الفصل الأول

أنظمة الكمبيوتر

مقدمه عن النظام الثنائي والنظام السداسي عشر:

إذا كنت تستعجب ما معنى العنوان السابق فهو مجرد نظام آخر للحساب غير النظام العادي العشري decimal الذي يعتمد على عشرة حدود من 0 إلى 9 فمصمموا أول كمبيوتر في العالم لم يروق لهم غير النظام الثنائي الذي يعتمد على حدين فقط هما 0 و1.

وسبب الاختيار هو بساطة هذا النظام للتعبير عن حالة الدارات الالكترونية فحالة الإغلاق off يتم التعبير عنها بالرقم 0 وحالة النشاط on يعبر عنها بالرقم 1

و رغم صعوبة التعامل مع هذا النظام لأي مستخدم إلا انه يمكن بمعادلات بسيطة تحويل الرقم الثنائي إلى أي نظام أخر. وبالعكس

ويستخدم النظام السداسي عشر Hexadecimal اليوم في أجهزه الكمبيوتر الجديدة كما يستخدم النظام الثماني Octal قديما في أنظمة اللينيكس لعرض البيانات من أجهزة الخادم mainframes وحاليا في أجهزة المكتب العادية.

ولذلك تأتى هنا أهمية تعلم النظام الثنائي والنظام السداسي عشر فبدون فهم حقيقي لهذين النظامين وكيفية التحويل بينهم لا يمكن إطلاقا إعادة هندسه أي برنامج حقيقي .

وفيما يلي كيفية القيام بعملية التحويل:

كقاعدة عامة فإن أي نظام حسابي يمكن تحويله إذا افترضنا التعبير عنه بالرمز (B) مثلا حيث يبدأ هذا النظام من الصفر دائما وحتى العدد (B-1) فمثلا النظام العشري يبدأ من 0 إلى 9 ويستثنى من هذه القاعدة النظام الحسابي الروماني الذي لا يوجد به صفر ويبدأ من الواحد

وبالمثل النظام الثنائي يبدأ من 0 إلى 1 والنظام الثماني من 0 إلى 7 بينما يحتوي النظام السداسي عشر على 16 حد ويمكننا التعبير عنه ببساطه عن طريق اخذ 10 حدود 0 إلى 9 كما بالنظام العشري ثم نقوم باستكمال باقي الحدود بالحروف الأبجدية فيتكون هذا النظام من الحدود التالية

0 1 2 3 4 5 6 7 8 9 A B C D E F

و يمكنك استكمال العد عن طريق بقية الأرقام من النظام العشري كما يلي

… E F 10 11 12 … 18 19 1A 1B 1C 1D 1E 1F 20 21 22 …

لذلك فإن 10 في النظام السداسي عشر هو في الواقع 16 في النظام العشري والرقم 20 بالنظام السداسي عشر هو 32 بالنظام العشري و30h هو 48 و40h هو 64 ..... وهكذا

و يمكنك بالمثل استنتاج طريقة العد بالنظام الثنائي هكذا

0 1 10 11 100 101 110 111 1000…

وبهذه الطريقة نقوم بوضع قاعدة عامة فيمكن التعبير عن أي نظام بالرقم 10 أو الرمز (B) فالرقم 10 في النظام الثنائي هو 2 في النظام العشري و10 في النظام الثماني هو 8 في النظام العشري..... وهكذا

و لكن بالطبع لن نقوم بالعد على أصابعنا عند تحويل رقم مثل B6h إلى نظيره العشري فيوجد معادله تؤدي هذا الغرض ويمكن التحويل بسهوله إلى أي نظام إذا قمنا أولا بتحويل الرقم إلى النظام الثنائي ثم إلى النظام الجديد

التحويل من النظام العشري إلى الثنائي:

سنقوم الآن بإجراء عمليات قسمه لأنها أسرع من عمليات الإضافة للوصول إلى الرقم المطلوب ونستعين بذلك بالدالتين DIV و MOD

بالنسبة للكمبيوتر فهو يقوم بتخزين البيانات بطريقتين أساسيتين الأولى هي عبارة عن حدود الرقم مع تخزين موقع العلامة العشرية أو رقم الأساس بالنسبة للأنظمة الغير عشريه والطريقة الثانية هي عن طريق أجزاء كسريه ومعامل (وهي البسط والمقام)

وسنقوم الآن بعملية القسمة كما تعلمنا عن طريق قسمة كل رقم على حدا وتسجيل الباقي وهكذا حتى نصل للنتيجة.

فبالنسبة لأي عمليه قسمه فهناك دائما نتيجتين تعطيان الحل الصحيح وهي القيمة الأساسية للنتيجة والرقم الباقي . فتعبر الدالة DIV عن القسمة الصحيحة أو القيمة الأساسية لناتج القسمة بينما تعبر الدالة MOD عن باقي القسمة.

وكمعلومة عامة الدالة MOD هي المسئولة فعليا عن عملية توليد الأرقام العشوائية التي يستطيع الكمبيوتر أن يقوم بها وعملية انزلاق القوائم المنسدلة وغيرها من العمليات لكن لن نتعرض لهذه الجوانب من الاستخدامات الآن.

ويمكننا التعبير عن عملية استخراج باقي القسمة بالمعادلة 45 MOD 4 والتي تكتب في لغة C هكذا 4 % 45 أما في لغة الباسكال أو الدلفي فتكتب 45 MOD 4 ويكون ناتج هذه العملية هو 11 مع باقي قسمه يساوي 1

DIV = 11, MOD = 1
45 MOD 4 = 1

ويمكننا من هنا إجراء عملية التحويل من النظام العشري إلى النظام الثنائي . الرقم 47 في النظام الثنائي هو الرقم 101111 وستتعلم الآن كيفية استنتاج الرقم الثائي تلقائيا.

فسوف نقوم في الأساس بقسمة الرقم المراد تحويله على 2 وتسجيل قيمة الباقي في كل عملية لأنه يعبر عن الحد الثنائي التالي ويتم ذلك كما يلي:

47 / 2 gives DIV=23 MOD=1, bin string = 1
23 / 2 gives DIV=11 MOD=1, bin string = 11
11/2 gives DIV=5 MOD=1, bin string = 111
5/2 gives DIV=2 MOD=1, bin string = 1111
2/2 gives DIV=1 MOD=0, bin string = 01111
1/2 gives DIV=0 MOD=1, bin string = 101111

لاحظ أننا قمنا ببناء الرقم الثنائي من اليمين إلى اليسار فيتم اعتبار الآحاد من اليمين والعشرات إلى اليسار وبعده تأتي المئات وهكذا .

ويمكننا الآن استخدام طريقة pseudo code التي تعبر عن رسم تخطيطي للبرنامج قبل ترميزه فعليا فنلاحظ من العمليات السابقة أننا توقفنا عندما وصلنا للقيمة DIV = 0 لذلك فيمكننا الآن إتباع الخطوات التالية للتحويل:

1. اخذ قيمه من البرنامج أو المستخدم وتخزينها في متغير وليكن اسمه DIV
2. نقوم بقسمه قيمة DIV على 2 ونترك النتيجة الأساسية في DIV والباقي في MOD
3. تخزين قيمة MOD في متغير نصي يعبر عن الحد التالي للرقم الثنائي
4. 4- تكرار الخطوات من الخطوة 2 حتى الوصول إلى تحقيق الشرط DIV = 0

التحويل من النظام الثنائي إلى النظام السداسي عشر:

نرى هنا سهولة عملية التحويل عند استنتاج المعادلة 16 = 2^4 (2 مرفوعة للأساس 4) هذا يعني أن كل 4 أرقام من النظام الثنائي تكون حد واحد من النظام السداسي عشر

سنقوم بالمثل بالاتجاه من اليمين إلى اليسار للرقم 47

101111

و الآن قم بفصل الرقم إلى مجموعه من أربعه

10 | 1111

ملحوظة: بالنسبة للنظام الثماني فقم بالفصل إلى مجموعه من ثلاثة 8 = 2^3

و الآن نقوم بتكوين جدول التحويل التالي بحيث كل رقم في النظام السداسي عشر يقابله نظيره في النظام الثنائي كما يلي:

0000 = 0
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110 = 6
0111 = 7
1000 = 8
1001 = 9
1010 = A
1011 = B
1100 = C
1101 = D

1110 = E
1111 = F

بالنظر إلى الرقم السابق نجد أن الرقم 10 يساوى 2 بالنظام السداسي عشر والرقم 1111 يساوي F لذلك فإن الرقم 101111 يساوي الرقم 2F في النظام السداسي عشر والذي يساوي 47 في النظام العشري.
بالمثل بعد النظر إلى الجدول السابق يمكن التحويل إلى النظام الثماني بعد التقسيم إلى ثلاث مجموعات 111 | 101 فنجد أن 111 تساوى 7 والرقم 101 يساوى 5 لذلك فإن ناتج هذا الرقم بالنظام الثماني يساوى 57

و بهذه الطريقة سوف تتمكن من فهم كود الاسمبلي الذي سوف نقوم باستخدامه فيما بعد.

نظره عامة للبرمجة :

و نعنى بها هنا الطريقة العامة لسير أي برنامج مكتوبا بأي لغة وكان يتم استخدام رسومات ذو أشكال معينه لتدل على معنى كل عمليه وكانت تسمى flowchart ولكن يفضل الآن الطريقة البسيطة للتعبير عن البرنامج بمجرد توضيح الثلاث عمليات الأساسية في أي برنامج داخل مستطيلات يوجد بها شرح لكل عمليه واسهم تدل على اتجاه سير البرنامج .

وتتضمن العمليات الثلاث الأساسية عملية الإدخال (Input) – عملية المعالجة (Process) – عملية الإخراج (Output) وتسمى اختصارا IPO كما يتضح من الشكل التالي:

ونستطيع توضيح سير أي برنامج بمثال بسيط فإذا تخيلنا موظف يقوم كل يوم صباحا ويذهب للعمل فالخطوات المنطقية لهذه العملية هي:

النهوض من السرير – الذهاب للحمام – تحضير الإفطار – ارتداء ملابس العمل – يركب أتوبيس الشركة – الوصول للعمل

فإذا تخيلنا مثلا عدم وجود مواصلات للذهاب للعمل فإن الموظف سيذهب متأخرا أو قد لا يصل على الإطلاق وهكذا إذا تم حذف أي خطوة من الخطوات فإن هذا سيؤثر سلبا على الموظف ونتيجة عمله .

وبالنسبة للكمبيوتر فأنه يحتاج دائما إلى بيانات أولا وتسمى المدخلات وقد تكون من مستخدم أو من جهاز متصل بالكمبيوتر مثل لوحة المفاتيح أو حتى برنامج آخر ثم يتم معالجه هذه المدخلات بطريقة خاصة (عمليات حسابيه – عملية فهرسه) تؤدى للحصول على نتيجة التي يتم عرضها بعد ذلك للمستخدم أو

إرسالها إلى جهاز مثل الطابعة أو حتى إلى برنامج آخر يقوم بالمزيد من المعالجة.

هذه الطريقة للبرمجة هامه جدا عندما تترك البرنامج لشهور عديدة أو سنين ثم ترجع بعد ذلك وتحاول أن تقوم بتعديل أي جزء منها فأنك لن تتذكر فائدة الكود وغالبا لن تفهمه أيضا إذا لم تحتفظ بمخطط تفصيلي للبرنامج بجانب إضافة تعليقات داخل الكود الفعلي للبرنامج.

نموذج التركيب الداخلي للملفات التنفيذية:

يمكننا أن نوضح التركيب الداخلي للملفات من نوعية COM. وEXE. من خلال النموذجين التاليين:

ملفات COM. التنفيذية:

فيما يلي مثال لملف COM. تم إنشاؤه بالمعالج TASM :

```
    .model small
    .code
    .386
     org    100h
start:  jmp    MAIN_PROGRAM
;----------------------
; Your Data Here
;----------------------
;----------------------
MAIN_PROGRAM:
;---------------
; Your Code Here
;---------------
;---------------
mov al, 0h          ; return code (0 = no error)
exit_program:
mov ah,4ch          ; quit to DOS
int 21h
```

```
end    start
```

مثال آخر لملف COM. يمكن مقارنته بالنموذج الخاص بالملفات EXE.

```
COM_PROG  segment byte public
      assume  cs:COM_PROG
      org     100h
start:  jmp     MAIN_PROGRAM
;----------------------
; Your Data Here
;----------------------
;----------------------
MAIN_PROGRAM:
;---------------
; Your Code Here
;---------------
;---------------
mov al, 0h          ; return code (0 = no error)
exit_program:
mov ah,4ch          ; quit to DOS
int 21h
COM_PROG  ends
      end     start
```

نموذج ملف تنفيذي EXE. تم إنشاؤه عن طريق المعالج TASM:

```
.model small     ; normally small medium or large here
.stack 200h
.code
EXE_PROG  segment byte public
   assume
cs:EXE_PROG,ds:EXE_PROG,es:EXE_PROG,ss:EXE_PR
OG
start:  jmp     MAIN_PROGRAM
```

```
;---------------------
; Your Data Here
;---------------------
;---------------------
MAIN_PROGRAM:
;---------------
; Your Code Here
;---------------
;---------------
mov al, 0h          ; return code (0 = no error)
exit_program:
mov ah,4ch          ; quit to DOS
int 21h
EXE_PROG  ends
      end     start
```

نموذج آخر للملف EXE. يمكن مقارنته بالنموذج الأول للملفات COM.

```
;dosseg                    ; directive is ignored in tasm 4, so
            ; uncomment it if you have any errors

.model small
.stack 200h
.data
.code
start:  jmp     MAIN_PROGRAM
;---------------------
; Your Data Here
;---------------------
;---------------------
MAIN_PROGRAM:
;---------------
; Your Code Here
;---------------
```

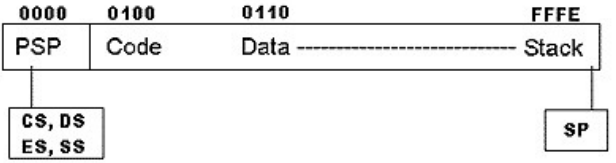
```
;---------------
mov al, 0h          ; return code (0 = no error)
exit_program:
mov ah,4ch          ; quit to DOS
int 21h
    end    start
```

باستعارة تعريف الملفات التنفيذية من دليل لغة الاسمبلي الخاص بشركة IBM يمكننا أن نحدد نوعين من الملفات التنفيذية بناء على الامتداد المستخدم وهما الملفات التنفيذية COM وEXE

ويمكن استخدام أمر Debug الخاص بالدوس لإنشاء برامج من نوعية COM والتي تعتبر صوره مباشره لترجمة الكود من الاسمبلى إلى أرقام ثنائيه ويتم تحميلها في ذاكرة الدوس في أدنى عنوان متاح segment وبيانات الملف Code Data , Stack Data يتم تخزينهم فعليا في نفس العنوان المنطقي أو الفعلي

و في هذا النوع من الملفات يتم أنشاء بادئه البرنامج (Program Segment Prefix) في العنوان Offset 0 (يرجى مراجعة كتب الاسمبلى للتعرف جيدا على المصطلح PSP)

وقد يصل حجم البرنامج إلى 64K ناقص منه حجم PSP و2 bytes محجوزين في الذاكرة Stack وتبدأ منطقة الكود في العنوان 100h ويليها مباشره منطقة Data area وفي الآخر منطقة Stack كما يتضح من الرسم التالي:



مثال:

يمكنك كتابة برنامج بسيط بالاسمبلى لعرض الرسالة المشهورة "Hello World" كما يلي:

```
.model tiny
.code
org 100h

maine proc
  mov ah,9
  mov dx, offset helo_msg
  int 21h
  mov ax, 4c00h
  int 21h
maine endp

helo_msg db 'Hello, world!' '$'
end maine
```

و يمكن تنفيذه في برنامج Debug كما يلي

```
C:\>debug [اضغط على مفتاح الإدخال]
-a
mov ah,9
mov dx,108
int 21
ret
db 'Hello, World!',0d,0a,'$'
[ اضغط المفتاح F5]

r cx
```

```
18
n c:\hello.com
w
q
```

قم الآن بكتابة اسم الملف في الدوس كما يلي

C:\> hello [اضغط على مفتاح الإدخال]

- لاحظ أن المتجهات DOSSEG, .DATA and .STACK غير ضرورية وقمنا باستخدام المتجه org لتحديد مكان بداية الكود في العنوان 100h وهذا يعطى مساحه كافيه لـ PSP التي تأخذ العنوان 0 دائما كما ستجد أيضا أن أي ملف EXE يقوم بنفس الوظيفة سيكون حجمه اكبر من ملفات COM.

نظرة مبدئية للغة الاسمبلى:

بعد أن تعرفت على النظرية العامة لتطوير أي برنامج قد تود الآن أن تقوم بكتابه برنامج بنفسك وها ما سنفعله الآن وسنقوم بعمل برنامج صغير يمكننا أن نقوم بإدخال قيم مختلفة به.

سنقوم الآن بشرح ما يسمى registers وهي تعتبر متغيرات موجودة في أي جهاز كمبيوتر ويمكن تخزين بيانات بها مثل المتغيرات العادية الخاصة بأي لغة برمجة ولكن هذه المتغيرات تتميز بسرعة فائقة.

تعتبر لغة البيزيك من اللغات القديمة والسهلة والتي تشبه أوامرها اللغة الانجليزية وكانت معظم الأجهزة قديما تحتوى على محاكي ما يمكن عن طريقه البرمجة بهذه اللغة وسنقوم باستخدامها لمعرفة الكثيرين بهذه اللغة ويمكننا بعد ذلك تحويل الكود من البيزيك إلى مثيله بالاسمبلى

مثلا في برنامج مكتوب بالبيزيك لطبع كلمة "Hello Hollywood" يمكننا أن نرى التعليمات التالية

```
DATA "Hello Hollywood";
READ D$; [من السطر السابق]
PRINT D$;
```

أو بطريقة أخرى

```
LET D$ = "Hello Hollywood";
PRINT D$;
```

و لكن بالنسبة للاسمبلى فسوف يناسبنا أكثر إذا حولنا الكود من النموذج الأول فيتم تحويل أول سطرين بالتعليمات التالية

```
MYSTRING DB ' Hello Hollywood ','$'
MOV DX, OFFSET MYSTRING
```

ولكي نقوم بطبع النص الحرفي على الشاشة نقوم باستخدام أمر ثابت يستخدم دائما لهذا الغرض وهو

```
MOV AH, 09h
```

ولكن الأمر الفعلي الذي يجعل الدوس ينفذ هذا الأمر هو الأمر التالي

```
INT 21h
```

و الآن الكود بطريقة كاملة

```
MYSTRING DB 'Hello Planet Hollywood','$'
MOV DX, OFFSET MYSTRING
MOV AH, 09h
INT 21h
```

قم الآن بالمقارنة مع برنامج البيزيك ستجد أن برنامج الاسمبلى ليس صعبا ولكنه أطول.

لاحظ علامة '$' في آخر النص الحرفي . يحتاج الاسمبلى إلى علامة تدل على نهاية النص الحرفي لذلك نضع العلامة '$' ليعرف مكان النهاية فإذا لم نضعها فإن الاسمبلى يقوم بمتابعه باقي النص الذي يلي الكلمة المراد طبعها فيتم طباعه الكود على انه نص حرفي.

و يوجد طريقة أخرى يستطيع الاسمبلى التعامل مع النصوص وهي إنهاء النص بالرقم 0 بدلا من '$' كما يمكنك دائما تحديد الحرف الذي يعبر عن انتهاء النص

لاحظ من المثال السابق انه تم فصل الجزء الخاص بالكود عن الجزء الذي يحتوى بيانات البرنامج وإلا قام البرنامج بتنفيذ البيانات Data كجزء من كود البرنامج وكما تلاحظ من أمثلة ملفات EXE. وCOM. أننا قمنا بالفصل بين البيانات والكود فعندما يبدأ البرنامج بالتنفيذ فيقوم البرنامج بالقفز إلى الجزء الخاص بالكود وبذلك لا يمكن أن يقوم بتنفيذ تعليمات خاطئة.

و هذا لا يختلف عن البرامج المكتوبة بالبيزيك فيقوم المبرمجون عادة بوضع البيانات في آخر الملف وهكذا لم يختلف كثيرا عن الاسمبلى كما يمكنك أن تلاحظ أن كل لغات البرمجة الحديثة التي ظهرت بعد الاسمبلى تتعامل مع النصوص وتشكيلها بنفس الطريقة لان البنية التحتية الخاصة بهذه اللغات مكتوبة بالاسمبلى أو جزء كبير منها على الأقل.

كيف تعمل المتغيرات Registers:

من المثال السابق قمنا باستخدام المسجل DX كمتغير نصي وفي البيزيك قمنا باستخدام المتغير D$ لتتم المقارنة بسهوله ففي لغة البيزيك يمكنك أن تقوم بتعريف أي متغير تريده أما بالنسبة للاسمبلى فلديك فقط بضعة مسجلات يمكنك الاختيار منهم وهذا مناسب فلن تحتاج إلى أكثر من ذلك . بالنسبة للاسمبلى يمكنك تخزين أي بيانات في أي مكان بالذاكرة مباشره وليس في متغيرات مثل البيزيك وفيما يلي عرض لهذه المسجلات:

AX – The Accumulator (يستخدم عادة لتخزين ناتج العمليات الحسابية)

BX – The Base register (يستخدم عادة لتخزين مكان المنشآت في الذاكرة)

CX – The Counter (يستخدم لعد أي شئ مثل عدد حروف نص معين)

DX – The Data Register (يستخدم عادة لتخزين مؤشر لمكان أو بيانات مخزنه بالذاكرة)

يمكن استخدام المسجلات السابقة في أي استخدام ولكن عند استخدام النداءات الخاصة بالأمر int 21 فيتوقع الدوس وجود بيانات معينه في مسجلات معينه ويمكن ملاحظة ذلك من المثال السابق .

المسجل AX يعتبر أكثر المسجلات أهميه نظرا لتعدد استخداماته فهو دائما يقوم بتخزين أي شئ عند الخروج من البرنامج أو من روتين فرعى (جزء من الكود داخل البرنامج) وعادة يتم تخزين اكواد الخطأ أو نتيجة عمليه حسابيه وعندما يتم استدعاء روتين آخر أو برنامج آخر تستخدم لتخزين كود الأمر مثل AH=9 من المثال السابق

تعتبر هذه المسجلات من نوعيه 16 bit أي يمكن لأي واحد منهم أن يحمل بيانات تصل إلى 16 bit أي حرفين 2 bytes من البيانات ويمكن الوصول المباشر إلى من الحرفين عن طريق قيمة low أو قيمة high لذلك المسجل AX مثلا ينقسم في الواقع إلى AH وAL وهكذا المسجل BX إلى BH وBL ...

وبالنسبة للمسجلات 32 bit فتسمى ببساطه إلى EAX, EBX, ECX and EDX .

والآن بعد أن تعرفنا على المسجلات الأساسية يمكننا الانتقال إلى مسجلات أخرى لها استخدامات خاصة فالمسجلين التاليين يستخدمان عادة لنسخ مصفوفة نصيه أو مصفوفة لاماكن في الذاكرة ويعرفان بالمسجلات النصية (String Registers)

DI – Destination Index (لتحديد المكان الذي سيتم النقل إليه)

SI – Source Index (لتحديد مصدر النقل)

و يستعمل معهم المسجل التالي

BP – Base Pointer

و يمكن عن طريقة ما معرفة المكان الحالي في الكود ولا فرق في الاستخدام بينهم إلا إذا كان هناك شئ ما يتوقع استخدامك لمسجل معين مثل الفيروسات فبعضهم يستخدم المسجل SI والبعض الآخر يستخدم BP

و هناك المسجل الهام IP أو Instruction Pointer ويستخدم بواسطة المعالج لتحديد أي التعليمات هي التي يجب تنفيذها.

فمثلا عندما تقوم باستخدام برنامج Soft Ice لاختبار برنامج فمن الممكن أن يكون موضع التنفيذ داخل حلقه تكراريه فيمكنك الخروج من هذه الحلقة عن طريق زيادة قيمة IP للخروج من الحلقة ويمكنك تنفيذ العديد من الأشياء بواسطتها ولكن احترس لان تغيير مسار البرنامج الطبيعي قد يؤدى إلى نتائج غير متوقعه.

و سنقوم الآن في الفصل التالي بشرح كيفيه كتابة برامج تتعامل مع المستخدم وكيف نقوم بتمرير قيمه إلى البرنامج وقراءتها .