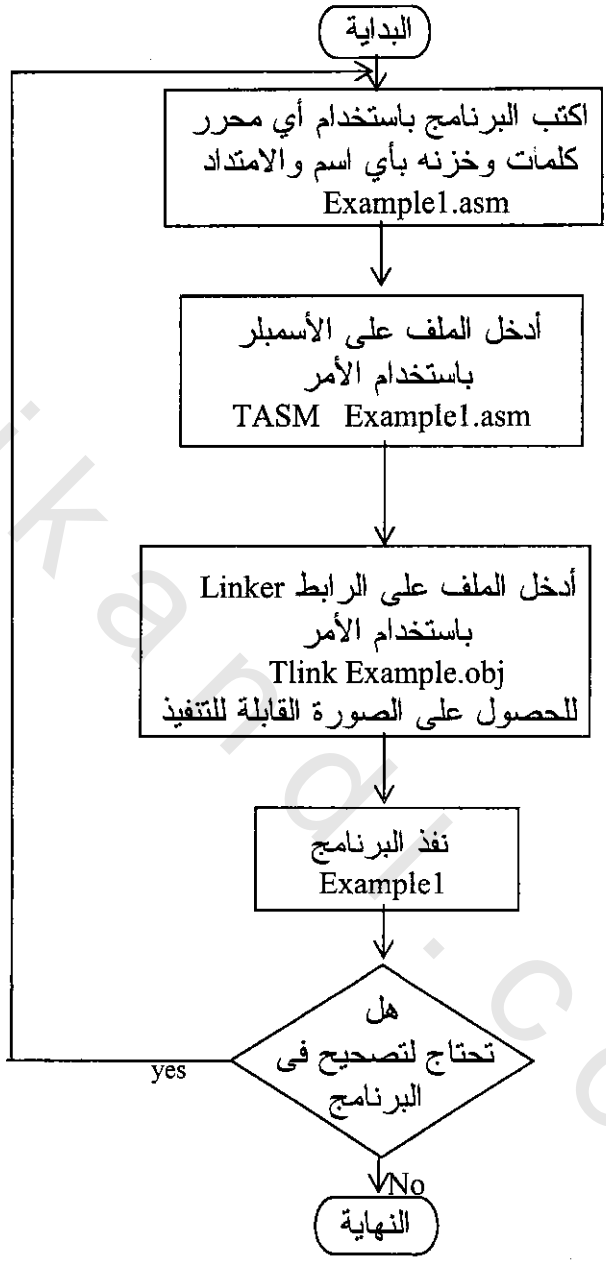
بعد الانتهاء من الخطوات الثلاث السابقة يمكن تنفيذ البرنامج ، وكذلك يمكن رؤية خرجه ؛ فإذا كان الخرج على ما يرام . . . نكون قد انتهينا من البرنامج ، أما إذا جاءت النتائج على خلاف ما نتوقع ، فإن ذلك يدل على وجود أخطاء في البرنامج . . . فكيف يمكننا إذن الكشف عن هذه الأخطاء والتعامل معها ؟ إن هذا يتم عن طريق استخدام برنامج الديبجر ، وسنرى في هذا الفصل كيفية الدخول في هذا البرنامج واستخدامه . شكل (15-1) يبين رسما توضيحيا لكتابة برنامج بلغة الأسمبلى ، وخطوات تنفيذه ، وذلك بفرض أن البرنامج تمت كتابته في ملف اسمه Example1.asm .

15-3 مكونات برنامج الأسمبلى

لكي تتعرف على مكونات برنامج لغة التجميع ، سنسوق المثال الأول الذي يكتب الرسالة الآتية " آهلا يا عرب ، استيقظوا " على الشاشة ، وذلك دون أن ننشغل بتفاصيل البرنامج الآن ، حيث سيرد ذكرها فيما بعد بإذن الله .

مثال 15-1

```
DOSSEG
.MODEL SMALL
.STACK 100H
.DATA
message DB ' Hello Arab , woke up ' ,B,10,'$'
.CODE
mov ax, @data
mov ds,ax ; set ds to the beginning of the data segment
```



شكل (15-1) خطوات كتابة وتنفيذ برنامج بلغة التجميع .

```
mov ah,9 ; load register ah with 9
mov dx , offset message
int 21h
mov ah , 4ch
int 21h
END
```

من هذا المثال نرى الآتي :

1. وجود مجموعة من الأوامر في أول البرنامج وهي عبارة عن أوامر توجيهية للأسمبلر Assembler directives . هذه الأوامر تخبر الأسمبلر عن حقائق معينة يريد المبرمج أن يأخذها في الاعتبار ، مثل تحديد جزء ذاكرة البيانات data segment ، وجزء ذاكرة البرنامج code segment ، وجزء ذاكرة المكدسة stack segment وكذلك نهاية البرنامج .
2. القسم الثاني من الأوامر هو أوامر لغة الأسمبلى مثل الأوامر mov و add و sub وغيرها ، وكلها عبارة عن أوامر سيقوم الأسمبلر بتحويلها إلي شفرات لغة الآلة و تخزينها في الذاكرة . لاحظ أن الأوامر التوجيهية التي سبق الإشارة إليها لا يتم تحويلها إلي شفرات لأنها ليس لها شفرات أصلا ، وذلك لأنها ليست أوامر قابلة للتنفيذ بواسطة المعالج ، ولكنها مجرد توجيهات للأسمبلر لا يراها المعالج . فيما يلي سنأخذ فكرة موجزة عن أوامر التوجيه الموجودة في البرنامج السابق ، وهي كما يلي :

1- أمر التوجيه DOSSEG

هذا الأمر هو أول ما يكتب في أي برنامج أسمبلي ، وهذا الأمر يجعل كل أجزاء البرنامج (البيانات والكود) تتبع نظام الميكروسوفت في التجزيء ، وهذا النظام لا يعنينا هنا في شئ ، ولن ننظر في أية تفاصيل أخرى له طالما أن هذا الأمر يقوم بهذه المهمة . المهم هنا هو أن نبدأ البرنامج بهذا الأمر كما ذكرنا .

2- أمر التوجيه .MODEL

يحدد هذا الأمر للأسمبلر موديل الذاكرة الذي سيتم التعامل معه ، حيث تبعا لهذا الموديل سيتحدد ما إذا كانت البيانات التي سيتعامل معها المعالج قريبة ؛ بحيث يتم عنونتها ب16 بت فقط داخل الجزء الخاص بها ، أم بعيدة فيتم التعامل معها على أساس 32 بت ، 16 منها تحدد العنوان داخل الجزء و 16 أخرى تحدد مكان أو بداية هذا الجزء . وهناك أكثر من موديل للذاكرة يتعامل معها الأسمبلر كما يلي :

1- الموديل tiny

في هذا الموديل يكون البرنامج والبيانات موجودة في نفس الجزء حيث الجزء يبلغ 64 ك ب .

2- الموديل small

يوجد كود البرنامج في جزء أو مقطع (64 ك ب) وبيانات البرنامج في جزء آخر منفصل عن الأول ولكن كلا من البرنامج والبيانات لا يتعدى الجزء الموجود فيه .

3- الموديل compact

يوجد كود البرنامج في جزء معين ، أما بيانات البرنامج فيمكن أن تشغل أكثر من جزء واحد ، ولذلك فإن التعامل مع البيانات في هذه الحالة يكون على أساس أنها بعيدة ويكتب في 32 بت (segment : offset) بالرغم من أن البيانات هنا تشغل أكثر من جزء إلا أنه غير مسموح فى هذا الموديل أن يكون لمصفوفة واحدة array أن تخرج خارج حدود هذا الجزء ، أي أن أي مصفوفة لا تتعدى 64 ك ب .

4- الموديل large

هنا يمكن لكود البرنامج وبياناته أن يشغل كل منهما أكثر من جزء واحد ، وسيكون التعامل مع العناوين هنا على الأساس البعيد سواء في حالة كود البرنامج أو بياناته . هنا أيضا يجب أن لا يتعدى حجم أي مصفوفة 64 ك.ب .

5- الموديل huge

وهو يشبه تماما الموديل large في الوقت الحالي .

معظم البرامج التي نتعامل معها سواء في هذا المقرر أو في الكثير من التطبيقات الأخرى ، يكون الموديل small مناسبا جدا لها حيث أن البرنامج يكون مخصصا له 64 ك.ب ، وكذلك 64 ك.ب للبيانات ، وهذا يعتبر كافي جدا لهذه التطبيقات الحالية . ويجب أن نستخدم هذا الموديل كلما أمكن إلا إذا كانت هناك ضرورة لغير ذلك لأن العنونة البعيدة الموجودة في الموديلات 3 ، 4 ، 5 تأخذ وقتا أطول في التنفيذ . وأخيرا يجب أن يوضع الأمر MODEL. قبل أوامر تحديد الأجزاء المختلفة stack. و data. و code.

3- أمر التوجيه stack.

هذا الأمر يحدد كمية الذاكرة التي سيستخدمها البرنامج كمكدسة . والمكدسة يخزن فيها البرنامج عناوين الرجوع عند النداء على البرامج الفرعية ، أو تنفيذ

برامج خدمة المقاطعة . إن 200 كلمة تعتبر مناسبة جدا كمكدسة في الكثير من الأغراض ، حيث يتم تحديد هذه الكمية كما في الأمر التالي :

```
.stack 200h
```

4- أمر التوجيه .code

هذا الأمر يحدد بداية الجزء الذي سنكتب فيه شفرات أو كود البرنامج . لاحظ أن هذا الأمر ليس له معاملات تكتب بعده كما كان في الأمر .stack 200h. ولكن هذا الجزء يبدأ بالأمر .code وينتهي بالأمر END ، وكمثال على ذلك ما يلي :

```
----------
 ----------
  .code
  add ax,bx
  sub ax,bx
  mov cx,100
-------------
-------------
END
```

5- أمر التوجيه .DATA

هذا الأمر يحدد بداية جزء البيانات المستخدمة في البرنامج كما يلي :

```
--------------
--------------
.DATA
boundary    DW 100
counter     DW 2
message     DW ' ** ERROR MESSAGE ** ' , '$'
-------------
------------
```

6- أمر التوجيه END

بهذا الأمر تتحدد نهاية البرنامج ، وبدون هذا الأمر يعطي الأسمبلر رسالة خطأ ، لأنه من الضروري أن ينتهي البرنامج بهذا الأمر .

ملحوظة : إن نسيان بعض أوامر التوجيه السابقة يسبب خطأ ، وبعضها يسبب تحذير ، لذلك نؤكد على ضرورة الالتزام بها .

شكل (15-2) يبين الصورة العامة لبرنامج أسمبلى وقد احتوى كل أوامر التوجيه السابقة .

```
DOSSEG
.MODEL  SMALL
.STACK  100H
.DATA
 DB
 DW
 ------------
 --------------
.Code
   Your program


END
```

شكل (15-2) الصورة العامة لبرنامج الأسمبلى .

15-4 أوامر لغة التجميع

لغة التجميع للمعالج 8088/8086 تحتوى العديد من الأوامر بحيث أنه سيكون من الصعب ومن الممل جدا أن ندرس هذه الأوامر عن طريق سردها الواحد بعد الآخر إلى أن نصل إلى نهايتها بحيث عندما نصل إلى النهاية نكون قد نسينا ما درسناه فى البداية . لذلك فقد اخترنا أن نقسم هذه الأوامر إلى مجموعات كما فعلنا عند دراسة لغة الأسمبلى للمعالجات السابقة بحيث ندرس كل مجموعة على حدة مع إعطاء بعض الأمثلة السريعة والتمارين على كل مجموعة ، معتمدين على أن الدارس لديه الخبرة الآن بمعظم أساسيات البرمجة بهذه اللغة .

15-5 مجموعة أوامر الانتقال
Transfer instructions

هذه المجموعة من الأوامر خاصة بنقل البيانات من مكان لآخر دون إجراء أى تعديل أو تغيير عليها . الصورة العامة لهذه الأوامر هى :

mov destination, source

حيث الكلمة mov هى اختصار move بمعنى أنقل أو حرك ، وأما source فهو مصدر المعلومة ، و destination هو الهدف أو الملجأ الذى ستذهب إليه المعلومة . أى أن المعلومة ستنتقل من المصدر إلى الهدف . كل من المصدر

والهدف من الممكن أن يكون مسجلا من مسجلات المعالج أو عنوان من عناوين الذاكرة . كما يمكن أن يكون المصدر معلومة فورية immediate أو ثابت .
من أمثلة نقل البيانات بين المسجلات المختلفة ما يلي :

```
mov al,bl ;      نقل محتويات المسجل bl (8بت) إلى المسجل al(8بت)
mov ax,cx ;  نقل محتويات المسجل cx (16بت) إلى المسجل ax (16بت)
mov bp,sp ;  نقل محتويات المسجل sp (16بت) إلى المسجل bp (16بت)
mov ds, ax
mov di,si
mov bx,es
mov cs,ds ;  هذا الأمر خطأ لأنه لا يمكن نقل محتويات مسجل مقطع إلى
                                                    ;  إلى مسجل مقطع آخر
mov bl,ax ;  هذا الأمر خطأ لأن المسجلين أحدهما 16 بت والآخر 8 بت
```

لغة التجميع ليست حساسة لشكل الحرف كما رأينا بحيث أنه يمكننا الكتابة بالأحرف الكبيرة أو الصغيرة ، فليس هناك فرق بين الأمر MOV AX,CX والأمر mov ax,cx ، كليهما يفهمه المجمع ويترجمه .
جميع الأوامر السابقة كانت تتعامل مع مسجلات فقط سواء كمصدر للمعلومة أو هدف ستذهب إليه المعلومة . هذا هو ما يسمى أحيانا بعنونة المسجلات register addressing حيث لا يكون هناك تعامل مع الذاكرة فى طرفي الأمر ، فقط مسجلات . يمكن تحميل أى مسجل بمعلومة فورية immediate data 8 بت أو 16 بت كما فى الأوامر التالية :

```
mov al,03h
```

هذا الأمر يحمل النصف الأول من مسجل التراكم (8 بت) بالمعلومة الفورية 03h (8 بت) ، حيث الحرف h يعنى أن هذه المعلومة مكتوبة فى النظام الستعشرى .

```
mov ax,0ff35h
```

حيث هنا تم تحميل المسجل ax (16بت) بالمعلومة ff35h (16بت) . عادة يوضع صفر قبل أى رقم ستعشرى يبدأ بحرف كما فى المثال السابق .
من المفيد جدا فى الكثير من البرامج أن نرمز لقيمة فورية بأي رمز ثم نستخدم هذا الرمز فى البرنامج بدلا من القيمة الثابتة ، كما فى الأوامر التالية :

```
kkk  equ  33h
.....
mov al,kkk
```

equ هو أمر توجيه جديد موجه للأسمبلر بإعطاء القيمة 33h للرمز kkk حيث يقوم الأسمبلر باستبدال الرمز kkk بقيمته عند كل موضع يظهر فيه هذا الرمز فى البرنامج .

مثال 15-2

أكتب برنامج يحمل المسجلات ah, al, bh, bl, ch, cl, dh بالقيم 00, 01, 02, 03, 04, 05, 06 على الترتيب ثم يقوم بعمل إزاحة دورانية على محتويات هذه المسجلات . هذا المثال تم تناوله مع كل المعالجات 8 بت ولذلك سنقدم البرنامج مباشرة كالتالى :

```
dosseg
.model small
.stack 100h
.data
.code
mov ah,00
mov al,01h
mov bh,02h
mov bl,03h
mov ch,04h
mov cl,05h
mov dh,06h
mov  dl,dh
mov  dh,cl
mov cl,ch
mov  ch,bl
mov  bl,bh
mov  bh,al
mov  al,ah
mov  ah,dl
end
```

بعد كتابة هذا البرنامج سجله فى ملف اسمه example1.asm وبعد ذلك استدعى الأسمبلر وأدخل عليه البرنامج باستخدام الأمر :

```
Tasm example1.asm
```

للحصول على برنامج الهدف example1.obj . بعد ذلك استدعى برنامج التوصيل tlinker للحصول على الصورة القابلة للتنفيذ للبرنامج كما يلي :

```
tlink example1.obj
```

بالحصول على الصورة القابلة للتنفيذ من البرنامج يمكنك تنفيذه باستخدام الأمر :
example1
حيث سينفذ البرنامج ويرجع الحاسب إلى dos دون أن ترى نتيجة محسوسة للبرنامج لأن مثل هذا البرنامج لا يطبع شيئا على الشاشة ولا يرسل نتائج إلى الطابعة ، لذلك فلن تحس به لأنه فقط يغير من محتويات المسجلات داخل المعالج . فى مثل هذه الظروف يلعب الديبجر دورا مهما فى أنه يمكننا به أن نرى نتيجة تنفيذ البرنامج فى المسجلات ، حيث يمكن باستخدام الديبجر أن نفحص كل مسجلات المعالج لنرى محتوياتها بعد تنفيذ البرنامج لنعرف هل تم تنفيذ البرنامج بالطريقة المطلوبة أم لا . بل إنه من مزايا استخدام الديبجر أنه يمكننا تنفيذ البرنامج خطوة بخطوة لنرى نتيجة البرنامج بعد تنفيذ كل أمر ونفحص عند أى لحظة لنعرف هل البرنامج يسير على ما يرام أم لا . وهذه فى الحقيقة تعتبر فائدة عظيمة فى استخراج الأخطاء من البرامج . لذلك سنعرض فى الجزء القادم لكيفية الدخول فى البرنامج من الديبجر واستخدامه لتتبع تنفيذ البرنامج .

15-6 الديباجر Debugger

لكي تدخل فى الديبجر لابد وأن يكون لديك الصورة القابلة للتنفيذ من البرنامج الذى تريد استخراج أخطاؤه أو التعامل معه ، لذلك يمكننا الدخول فى الديبجر بالأمر التالي :

```
C:\TASM>debug example1.exe
```

بذلك تدخل فى الديبجر وتظهر لك علامة وجودك فيه حيث يصبح دليل الكتابة Cursor هو الشكل '-' ويمكنك استخدام أوامره كالتالي :

15-6-1 إظهار محتويات المسجلات

بكتابة الحرف R (أو r لأن لغة الأسمبلى ليست حساسة لشكل الحرف) ثم enter تظهر أمامك جميع المسجلات بمحتوياتها كما يلى :

```
-r
AX=0000 BX=0000 CX=001E DX=0000 SP=0100 BP=0000 SI=0000 DI=000
DS=272C ES=272C SS=273E CS=273C IP=0000 NV UP EI PL NZ NA PO NC
```

حيث نرى أن محتويات المسجل AX=0000 والمسجل CS=273C وهكذا .
يمكنك إظهار محتويات مسجل معين بكتابة اسم المسجل بعد الحرف R حيث تظهر لك محتويات هذا المسجل فقط وفى السطر التالي تظهر العلامة ':' والتى

تتيح لك تغيير محتويات هذا المسجل بكتابة المحتويات الجديدة بعد هذه العلامة ، وإذا لم تريد تغيير هذه المحتويات اضرب enter .
يظهر فى آخر قائمة المسجلات بيان بحالة جميع الأعلام flags الموجودة فى المعالج وحالة كل علم إذا كان واحد أم صفر . جدول (15-1) يبين قائمة بهذه الأعلام وماذا يكتب فيها إذا كانت صفرا وماذا يكتب فيها إذا كانت واحد . سندرس معنى هذه الأعلام بالتفصيل عند دراستنا للقفز المشروط .

اسم العلم	العلم مرفوع set to one	العلم غير مرفوع set to zero
Over flow	OV	NV
Direction	DN	UP
Interrupt	EI	DI
Sign	NG	PL
Zero	ZR	NZ
Auxiliary	AC	NA
Parity	PE	PO
Carry	CY	NC

جدول (15-1) بيان بحالة الأعلام التى يظهرها الديبجر

15-6-2 عرض محتويات الذاكرة باستخدام الأمر Un

تحتوى الذاكرة الشفرات الثنائية لأوامر البرنامج ، وعرض هذه الشفرات الثنائية بنفس حالتها لا يفيد شا حيث يكون من الصعب فهمها . لذلك فقد أتاح الديبجر إمكانية عرض هذه الأوامر بشفرات الأسمبلى عن طريق كتابة الأمر Un والذى يعنى عرض الأوامر ابتداء من الأمر n نسبة للمسجل CS كما يلى:

```
C:\TASM>debug example1.exe
-r
AX=0000 BX=0000 CX=001E DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=1898 ES=1898 SS=18AA CS=18A8 IP=0000 NV UP EI PL NZ NA PO NC
18A8:0000 B400        MOV  AH,00
-u0
18A8:0000 B400        MOV    AH,00
18A8:0002 B001        MOV    AL,01
18A8:0004 B702        MOV    BH,02
18A8:0006 B303        MOV    BL,03
18A8:0008 B504        MOV    CH,04
18A8:000A B105        MOV    CL,05
18A8:000C B606        MOV    DH,06
18A8:000E 8AD6        MOV    DL, DH
18A8:0010 8AF1        MOV    DH, CL
```

```
18A8:0012 8ACD      MOV   CL, CH
18A8:0014 8AEB      MOV   CH, BL
18A8:0016 8ADF      MOV   BL, BH
18A8:0018 8AF8      MOV   BH, AL
18A8:001A 8AC4      MOV   AL, AH
18A8:001C 8AE2      MOV   AH, DL
18A8:001E 31E4      XOR   SP, SP
-
```

نلاحظ فيما سبق أن محتويات المسجل CS=18A8 ، لذلك تم عرض الأوامر ابتداء من العنوان 0000 نسبة لمحتويات هذا المسجل . بعد العنوان مباشرة ستجد شفرة الأمر الستعشرية ، فمثلا الأمر MOV AH,00 كانت شفرته B400 والأمر MOV AH,DL شفرته هى 8AE2 .

15-6-3 عرض محتويات جزء من الذاكرة بالشفرات الستعشرية بالأمر Dn

يمكن عرض محتويات جزء معين من الذاكرة بالأمر Dn حيث n هى عنوان البداية التى سيبدأ من عندها عرض المحتويات ، وعنوان البداية يكون هو العنوان الموجود فى المسجل DS ، كما يلى :

```
C:\TASM>debug example2.exe
-r
AX=0000 BX=0000 CX=001E DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=1898 ES=1898 SS=18AA CS=18A8 IP=0000  NVUP EI PL NZ NA PO NC
18A8:0000 B400        MOV   AH,00
-D0
1898:0000 CD 20 00 A0 00 9A F0 FE-1D F0 4F 03 FD 11 8A  03.....
1898:0010 FD 11 17 03 FD 11 EC 11-01 01 01 00 02 FF FF FF  .....
1898:0020 FF FF FF FF FF FF FF FF-FF FF FF FF 83 18 4C 01
1898:0030  98 18-FF FF FF FF 00 00 00 00
-
```

لاحظ أن محتويات المسجل DS=1898 وتم عرض محتويات أماكن الذاكرة منسوبة لهذا العنوان وكل سطر يبين محتويات 16عنوان (10 ستعشرى) .

15-6-4 تنفيذ البرنامج حتى عنوان معين Ga

هذا الأمر يبدأ فى تنفيذ البرنامج ابتدأ من العنوان المحدد بمسجل التجزيء CS ومحتويات مؤشر الأوامر IP . أى عنوان بداية التنفيذ سيكون CS:IP . سيقف التنفيذ عند العنوان a الموجود بعد الحرف G بحيث لن يتم تنفيذ الأمر الموجود

عند هذا العنوان . لذلك لتنفيذ البرنامج السابق من بدايته لابد من تصفير المسجل IP أولا باستخدام الأمر RIP – ثم تحميل المسجل IP بأصفار . بذلك سيبدأ التنفيذ من العنوان 1898:0000 . بعد ذلك نعطيه أمر التنفيذ G001E – حيث سيتوقف التنفيذ عند هذا الأمر الذى لا يدخل ضمن أوامر البرنامج ولذلك فإنه لن ينفذ .

15-6-5 متابعة تنفيذ البرنامج عن طريق تنفيذ عدد n من الخطوات Tn

يمكن متابعة Trace, T تنفيذ البرنامج لاستخراج أخطاء التنفيذ عن طريق تنفيذه خطوة بخطوة باستخدام الأمر Tn حيث n هى عدد الأوامر المطلوب تنفيذها . فمثلا T1 ستنفذ خطوة واحدة من البرنامج ، وهكذا . تذكر هنا أيضا أن مؤشر الأوامر IP لابد أن يحتوى عنوان الخطوة المراد تنفيذها منسوبا لمسجل التجزيء CS . بعد تنفيذ كل خطوة يظهر الديبجر محتويات جميع المسجلات وكذلك الأمر التالي فى التنفيذ كما يلي :

```
C:\TASM>debug example2.exe
r-
AX=0000 BX=0000 CX=001E DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=1898 ES=1898 SS=18AA CS=18A8 IP=0000  NVUP EI PL NZ NA PO NC
18A8:0000 B400        MOV    AH,00
t1-
AX=0000 BX=0000 CX=001E DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=1898 ES=1898 SS=18AA CS=18A8 IP=0002  NVUP EI PL NZ NA PO NC
18A8:0002 B001        MOV    AL,01
t1-
AX=0001 BX=0000 CX=001E DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=1898 ES=1898 SS=18AA CS=18A8 IP=0004  NVUP EI PL NZ NA PO NC
18A8:0004 B702        MOV    BH,02
t2-
AX=0001 BX=0200 CX=001E DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=1898 ES=1898 SS=18AA CS=18A8 IP=0006 NVUP EI PL NZ NA PO NC
18A8:0006 B303        MOV    BL,03

AX=0001 BX=0203 CX=001E DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
DS=1898 ES=1898 SS=18AA CS=18A8 IP=0008 NVUP EI PL NZ NA PO NC
18A8:0008 B504        MOV    CH,04
```

15-6-6 تغيير محتويات عنوان فى الذاكرة Ea

فى الكثير من الأحيان يتطلب تنفيذ البرنامج وضع قيمة معينة فى عنوان محدد فى الذاكرة . يتم ذلك بالأمر Ea حيث a هى عنوان البايت المراد تغيير

محتوياتها ، حيث بعد كتابة هذا الأمر يعرض الديبجر محتويات هذا العنوان الموجودة فعليا وينتظر منك تغيير هذه القيمة .

15-6-7 الخروج من الديبجر Q

بكتابة الحرف Q ثم enter تخرج من الديبجر إلى دوس .

حاول كتابة مثال 1 الخاص بالإزاحة الدورانية لمحتويات المسجلات وجرب عليه كل أوامر الديبجر .

15-7 تمارين

1. أشرح ما هو المقصود بالعنونة الضمنية ؟
2. ما هو المقصود بعنونة المسجلات ؟
3. أكتب باختصار عن الديبجر ؟ وماذا تفعل لكي تتبع تنفيذ البرنامج خطوة بخطوة ؟
4. أكتب برنامج يحمل المسجلين AL , AH بأي بيانات ثم يقوم البرنامج باستبدال محتويات هذين المسجلين دون فقد محتويات أى مسجل منهما ؟
5. أعد التمرين السابق ولكن على مسجلين 16 بت ، AX , BX مثلا ؟
6. أعد برنامج الإزاحة الدورانية فى مثال 1 ولكن هذه المرة اجعل الإزاحة تكون ناحية اليسار بدلا من ناحية اليمين كما كان فى المثال ؟
7. تتبع تنفيذ البرنامج التالي مع كتابة محتويات المسجلات بعد تنفيذ كل خطوة باعتبار أن جميع المسجلات تحتوى أصفار فى بداية البرنامج :

	AH	AL	BH	BL	CH	CL	DH	DL
	00	00	00	00	00	00	00	00
MOV AL,05	--	--	--	--	--	--	--	--
MOV AH,01	--	--	--	--	--	--	--	--
MOV BX,AX	--	--	--	--	--	--	--	--
MOV CL,BH	--	--	--	--	--	--	--	--
MOV CH,BL	--	--	--	--	--	--	--	--
MOV DX,CX	--	--	--	--	--	--	--	--

15-8 أوامر القفز
Jump Instructions

القاعدة العامة أن المعالج يقوم بتنفيذ البرنامج حسب ترتيب الأوامر من أول البرنامج حتى نهايته ، ولقد راعينا ذلك فى الأمثلة السابقة . ولكن هناك بعض التطبيقات التى تتطلب الخروج على هذه القاعدة ، كأن يطلب منك مثلا تنفيذ عملية معينة أو مجموعة من الأوامر عدد معين من المرات أو حتى عدد لا نهائي من المرات . لقد أتاح المعالج ذلك بتوفير بعض الأوامر التى تمكنك كمبرمج من القفز بعملية التنفيذ من مكان لآخر خلال البرنامج . عادة تنقسم أوامر القفز إلى نوعين كالتالي :

15-8-1 القفز غير المشروط Unconditional jump

عند تنفيذ أوامر القفز غير المشروط ينتقل المعالج بعملية التنفيذ إلى المكان الجديد والمحدد دون قيد أو شرط . هذا المكان الذى سيتم القفز إليه يكون محددا بعلامة معينة label حيث تستخدم هذه العلامة فى أمر القفز . هناك أمر وحيد للقفز غير المشروط وهو الأمر :

```
jmp label
```

كمثال على ذلك ما يلى :

```
again: mov ax,05H
       mov bx,ax
       mov cx,bx
      jmp again
```

حيث العلامة again تم استخدامها قبل الأمر mov ax,05H المراد القفز إليه ، كما تم استخدامها أيضا فى أمر القفز نفسه . من شروط العلامات المستخدمة أنها لابد أن تبدأ بحرف أبجدي ومن الممكن أن تحتوى أرقاما ومن الممكن أن يصل عدد حروف العلامة إلى 31 حرفا . يجب أيضا ألا تحتوى العلامة على مسافات ، ويجب أن تنتهي بالحرف ":" كدليل يبين نهاية العلامة . كذلك لابد من وجود مسافة بين نهاية العلامة ":" وبداية الأمر .

15-8-2 القفز المشروط Conditional jump

كما يوحي الاسم فإن هذا النوع من القفز لن يتم إلا إذا تحقق شرطا معينا ، إذا لم يتحقق هذا الشرط فإن القفز لن يتم وسيستمر البرنامج فى مساره الطبيعي حيث ينفذ الأمر التالي لأمر القفز . إن شروط القفز توضع دائما على الأعلام ، فهناك

قفز مثلا إذا كانت النتيجة تساوى صفرا ، أى أن علم الصفر يساوى واحد ، كما أن هناك قفزا إذا كان هناك حملا فى آخر عملية قام بها المعالج ، وهكذا . جدول (15-2) يبين معظم أوامر القفز الشهيرة والكثيرة الاستخدام مع المعالج 8086/8088 . هناك أيضا أوامر قفز مشروطة بحالة معينة لأكثر من علم مثل الأمر JA والذى يعنى اقفز إذا كانت النتيجة أكبر من الصفر ، وتكون النتيجة أكبر من الصفر إذا كان علم الصفر يساوى صفرا وعلم الإشارة يساوى صفرا أيضا .

هشـــــف الألـــــــــــد	آلـــد القفز
اقفز إذا كانت النتيجة فوق الصفر jump if above	JA
اقفز إذا كانت النتيجة فوق الصفر أو تساوى صفر	JAE
اقفز إذا كانت النتيجة تحت الصفر jump if below	JB
اقفز إذا كانت النتيجة تحت الصفر أو تساويه	JBE
اقفز إذا كانت النتيجة تساوى الصفر jump if equal	JE/JZ
اقفز إذا لم يكن هناك حمل jump if no carry	JNC
اقفز إذا كانت النتيجة لا تساوى الصفر	JNE
اقفز إذا لم يكن هناك فيضان في النتيجة	JNO
اقفز إذا كانت الباريتى فردية jump if parity odd	JPO
اقفز إذا كانت النتيجة موجبة jump if no sign	JNS
اقفز إذا كان هناك فيضان فى النتيجة	JO
اقفز إذا كانت الباريتى زوجية jump if parity even	JPE
اقفز إذا كانت النتيجة سالبة jump if sign	JS
اقفز إذا كانت النتيجة أكبر من الصفر jump if greater	JG
اقفز إذا كانت النتيجة أكبر من الصفر أو تساويه	JGE
اقفز إذا كانت النتيجة أقل من الصفر jump if less	JL
اقفز إذا كانت النتيجة أقل من أو تساوى	JLE
اقفز إذا كان المسجل CX=0 يساوى صفر	JCXZ

جدول (15-2) أوامر القفز المشروط

مثال 15-3

أكتب برنامجا يقارن محتويات المسجل AX والمسجل BX بحيث إذا كانا متساويان يضع 1 فى المسجل DX ، أما إذا كانا غير متساويين فيضع صفر فى المسجل DX هذا مع الحفاظ على محتويات كل من المسجلين AX و BX .

أحد الاقتراحات لهذا البرنامج هو البرنامج التالي مع العلم أنه من الممكن أن يكون هناك أكثر من حل بالذات بعد أن ندرس باقي أوامر الحساب .

```
        MOV CX,AX
        SUB AX,BX
        JZ HERE1
        MOV DX,0
        JMP HERE2
HERE1: MOV DX,1
HERE2: MOV AX,CX
```

نلاحظ أن هذا البرنامج قد احتفظ بمحتويات المسجل AX فى المسجل CX كما فى الأمر الأول ، بعد ذلك طرح محتويات المسجل BX من المسجل AX حيث ستوضع النتيجة فى المسجل AX ، لذلك فإن محتويات المسجل AX ستفقد ولذلك فقد احتفظنا بها فى المسجل CX . بعد ذلك إذا كانت نتيجة الطرح صفرا فإن المسجلين متساويين وسيضع البرنامج 1 فى المسجل DX بعد أن يقفز إلى العلامة HERE1 . أما إذا كان المسجلين غير متساويين فسيضع البرنامج صفر فى المسجل DX ثم يقفز إلى العلامة HERE2 ليسترد محتويات المسجل AX من المسجل CX .

15-8-3 الأمر LOOP

من الأوامر الكثيرة الاستخدام لعمل الحلقات الأمر LOOP والذى ينفذ مجموعة الأوامر المحصورة بينه وبين العلامة المذكورة فيه عدد من المرات يساوى محتويات المسجل CX . كمثال على ذلك انظر إلى ما يلي :

```
        MOV CX,0010H
HERE:  MOV AX,00H
        ……..
        …….
        LOOP HERE
```

حيث سينفذ هذا البرنامج مجموعة الأوامر الموجودة بين الأمر LOOP والعلامة HERE عدد 16 (10H) مرة حيث هذا العدد مخزن فى المسجل CX قبل الدخول فى الحلقة .

15-9 أول خطوات التعامل مع الذاكرة
First step to memory addressing

هناك طريقتان للتعامل مع الذاكرة وهما كما يلي :

15-9-1 الطريقة المباشرة Direct addressing

فى هذه الطريقة يوجد عنوان الذاكرة المراد التعامل معه فى الأمر نفسه مباشرة ، ولذلك سميت هذه الطريقة بالطريقة المباشرة للتعامل مع الذاكرة .
كمثال على ذلك الأوامر التالية :

MOV al,[2000H]

وهذا الأمر يعنى نقل محتويات البايت أو العنوان 2000H فى الذاكرة إلى المسجل al .

MOV AX,[2000H]

والذى يعنى نقل محتويات العنوان 2000H والذى يليه إلى المسجل AX .

MOV [31F2H],AL

والذى يعنى نقل محتويات المسجل AL إلى البايت التى عنوانها 31F2H .
نلاحظ أنه فى كل هذه الأوامر ظهر العنوان مباشرة فى الأمر نفسه ، ونلاحظ أيضا أن العنوان تم وضعه بين القوسين المربعين [ ] .
من الممكن أن يرمز للعنوان برمز معين كما فى الأمر التالي :

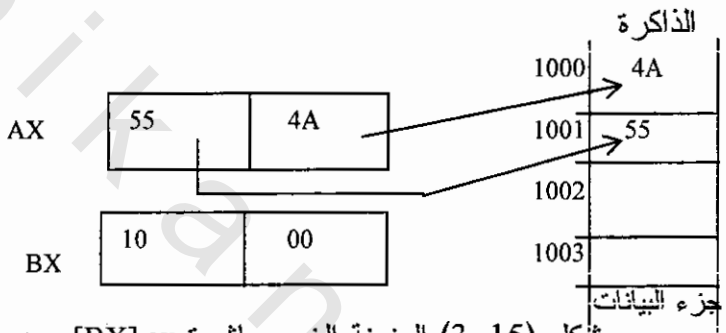
MOV AL,table

والذى يعنى نقل محتويات البايت المسماة بالاسم table إلى المسجل AL . إن جميع العناوين المباشرة تكون دائما محددة فى مقطع أو جزء البيانات . أى أن العنوان 2000H مثلا يحسب ابتداء من العنوان الموجود فى مسجل التجزيء D.

15-9-2 الطريقة غير المباشرة Indirect addressing

هذه الطريقة من العنونة كما ذكرنا فى الفصل الثاني تسمح بالتعامل مع بيانات موجودة فى الذاكرة حيث العنوان الذى سيتم التعامل معه فى هذه الحالة يكون موجودا فى أحد مسجلات المعالج التالية : BX, BP, SI, DI . كمثال على ذلك افترض أن المسجل BX يحتوى الرقم 1000H وطلبنا من المعالج تنفيذ الأمر التالي : MOV AX,[BX] . فى هذه الحالة سيقوم المعالج بإحضار نسخة من محتويات العنوان 1000H (والذى يليه) الموجود فى المسجل BX ويضعها فى المسجل AX . أى أن محتويات المسجل BX الموضوع بين قوسين مربعين كما رأينا تمثل عنوان المعلومة وليس المعلومة نفسها ، ففي عدم وجود القوسين سينسخ المعالج محتويات المسجل BX ويضعها فى المسجل AX كما رأينا فى

أول طرق العنونة (عنونة المسجل) . يجب أن نؤكد هنا أن العنوان الفعلي للمعلومة يحسب منسوبا لمحتويات مسجل التجزيء DS بعد إزاحته ناحية اليسار 4 بتات كما ذكرنا سالفا ، أى أنه إذا كانت محتويات المسجل DS=0100H فإن الأمر السابق سينسخ محتويات العنوان 01000+1000=02000H والذى يليه ويضعها فى المسجل AX . لاحظ أيضا أن المسجلات BX, SI, DI تعامل مع عناوين منسوبة إلى مسجل التجزيء DS بينما المسجل BP يعنون عناوين منسوبة لمسجل التجزيء SS . شكل (15-3) عبارة عن رسم توضيحي لعملية العنونة غير المباشرة للذاكرة . كأمثلة على هذا النوع من العنونة أيضا انظر إلى الأوامر التالية :



شكل (15-3) العنونة الغير مباشرة mov [BX],ax

```
MOV CX,[BX]
MOV [BP],BL
MOV [DI],AH
MOV [DI],[BX] (خطأ)
```

حيث الأمر الأول سينقل محتويات عنوان (والذى يليه) فى جزء البيانات أو ذاكرة البيانات data segment والمشار إليه بالمسجل BX إلى المسجل CX ، بينما الأمر الثاني سينقل محتويات النصف الأول من المسجل BX إلى عنوان مشار إليه بالمسجل BP ويقع فى جزء المكدسة . الأمر الثالث سينقل النصف العلوي من المسجل AX إلى عنوان مشار إليه بالمسجل DI ويقع فى جزء البيانات ، أما الأمر الثالث فغير مسموح به لأنه ينقل من ذاكرة إلى ذاكرة وهذا النوع من العنونة غير مسموح به إلا فى حالات خاصة جدا مع بعض أوامر سلاسل الحروف .

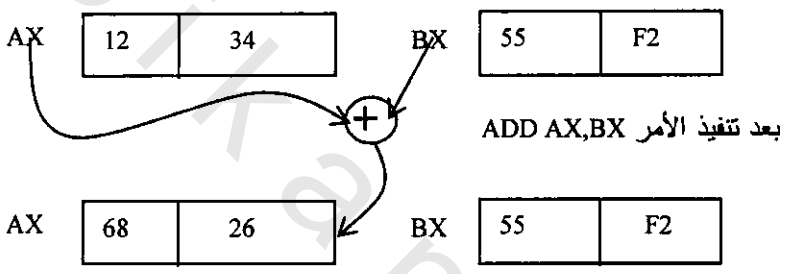
15-10 أوامر الحساب
Arithmetic Instructions

أوامر الحساب التى يمكن تنفيذها بلغة الأسمبلي هى الجمع والطرح والضرب والقسمة . كل هذه العمليات يمكن إجراؤها على بيانات 8 بت أو 16بت .

15-10-1 أمر الجمع ADD

الصورة العامة لهذا الأمر هى :

ADD destination, source



شكل (15-4) شكل توضيحي لأمر الجمع ADD

حيث فى هذا الأمر يتم جمع المصدر source مع الهدف destination وتوضع النتيجة فى الهدف . من أمثلة ذلك ما يلي :

ADD AX,BX

هذا الأمر يجمع محتويات المسجل BX مع محتويات المسجل AX ويضع النتيجة فى المسجل AX . لاحظ أن هذا الأمر يجمع مسجلين كل منهما 16بت . شكل (15-4) يبين رسما توضيحا لهذا الأمر .

أمثلة أخرى على هذا الأمر ما يلي :

ADD Al,CL

الذى يجمع محتويات النصف الأول من المسجل CX مع النصف الأول من المسجل AL وبضع النتيجة فى المسجل AL .

ADD AL,[BX]

الذى يجمع محتويات مكان الذاكرة (8بت) الذى عنوانه فى المسجل BX مع محتويات المسجل AL ويضع النتيجة فى المسجل AL .

ADD AX,0F437H

هذا الأمر يجمع الثابت أو القيمة الفورية F437H (16بت) مع المسجل AX ويضع النتيجة فى المسجل AX . فى العادة يتم وضع 0 قبل أى ثابت يبدأ بحرف وكذلك وضع الحرف H فى نهاية الثابت للدلالة على أنه فى النظام الستعشرى .

ADD table,20H

الذى يجمع الثابت 20H مع محتويات مكان الذاكرة المسمى table ويضع النتيجة فى نفس المكان . المكان المسمى table تتم تسميته بهذا الاسم فى مقطع البيانات فى بداية البرنامج باستخدام أمر التوجيه define . كمثال على ذلك الأمر التالى :

- table DB 22H

هذا الأمر التوجيهى يحجز بايت اسمها table ويعطيها القيمة الابتدائية 22 .

- table DB 22H, 25H, 'A', 33H

هذا الأمر يحجز عدد 4 بايت ويعطيها القيم الابتدائية السابقة ، وهذه الأربعة أماكن تأخذ الاسم table . هناك أيضا الأوامر DW الذى يحجز كلمة word ، والأمر DD الذى يحجز كلمتين ، والأمر DQ الذى يحجز 4 كلمات ، والأمر DT الذى يحجز 10 كلمات . فى جميع هذه الأوامر الاسم الملحق بالأمر يمثل عنوان أول بايت فى كمية الذاكرة المحجوزة .

من الأشياء المهمة التي يجب ألا تنسى هي أنه لا يمكن جمع مكان ذاكرة على مكان ذاكرة آخر . فمثلا الأمر التالي خطأ فى عرف لغة الأسمبلى :

ADD [BX],[3F20H]

لأنه يحاول جمع محتويات العنوان 3F20H مع محتويات العنوان الموجود فى المسجل BX وهذا خطأ أو غير مسموح .

15-10-2 أمر الجمع ADC

الصورة العامة لهذا الأمر هى :

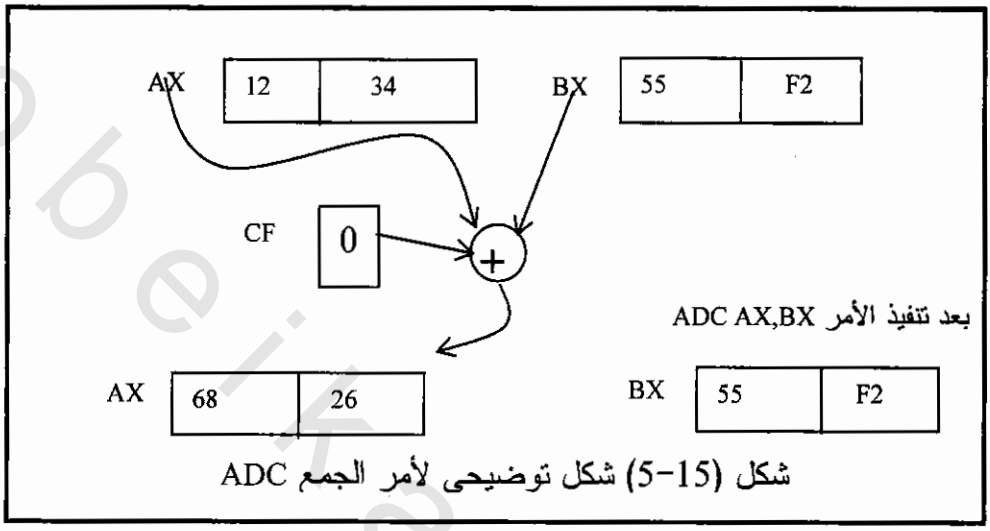
ADC destination, source

حيث فى هذا الأمر يتم جمع المصدر source مع الهدف destination مع محتويات علم الحمل carry flag والتى تكون صفرا أو واحد ، وتوضع النتيجة فى الهدف . من أمثلة ذلك ما يلي :

ADC AX,BX

هذا الأمر يجمع محتويات المسجل BX مع محتويات المسجل AX مع محتويات علم الحمل ويضع النتيجة فى المسجل AX . لاحظ أن هذا الأمر يجمع مسجلين كل منهما 16بت مع علم الحمل . شكل (15-5) يبين رسما توضيحا لهذا الأمر. أمثلة أخرى على هذا الأمر ما يلي :

ADC AL,CL
ADC BL, table
ADC CL,40H
ADC BX, [SI]

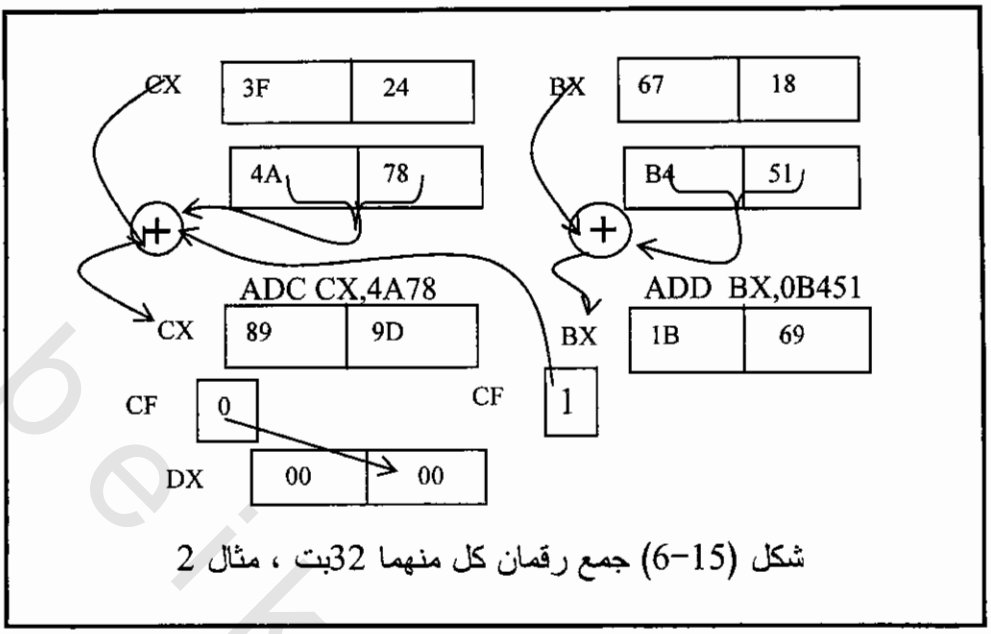


شكل (15-5) شكل توضيحى لأمر الجمع ADC

كل هذه الأوامر تجمع محتويات المصدر مع الهدف مع علم الحمل وتضع النتيجة فى الهدف . هنا يظهر سؤال مهم عن الفرق بين الأمرين ADD و ADC . لكي نفهم الفرق بينهما نسوق المثال التالي :

مثال 15-4

اكتب برنامجا يجمع الرقمين 3F246718 و 4A78B451 ويضع النتيجة فى المسجلات BX و CX و DX . نلاحظ أن كل من الرقمين مكون من 32بت ، وليس هناك وسيلة لجمع رقمين كل منهما 32بت مرة واحدة . لذلك سنضع الرقم الأول فى المسجلين BX وCX ثم نجمع النصف الأول من الرقم الثاني (B451) مع المسجل BX باستخدام الأمر ADD وبعد ذلك نجمع النصف الثاني من الرقم الثاني (4A78) مع المسجل CX باستخدام الأمر ADC حتى يتم أخذ علم الحمل فى الاعتبار لأنه قد يكون هناك حمل من عملية الجمع الأولى فيجب أخذه فى الاعتبار . شكل(15-6) يبين رسما توضيحيا لهذا المثال . لاحظ أن عملية الجمع الأولى يجب أن تتم باستخدام الأمر ADD وليس باستخدام الأمر ADC لأنه لو استخدم الأمر ADC فسيجمع علم الحمل من عملية سابقة مما سيؤثر على النتيجة ويجعلها خطأ .



شكل (15-6) جمع رقمان كل منهما 32بت ، مثال 2

البرنامج الذى سيقوم بعملية الجمع السابقة من الممكن أن يكون كالتالي :

```
ADD BX,0B451H
ADC CX,4A78H
MOV DX,00H
ADC DX,DX
```

15-10-3 أمر الطرح SUB

هناك أمران للطرح مثل الجمع ، أحدهما لا يأخذ علم الحمل فى الاعتبار والآخر يأخذ علم الحمل فى الاعتبار . الأمر SUB يطرح محتويات المصدر من محتويات الهدف ويضع النتيجة في الهدف ، فمثلا الأمر :

```
SUB AX,BX
```

سيطرح المسجل BX (المصدر) من المسجل AX (الهدف) ويضع النتيجة فى المسجل AX . نؤكد هنا على أن الهدف يكون هو دائما المطروح منه .
أمثلة أخرى على أوامر الطرح كالتالي :

```
SUB DL,CL    ;    DL-CL ⟶ DL
SUB AL,[BX]  ;    AL-[BX] ⟶ AL
SUB BL,20H   ;    BL-20 ⟶ BL
```

15-10-4 أمر الطرح SBB

هنا يتم طرح المصدر وعلم الحمل CF من الهدف وتوضع النتيجة فى الهدف .
من أمثلة ذلك ما يلي :

```
SBB AX,BX   ;    AX-BX-CF ⟶ AX
SBB DL,CL   ;    DL-CL-CF ⟶ DL
SBB AL,[BX] ;    AL-[BX]-CF ⟶ AL
SBB BL,20H  ;    BL-20-CF ⟶ BL
```

يجب أن نكون حذرين جدا فى الأماكن التى نستخدم فيها الأمر SUB والأماكن الأخرى التى يجب أن نستخدم فيها الأمر SBB لأن ذلك من الممكن أن يعطى نتيجة خاطئة كما أوضحنا مع أوامر الجمع .

15-10-5 أمر المقارنة CMP

من الأوامر الشهيرة الاستخدام فى الكثير من التطبيقات أمر المقارنة CMP الذى له الصورة العامة التالية :

CMP destination, source

حيث يتم طرح المصدر source من الهدف destination ولكن هنا لا يتم وضع النتيجة فى الهدف ولكن يبقى الهدف دون تغيير ، وهذا هو الاختلاف الأساسي بين هذا الأمر وأوامر الطرح السابقة . الاستفادة الوحيدة من تنفيذ هذا الأمر هى تأثر الأعلام بنتيجة هذا الطرح . بالطبع ما أكثر التطبيقات التى نحتاج فيها لمقارنة رقمين لمعرفة أيهما أكبر من الآخر مثلا دون تغيير قيمة أى واحد من الرقمين حيث فى هذه الحالة فإن أوامر الطرح السابقة لا تؤدي هذا الغرض مباشرة ، لذلك فى هذه الأحوال نستخدم الأمر CMP .

15-10-6 الأوامر INC و DEC

تعمل هذه الأوامر على زيادة محتويات المصدر أو إنقاصها بمقدار واحد .
الصورة العامة لهذين الأمرين هى كالتالي :

INC source
DEC source

ومن أمثلة ذلك ما يلي :

```
INC BL
INC CX
INC [SI]
DEC DL
```

DEC [BX]

وجميعها تجمع 1 أو تطرح 1 من محتويات المصدر الموجود فى الأمر سواء كان مسجل (8 أو 16 بت) أو بايت ذاكرة .

مثال 15-5

أكتب برنامجا يقوم بضرب محتويات المسجلين CL و BL ويضع النتيجة فى المسجل AX وذلك عن طريق الجمع المتكرر .

```
        MOV AL,00
        MOV AL,CL
        DCR BL
YY:     ADD AL,CL
        JNC XX
        INC AH
XX:     DCR BL
        JNZ YY
        HLT
```

مثال 15-6

أكتب برنامجا يقسم محتويات المسجل AL على محتويات المسجل CL ويضع النتيجة فى المسجل AH والباقي فى المسجل DL وذلك عن طريق الطرح المتكرر .

```
        MOV AH,00
        CMP AL,CL
        JS XX ;         CL>AL put AL into DL as a remainder
              ;         and stop
YY:     SUB AL,CL
        JS XX1
        INC AH
        JMP YY
XX1:    ADD AL,CL
XX:     MOV DL,AL ;   put remainder in DL.
        HLT
```

15-10-7 أمر الضرب MUL

الصورة العامة لهذا الأمر هى :

MUL source

حيث يقوم هذا الأمر بضرب المصدر source الذى يكون مسجلا (8 بت أو 16 بت) أو بايت ذاكرة فى المسجل AL وذلك إذا كان المصدر 8 بت ، أما إذا كان المصدر 16 بت فإنه يضرب المصدر فى المسجل AX . أى أن الطرف الثاني لعملية الضرب يكون دائما إما المسجل AL أو المسجل AX على حسب عدد بتات المصدر . إذا كان المصدر 8 بت فإن الطرف الثاني لعملية الضرب يكون المسجل AL كما ذكرنا وتوضع النتيجة فى المسجلين AL و AH حيث يحتوي AL النصف الأدنى من النتيجة ويحتوى المسجل AH النصف الأعلى منها . إذا كان المصدر 16 بت فإن الطرف الثاني لعملية الضرب يكون المسجل AX كما ذكرنا ، ولكن النتيجة فى هذه المرة توضع فى المسجلين AX (الذى يحتوي الجزء الأدنى من النتيجة) و DX الذى يحتوى الجزء الأعلى منها . بالطبع فإن جميع الأعلام تتأثر بهذه العملية . من أمثلة ذلك ما يلي :

MUL BL

الذى يضرب محتويات المسجل BL فى المسجل AL ويضع النتيجة فى المسجل AX .

MUL CX

الذى يضرب محتويات المسجلين CX و AX ويضع النتيجة فى المسجلين AX و DX .

MUL [BX]

الذى يضرب محتويات بايت الذاكرة المشار إليها بالعنوان الموجود فى المسجل BX فى المسجل AL ويضع النتيجة فى المسجل AX .

من الأشياء المهمة التى يجب أن نحذرها أو نتجنبها هى ضرب قيمة فورية (ثابت) فى المسجل AL أو AX على الصورة التالية :

MUL 05
MUL 3f24

هذا الشكل لأمر الضرب ممنوع

فى هذه الحالة يجب أن نضع الثابت فى أى مسجل أولا ثم نستخدم هذا المسجل فى عملية الضرب .

15-10-8 أمر القسمة DIV

الصورة العامة لهذا الأمر هى :

DIV source

حيث يقوم هذا الأمر بقسمة المسجل AX أو المسجلين AX و DX على المصدر source الذى يكون مسجلا (8 بت أو 16 بت) أو بايت ذاكرة . أى أن المصدر يكون دائما هو المقسوم عليه . أما المقسوم فيتحدد على حسب المصدر كالتالي :

1. إذا كان المصدر 8 بت فإن المقسوم لابد أن تكون بتاته ضعف بتات المقسوم عليه ولذلك فإنه يكون المسجل AX فى هذه الحالة . وفى هذه الحالة يخزن خارج القسمة فى المسجل AL والباقي من عملية القسمة فى المسجل AH .
2. إذا كان المصدر 16 بت فإن المقسوم يكون المسجلين AX و DX حيث AX يحتوى النصف الأدنى و DX يحتوى النصف الأعلى للمقسوم . أما ناتج القسمة فإنه يوضع فى المسجل AX والباقي من القسمة فإنه يوضع فى المسجل DX .

من أمثلة عمليات القسمة ما يلي :

DIV BL

الذى يقسم محتويات المسجل AX على المسجل BL ويضع النتيجة فى المسجل AL والباقي فى المسجل AH .

DIV CX

الذى يقسم محتويات المسجلين AX و DX على المسجل CX ويضع النتيجة فى المسجل AX والباقي فى المسجل DX .

DIV [BX]

الذى يقسم محتويات المسجل AX على بايت الذاكرة المشار إليها بالعنوان الموجود فى المسجل BX ويضع النتيجة فى المسجل AL والباقي فى المسجل AH .

من الأشياء المهمة التى يجب أن نحذرها أو نتجنبها هى استخدام قيمة فورية (ثابت) فى عملية قسمة كالتالي :

DIV 05
DIV 3F24

هذا الشكل لأمر القسمة ممنوع

فى هذه الحالة يمكن أن يوضع الثابت فى أى مسجل ثم يستخدم هذا المسجل فى عملية القسمة .

مثال 15-7

أكتب برنامجا يحسب مضروب الرقم الموجود فى عنوان الذاكرة 1800H ويضع نتيجة هذا المضروب فى المسجل AX .

```
    MOV AX, 0001
    MOV BL,[1800]
    MOV BH,00
    MOV CX,BX
    DEC CX
XX: MUL BL
    DEC BL
    LOOP XX
    HLT
```

15-11 أوامر المنطق
Logic Instructions

هذه المجموعة خاصة بإجراء العمليات المنطقية ، ويقصد بالعملية المنطقية العملية التى معاملاتها هى الواحد أو الصفر . العمليات المنطقية التى يجريها المعالج 8088 هى عمليات AND, OR, XOR, NOT .

15-11-1 الأمر AND

الصورة العامة لهذا الأمر هى :

AND destination, source

حيث يتم إجراء عملية AND على كل من المصدر source والهدف destination وتوضع النتيجة فى الهدف destination . بالطبع فإن كل من المصدر والهدف لابد أن يكونا نفس الحجم أى لهما نفس العدد من البتات . من أمثلة ذلك ما يلي :

AND AL,BL

حيث يجرى عملية AND على محتويات المسجلين AL, BL ويضع النتيجة فى المسجل AL . فمثلا افترض أن المسجلين AL, BL يحتويان البيانات التالية :

BL = 10111101=BDH
AL = 11101110=EEH

فإنه بعد تنفيذ الأمر AND AL,BL ستكون محتوياتهما كالتالي :

BL = 10111101=BDH
AL = 10101100=ACH

محتويات BL المصدر لم تتغير

النتيجة وضعت فى الهدف

```
AND CX,DX
AND BH,table
AND CH,11011011B
```

حيث الأمر الأخير سيجرى عملية AND على المسجل CH والمعلومة الفورية أو الثابت 11011011B المعطى فى الصورة الثنائية والتى تم الإشارة إليها باستخدام الحرف B فى آخر الثابت وحيث تسمح لغة التجميع بذلك .

15-11-2 الأمر OR

الصورة العامة لهذا الأمر هى :

```
OR destination, source
```

حيث يتم إجراء عملية OR على كل من المصدر source والهدف destination وتوضع النتيجة فى الهدف destination . بالطبع فإن كل من المصدر والهدف لابد أن يكونا نفس الحجم أى لهما نفس العدد من البتات . من أمثلة ذلك ما يلي :

```
OR AL,BL
```

حيث يجرى عملية OR على محتويات المسجلين AL, BL ويضع النتيجة فى المسجل AL

```
OR CX,DX
OR BH,table
OR CH,11011011B
```

15-11-3 الأمر XOR

الصورة العامة لهذا الأمر هى :

```
XOR destination, source
```

حيث يتم إجراء عملية XOR على كل من المصدر والهدف وتوضع النتيجة فى الهدف . من أمثلة ذلك ما يلي :

```
XOR AL,BL
```

حيث يجرى عملية XOR على محتويات المسجلين AL, BL ويضع النتيجة فى المسجل AL .

```
XOR CX,DX
XOR BH,table
XOR CH,11011011B
```

15-11-4 أمر النفي المنطقي NOT

هذا الأمر له معامل واحد والصورة العامة له هى كالتالي :

```
NOT source
```

حيث يتم عكس المصدر عكسا منطقيا بجعل كل صفر واحد وكل واحد صفر وبالطبع فإن نتيجة هذا العكس توضع في نفس المصدر . من أمثلة ذلك ما يلى :

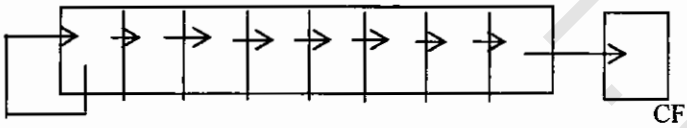
NOT AL
NOT DX
NOT [BX]

15-12 أوامر الإزاحة والدوران
Shift And Rotate Instructions

هناك نوعان من الإزاحة ، إزاحة حسابية وهي التي يتم الاحتفاظ فيها بإشارة الرقم الذى تجرى عليه الإزاحة وبالذات إذا كانت الإزاحة ناحية اليمين كما سنرى . أما الإزاحة المنطقية فلا اعتبار فيها للإشارة .

15-12-1 أمر الإزاحة الحسابية لليمين SAR

يتم هنا إزاحة محتويات كل خانة إلى الخانة التى على يمينها ، وأما الخانة التى يتم تفريغها فى أقصى اليسار فيتم ملأها دائما بمحتويات خانة الإشارة والتى هى نفس محتويات الخانة الأخيرة . أى أن محتويات آخر بت ناحية اليسار يتم الاحتفاظ بها دائما ولا تفرغ . أما محتويات البت فى أقصى اليمين فتذهب إلى علم الحمل . شكل (15-7) يبين رسما توضيحيا لهذا الأمر والصورة العامة له ستكون كالتالي :



شكل (15-7) الإزاحة الحسابية لليمين

SAR destination, count

حيث count إما أن تكون واحد إذا كانت الإزاحة ستتم مرة واحدة فقط وأما إذا أردنا الإزاحة أكثر من مرة فيوضع عدد المرات فى المسجل CL ثم يستخدم المسجل CL بدلا من count فى الصورة العامة للأمر . كمثال على ذلك نفترض أن المسجل AL = 10011010 بعد تنفيذ الأمر SAR AL,1 فإن محتويات المسجل AL وعلم الحمل ستكون كالتالي :

AL = 1 0 0 1 1 0 1 0

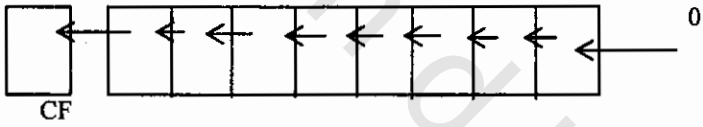
AL = 1 1 0 0 1 1 0 1 , CF = 0

15-12-2 أمر الإزاحة الحسابية لليسار SAL

يتم هنا إزاحة محتويات كل خانة إلى الخانة التى على يسارها ، وأما الخانة التى يتم تفريغها فى أقصى اليمين فيتم ملأها بأصفار . أما محتويات البت فى أقصى اليسار فتذهب إلى علم الحمل . شكل (15-8) يبين رسما توضيحيا لهذا الأمر والصورة العامة له ستكون كالتالي :

SAL destination, count

حيث count إما أن تكون واحد إذا كانت الإزاحة ستتم مرة واحدة فقط وأما إذا أردنا الإزاحة أكثر من مرة فيوضع عدد المرات فى المسجل CL ثم يستخدم المسجل CL بدلا من count فى الصورة العامة للأمر . كمثال على ذلك نفترض أن المسجل AL= 10011010 بعد تنفيذ الأمر SAL AL,1 فإن محتويات المسجل AL وعلم الحمل ستكون كالتالي :



شكل (15-8) الإزاحة الحسابية لليسار

AL = 1 0 0 1 1 0 1 0

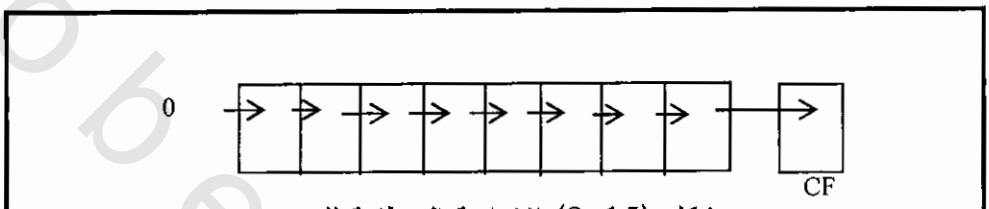
AL = 0 0 1 1 0 1 0 0 ← 0 CF = 1

15-12-3 أمر الإزاحة المنطقية لليمين SHR

يتم هنا إزاحة محتويات كل خانة إلى الخانة التى على يمينها ، وآخر خانة ناحية اليسار يتم ملأها دائما بصفر مع كل إزاحة . أما محتويات البت فى أقصى اليمين فتذهب إلى علم الحمل . شكل (15-9) يبين رسما توضيحيا لهذا الأمر والصورة العامة له ستكون كالتالي :

SHR destination, count

حيث count إما أن تكون واحد إذا كانت الإزاحة ستتم مرة واحدة فقط وأما إذا أردنا الإزاحة أكثر من مرة فيوضع عدد المرات فى المسجل CL ثم يستخدم المسجل CL بدلا من count فى الصورة العامة للأمر . كمثال على ذلك نفترض أن محتويات المسجل هى AL = 10011010 ، بعد تنفيذ الأمر SHR AL,1 فإن محتويات المسجل AL وعلم الحمل ستكون كالتالي :



شكل (15-9) الإزاحة المنطقية لليمين

AL = 1 0 0 1 1 0 1 0

AL = 0 1 0 0 1 1 0 1 , CF = 0

15-12-4 أمر الإزاحة المنطقية لليسار SHL

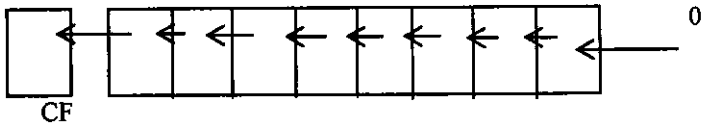
يتم هنا إزاحة محتويات كل خانة إلى الخانة التى على يسارها ، وأما الخانة التى يتم تفريغها فى أقصى اليمين فيتم ملأها بأصفار . أما محتويات البت فى أقصى اليسار فتذهب إلى علم الحمل تماما مثل الأمر SAL . شكل (15-10) يبين رسما توضيحيا لهذا الأمر والصورة العامة له ستكون كالتالي :

SHL destination, count

حيث count إما أن تكون واحد إذا كانت الإزاحة ستتم مرة واحدة فقط وأما إذا أردنا الإزاحة أكثر من مرة فيوضع عدد المرات فى المسجل CL ثم يستخدم المسجل CL بدلا من count فى الصورة العامة للأمر . كمثال على ذلك نفترض أن المسجل AL = 10011010 بعد تنفيذ الأمر SAL AL,1 فإن محتويات المسجل AL وعلم الحمل ستكون كالتالي :

AL = 1 0 0 1 1 0 1 0

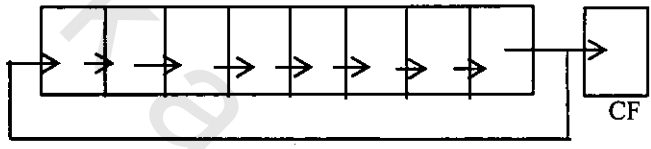
AL = 0 0 1 1 0 1 0 0 ← 0 CF = 1



شكل (15-10) الإزاحة المنطقية لليسار

15-12-5 أمر الدوران لليمين ROR

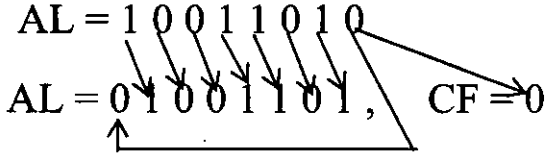
أوامر الدوران تشبه أوامر الإزاحة تماما فيما عدا أن المحتويات التى تزاح من أى جهة يتم إدخالها مرة أخرى من الجهة الأخرى . شكل (15-11) يبين رسما توضيحيا لأمر الدوران لليمين والصورة العامة له ستكون كالتالي :



شكل (15-11) الدوران لليمين

ROR destination, count

حيث count هى كما شرحنا فى الأوامر السابقة تماما . كمثال على ذلك نفترض أن المسجل AL = 10011010 بعد تنفيذ الأمر ROR AL,1 فإن محتويات المسجل AL وعلم الحمل ستكون كالتالي :

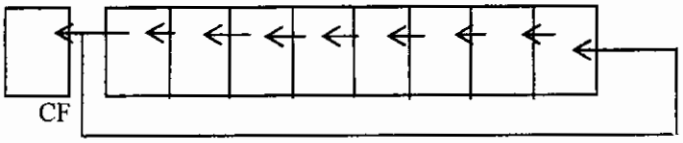


15-12-6 أمر الدوران لليسار ROL

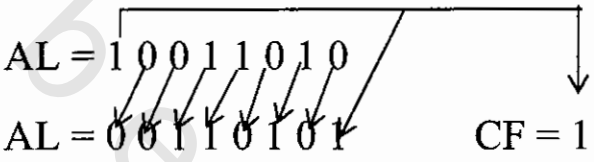
شكل (15-12) يبين رسما توضيحيا لهذا الأمر والصورة العامة له ستكون كالتالي :

ROL destination, count

كمثال على ذلك نفترض أن المسجل AL = 10011010 حيث بعد تنفيذ الأمر ROL AL,1 فإن محتويات المسجل AL وعلم الحمل ستكون كالتالي :



شكل (15-12) الدوران لليسار

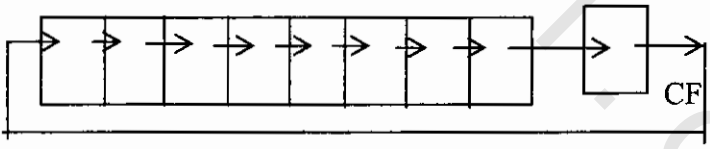


15-12-7 أمر الدوران لليمين من خلال علم الحمل RCR

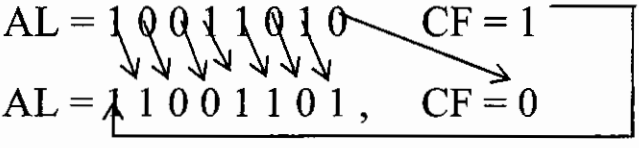
شكل (15-13) يبين رسما توضيحيا لأمر الدوران لليمين من خلال علم الحمل والصورة العامة له ستكون كالتالي :

RCR destination, count

حيث count هى كما شرحنا فى الأوامر السابقة تماما . كمثال على ذلك نفترض أن المسجل AL = 10011010 بعد تنفيذ الأمر RCR AL,1 فإن محتويات المسجل AL وعلم الحمل ستكون كالتالي بافتراض أن علم الحمل كان فيه واحد قبل تنفيذ الأمر :



شكل (15-13) الدوران لليمين من خلال علم الحمل

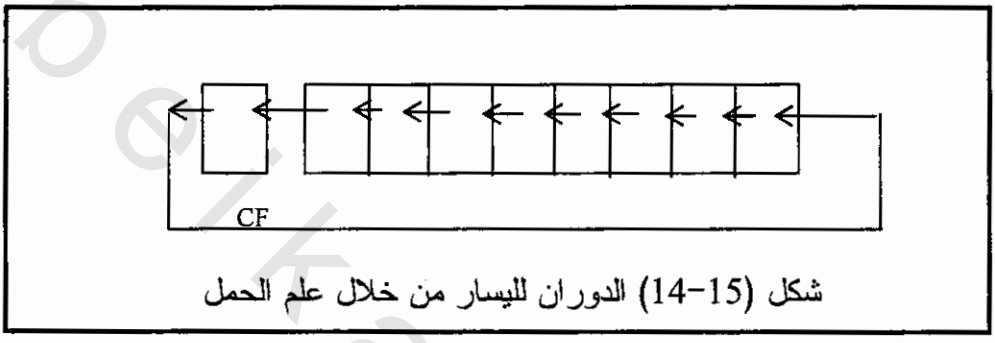


15-12-8 أمر الدوران لليسار من خلال علم الحمل RCL

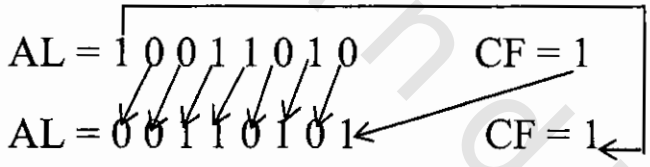
شكل (15-14) يبين رسما توضيحيا لهذا الأمر والصورة العامة له ستكون كالتالي :

RCL destination, count

كمثال على ذلك نفترض أن المسجل AL = 10011010 حيث بعد تنفيذ الأمر RCL AL,1 فإن محتويات المسجل AL وعلم الحمل ستكون كالتالي بافتراض أن علم الحمل كان واحدا قبل تنفيذ الأمر :



شكل (15-14) الدوران لليسار من خلال علم الحمل



مثال 15-8

أكتب برنامجا يحسب مضروب الأرقام المخزنة فى الذاكرة من 1800H إلى 1810H ويخزن المضروب فى الأماكن 1900H إلى 1910H . استخدم البرامج الفرعية معتبرا أن قيمة المضروب لن تزيد عن بايت واحدة .

```
        MOV DI,1800
        MOV SI,1900
        MOV CX,0010H
        MOV BH,00
XX:     MOV BL,[DI]
        PUSH CX
        CALL FACT
        POP CX
        MOV [SI],AL
```

```
        INC SI
        MOV [SI],AH
        INC SI
        INC DI
        LOOP XX
        HLT
FACT: MOV AL,01
        MOV CX,BX
        DEC CX
YY:     MUL BL
        DEC BL
        LOOP YY
        RET
```

12-15 تمارين

1. هل يمكن جمع المسجلين CX و DS ؟
2. إذا كان المسجل CX=1001H والمسجل DX=20FFH فما هى محتويات كل من المسجلين وكذلك قيمة كل علم من الأعلام بعد تنفيذ أمر الجمع ADD عليهما؟
3. كرر السؤال السابق فى حالة تنفيذ أمر الطرح SUB ؟
4. كرر السؤال الثانى فى حالة إجراء الأوامر التالية AND و OR و XOR و NOT ؟
5. أكتب برنامج يضع صفرا فى جميع أماكن الذاكرة 2000H إلى 20FFH ، وبعد ذلك يختبر نفس الأماكن ليرى إذا كان الصفر ما زال موجودا أم لا ؟ هل يصلح هذا ليكون بمثابة اختبار لذاكرة الحاسب ؟
6. اكتب برنامج ينسخ محتويات بلوك الذاكرة 2000H إلى 20FFH فى البلوك 3000H إلى 30FFH ؟
7. أكتب برنامجا يختبر محتويات أماكن الذاكرة 1800H إلى 18FFH ويخرج منها الأعداد الفردية ويخزنها فى الذاكرة من 1900H إلى 19FFH والأعداد الزوجية يخزنها فى الذاكرة من 2000H إلى 20FFH .
8. اكتب برنامج يفحص محتويات الذاكرة من 2000H إلى 2500H ويبحث فيها عن عدد مرات تكرار الرقم 33H المكون من بايت واحدة؟
9. اكتب برنامج يفحص محتويات الذاكرة من 2000H إلى 2500H ويبحث فيها عن عدد مرات تكرار الرقم 33FFH المكون من 2 بايت؟

10. أكتب برنامجا يحسب عدد الوحايد فى كل بايت من بايتات الذاكرة من 1800H إلى 18FFH ويخزنها فى الذاكرة من 1900H إلى 19FFH .
11. أعد السؤال السابق مستخدما البرامج الفرعية .
12. أكتب برنامجا يحسب مضروب الأرقام الموجودة فى الذاكرة من 1800H إلى 18FFH ويخزن هذا المضروب فى الذاكرة من 1900H إلى 19FFH . استخدم البرامج الفرعية .
13. أكتب برنامج يحسب عدد الأرقام السالبة وعدد الأرقام الموجبة فى المدى العنوانى من 1800-18FFH .
14. أكتب برنامج يقرأ محتويات المدى العنوانى 1700-17FFH ثم ينقل البيانات السالبة منها ويخزنها ابتداء من العنوان 1800H والبيانات الموجبة يخزنها ابتداء من العنوان 1900H .
15. أكتب برنامج يحسب عدد الأرقام الفردية وعدد الأرقام الزوجية فى المدى العنوانى 1700-17FFH .
16. اكتب برنامج يبحث عن أكبر رقم من بايت واحدة فى المدى العنوانى 2000H إلى 20FFH ؟
17. اكتب برنامج يقوم بترتيب الأرقام (كل رقم بايت واحدة) الموجودة فى المدى العنوانى 2000H إلى 20FFH ترتيبا تصاعديا؟
18. المدى العنوانى 2000H إلى 3000H يحتوى بيانات إشارة صوتية ، والمطلوب حساب عدد مرات عبور هذه الإشارة للقيمة صفر؟
19. استخدم البرامج الفرعية فى حساب المعادلة التالية :

$M=20! + 30! + 40! + 24^3 + 25^5 + 10^6$