الفصل الثاني

برامج تفاعلية

كتابة برامج تفاعليه

سنقوم الآن بكتابة برنامج يتعامل مع المستخدم ولكن نريد أولا أن نعرف أن استخدام الفاصلة المنقوطة في أول السطر يعنى أن هذا السطر للتعليقات فقط ويقوم المعالج بتجاهله تماما وهذا متوافق تماما مع تربو اسمبلى وميكرو اسمبلى ولكن لا تقوم باستخدام الفاصلة المنقوطة عند كتابه تعليمات اسمبلى داخل كود السى أو الباسكال.

في الكود التالي يقوم البرنامج ببساطه بطباعة نص حرفي مثلما قمنا في المثال السابق

```
;*********************************************
;
; .COM File Program Template;
;;
;*********************************************
        .model small
        .code
        .386
        org    100h
start:  jmp    MAIN_PROGRAM
;----------------------
; Your Data Here
;----------------------
        CopyMsg    db     'this is the string to
print',0Dh,0Ah,'$'
        ;----------------------
MAIN_PROGRAM:
        ;---------------
        ; Your Code Here
        ;---------------
```

```
        mov dx, offset CopyMsg      ; نقوم هنا بإخبار الدوس أين تم تخزين الرساله
        mov ah, 09h                 ; هذا الأمر يعنى طباعة الرسالة
        int 21h                     ; إخبار الدوس بتنفيذ الأمر
        ;---------------
        mov al, 0h                  ; إرجاع صفر ليدل على عدم حدوث أخطاء
EXIT_PROGRAM:
        mov ah,4ch                  ; الخروج إلى الدوس
        int 21h
        end    start
```

لاحظ هنا إننا قمنا باستخدام التعريف CopyMsg ويسمى Label والذي يمكننا من الوصول إلى النص الحرفي في أي مكان وعموما فإن النصوص الحرفية في الاسمبلى يعبر عنها بمكانها في الذاكرة والتي تسمى offset وإذا نظرت إلى MAIN_PROGRAM فهي أيضا تعتبر label وليس نص حرفى ولا نقوم بالإشارة إلى مكانها بالذاكرة باستخدام الموجه offset في هذه الحالة.

وللتبسيط يمكنك أن تعتبر النصوص الحرفية في الدوس هي أماكن في الذاكرة offsets تنتهي دائما بالحرف $ أو الصفر

فمثلا كود مفتاح الإدخال في الاسمبلى (Enter) هو 0Dh,0Ah حيث تعبر 0D عن الكود بالنظام السداسي عشر لتحريك المؤشر إلى اليسار إلى نهاية السطر والكود 0A يعبر عن بدء سطر جديد على الشاشة وكمثال آخر على هذه الاكواد هو الكود 07h الذي يعبر عن صوت رنه مكتومة مشهورة بالاسم beep وهي موجودة للان في جميع إصدارات الويندوز ويمكن تنفيذها عن طريق داله API تسمى Message Beep.

وبالنسبة لاستقبال حرف من لوحة المفاتيح فنقوم باستخدام الكود التالي:

mov ah,08h ; دالة دوس تستخدم للحصول على المفتاح الذي قام المستخدم بالضغط عليه

int 21h ; لإخبار الدوس بتنفيذ الأمر

وبعد تنفيذ هذه التعليمات يتم تخزين الحرف الذي قام المستخدم بإدخاله في المسجل AL والسؤال الآن كيف نقوم بطباعة ما قام المستخدم بإدخاله على الشاشة ؟

في هذه الحالة نقوم باستخدام الدالة 06h ولكن هذه الدالة تتوقع وجود الحرف المراد طبعه في المسجل DL لذلك فنقوم أولا بنقل الحرف من المسجل AL إلى المسجل DL ونستخدم الأمر mov ولا يعنى هذا الأمر النقل لكنه فعليا يقوم بنسخ قيمة المسجل AL إلى المسجل DL كما يلي:

mov dl, al ; نسخ الحرف

mov ah,06h ; طباعة الحرف على الشاشة

int 21h ; تنفيذ الأمر

و سنقوم الآن باستخدام هذا الكود في المثال السابق كما يلي:

```
;*********************************************
; KEYPRESS.ASM
; Interactive Program
;
; Compile with:
;
;    TASM KEYPRESS.ASM
;
;*********************************************
        .model small
        .code
        .386
        org    100h
```

```
start:  jmp     MAIN_PROGRAM
;----------------------
; Your Data Here
;----------------------
CopyMsg     db      'some kind of string',0Dh,0Ah,'$'
PressEnter  db      0Dh,0Ah,'$'
;----------------------
MAIN_PROGRAM:
;---------------
; Your Code Here
;---------------
; DISPLAY OUR MESSAGE
mov dx, offset CopyMsg      ; اخبار الدوس بمكان الراساله
mov ah, 09h                 ; الامر 9 في الدوس لطبع الرساله
int 21h                     ; تنفيذ الامر
mov dx, offset PressEnter   ; تخزين نص مفتاح الأدخال في المسجل
الخاص بالدوس
mov ah, 09h                 ; طباعة النص
int 21h                     ; تنفيذ الامر
; الحصول على الحرف من المستخدم
mov     ah,08h      ; استخدام دالة الدوس للحصول على الحرف
int     21h         ; تنفيذ الامر
;اظهار الحرف المدخل
mov     dl, al      ; نسخ الحرف
mov     ah,06h      ; طباعة الحرف على الشاشة
int     21h         ; تنفيذ الأمر
; يمكننا أيضا تجربه الكود الخاص بصوت الجرس
```

```
mov     dl, 07h     ; نسخ الكود الخاص بالصوت إلى المسجل
mov     ah,06h      ; طباعة محتويات المسجل إلى الشاشة
int     21h         ; تنفيذ الأمر
;---------------
mov al, 0h          ; إرجاع كود 0
mov ah,4ch          ; الخروج للدوس
int 21h
end     start
```

لاحظ أن الكود 06 لا يعنى بالضرورة طباعة حرف مرئي على الشاشة ولكن عندما يرى الدوس الكود المطبوع يفهم ما هو المراد منه وهو في هذه الحالة صوت الجرس.

و يمكنك الآن أن ترى كيفية عمل برنامج متكامل له فائدة معينه وعندما تعمل على جهاز كمبيوتر ويكون أمام ك شاشه الكمبيوتر بها مؤشر الكتابة يظهر ويختفي فهو لا ينتظر فقط أن تقوم بكتابة حرف ولكن يقوم البرنامج المستخدم باستخدام حلقه تكراريه لاختبار إذا كان المستخدم قام بإدخال حرف أم لا.

و يتم استخدام الأمر INT 21 (اختصار interrupt) والذي يقوم بقطع عمل معالج الكمبيوتر المعتاد وتنفيذ الأمر وفي المثال حتى لا يستمر في حلقة لا نهائيه لاختبار ما إذا كان المستخدم قام بإدخال حرف ام لا من لوحة المفاتيح ولا تعتبر هذه مشكله للكمبيوتر فأي عملية إدخال سواء في برنامج أو لعبه هي في الحقيقة حلقه تكراريه لا نهائيه .

لذلك السبب يجب وضع شرط للخروج من هذه الحلقة عن طريق الضغط على حرف معين مثل حرف Y أو N اختصارا yes وno مع قبول الحروف الصغيرة أيضا وسنقوم بتنفيذ ذلك في المثال التالي مع العلم أن هذه الطريقة تجعل الملف التنفيذي أطول نسبيا من طرق أخرى ولكننا نستخدمها للتبسيط

مثال:

لاحظ وجود نقطتين فوق بعض التعريفات; :BEGIN_OF_LOOP
هنا تقوم بوضع الكود المراد تكراره ;
القفز إلى التعريف السابق لتكرار الحلقة ; JMP BEGIN_OF_LOOP
تعريف لتحديد بداية البرنامج بعد الحلقة ; :PROGRAM_START

والآن سنقوم بمعرفة الكود اللازم للخروج من الحلقة والنداء على التعريف PROGRAM_START وتذكر أننا نستخدم الكود التالي للحصول على الحرف من المستخدم

دالة 8 في الدوس ; mov ah,8
تنفيذ الأمر ; int 21h

ويجب علينا بعد ذلك معرفة إذا كان هذه المفتاح هو المفتاح الذي نريده فنستخدم الأمر CMP (compare) والذي يقوم بمقارنة شيئين ووضع النتيجة في المسجل AL فإذا كانت النتيجة True فإن AL = 0 أما إذا كانت النتيجة False فإن قيمة AL تكون شيئا آخر غير الصفر .
وبما أن الحرف مخزن في المسجل AL فإن الأمر CMP سيقوم بمسح هذه القيمة لذلك سنقوم بتخزين هذه القيمة في مكان امن مثل المسجل BL
ونستخدم الأمر CMP للمقارنة مع الحرف Y كما يلي:

```
CMP BL, 'Y'
```

لاحظ أن الحرف Y موضوع حوله فاصلتين ليدل على انه نص حرفي وإلا ارتبك معالج الاسمبلى ولكن مع الأرقام فتكتب بطريقة عاديه كما يلي:

```
CMP BL, 89
```

وبالنسبة للأرقام السداسية عشر كما يلي:

CMP BL, 059h

وبالنسبة للثلاث أمثله السابقة فنتيجتهم واحده لان كود اسكى ASCII للحرف Y هو 89 بالنظام العشري و59 بالنظام السداسي عشر

وهناك أمر هام جدا يستخدم دائما مع الأمر CMP وهو الأمر JZ والذي يعمل بناء على نتيجة الأمر CMP ومعناه إذا لم تكن خمنت هو (jump if zero) أو اذهب للعنوان التالي إذا كانت النتيجة صفر كما يسمى الأمر JZ أيضا بالاسم JE أو (jump if equal) ويتم القفز إلى عنوان معين إذا كانت النتيجة صفر أو تكملة سير البرنامج كما هو

وكما يوجد الأمر المناقض للأمر JZ وهو الأمر JNZ أو (jump if not zero) أو JNE وهو يؤدى عكس وظيفة الأمر JZ

و فيما يلي مثال يتم فيه شرح الأوامر السابقة

```
BEGIN_OF_LOOP:  ; بداية الحلقة
; الحصول على حرف من المستخدم
    mov    ah,8
    int    21h
    mov    bl, al      ;BL حفظ الحرف في المسجل
;CMP لان المسجل AL يتم تغييره بفعل الامر
    cmp    bl, 'Y'     ;Y اختبار إذا كان المستخدم قام بالضغط على
الحرف
    je     PROGRAM_START
    cmp    bl, 'N'     ;N اختبار إذا كان المستخدم قام بالضغط على
المفتاح
```

```
    je          PROGRAM_START
JMP BEGIN_OF_LOOP       ; الرجوع للتعريف لبدء حلقه أخرى
PROGRAM_START:;  تعريف ليدل على بداية البرنامج بعد الخروج من
الحلقه
```

و فيما يلي المثال بالكامل ويمكن تنفيذه بالمعالج TASM

```
;*********************************************
; KEYPRESS.ASM
; Interactive Program
;
; Compiled with:
;
;     TASM KEYPRESS.ASM
;     TLINK /t KEYPRESS.OBJ
;*********************************************
.model small
.code
.386
org    100h
start:  jmp    MAIN_PROGRAM
;----------------------
; Your Data Here الجزء الخاص ببيانات الملف
;----------------------
CopyMsg    db     'crack book 2005',0Dh,0Ah,'$'
PressEnter  db     0Dh,0Ah,'$'
;----------------------
MAIN_PROGRAM:
;---------------
; Your Code Here
;---------------
; عرض الرساله السابقه
        mov dx, offset CopyMsg
```

```
        mov ah, 09h             ; طباعة الرسالة
        int 21h
        ; طباعة سطر جديد
        mov dx, offset PressEnter  ; tell DOS where string is
        mov ah, 09h
        int 21h
BEGIN_OF_ENDLESS_LOOP:
    mov    ah,8
    int    21h
    mov    bl, al
    cmp    bl, 'Y'
    je     PROGRAM_START
    cmp    bl, 'N'
    je     PROGRAM_START
    cmp    bl, 'y'          ;اختبار الحروف الصغيرة
    je     PROGRAM_START
    cmp    bl, 'n'
    je     PROGRAM_START
    ; نقوم بإطلاق صفاره إذا تم الضغط على أي مفتاح آخر
     mov    dl, 07h
     mov    ah,6
     int    21h
     JMP BEGIN_OF_ENDLESS_LOOP
PROGRAM_START:
    ; إظهار الحرف الذي قام المستخدم بالضغط عليه
     mov    dl, bl
     mov    ah,6
     int    21h
    ;---------------
    mov al, bl              ; استخدام النتيجة لتكون هي كود إنهاء
```

البرنامج

```
    mov ah,4ch              الخروج للدوس ;
    int 21h
    end    start
;-------------------------------------------------------------------------
```

والآن للتمرين حاول تغيير البرنامج السابق ليقوم بطبع رسالة مع الحرف الذي تم الضغط عليه مثلا "You type no" أو "you agree" مع الحرف 'y' أو 'Y'

ويمكنك أيضا إضافة رسالة في أول البرنامج ليطبع رسالة توضح أن البرنامج يقوم بانتظار احد حرفين من المستخدم (Y/N) حتى يعرف المستخدم ما هو الحرف الذي يجب الضغط عليه .

لاحظ أن آخر جزء من البرنامج يقوم بإظهار كود الخطأ Error Level فيمكن بذلك استخدام هذا البرنامج مع برنامج خارجي بحيث يمكنه تحديد نتيجة تنفيذ هذا البرنامج بناء على رقم الخطأ.

التقنيات المتقدمة للاسمبلى

فيما يلي عدة تقنيات هامه جدا عند التعامل مع البرامج الكبيرة:

قم بوضع تعليقات في كل مكان ممكن

فمثلا السطر التالي

```
mov cx, 9
```

إذا وضعنا التعليق "cx = 9" لا يعتبر تعليق جيد أما إذا قمت بكتابة مفهوم ما يفعله هذا السطر مثل "cx = loop 9 times" أي انه سيتم استخدام cx في حلقه تكراريه لتسع مرات فستجد أن هذا التعليق هام جدا لفهم البرنامج فيما بعد بسرعة.

و إذا كنت مبرمج مبتدئ فثق تماما أن هذا الموضع من الأمور المسلمة به في برمجة التطبيقات الكبيرة خصوصا عندما يعمل أكثر من واحد في نفس البرنامج فبدون هذه التعليقات يكون إمكانية التعديل واكتشاف الأخطاء تقريبا مستحيلا.

ولا يقتصر وجود التعليقات على سطور الكود المختلفة فقط ولكن أيضا على أقسام الكود نفسه حتى يمكن فهم الفكرة العامة للكود في هذا القسم وما سيؤدى إليه هذا الكود .

تقسيم البرنامج Modularity:

يجب أن تعرف الآن أن البرنامج يتكون من عدة سطور متتالية يتم تنفيذها الواحدة تلو الأخرى حتى نهاية البرنامج وقد تكون الآن قد كونت فكره أن البرامج كلها يتم برمجتها مره واحده من البداية للنهاية ولكن هذا ابعد ما يكون عن الحقيقة فالبرامج كلها خصوصا الكبيرة منها يتم تقسيمها إلى عدة أقسام منفصلة يقوم احدها بالنداء على الآخر بغض النظر عن مكانه الفعلي في البرنامج

فإذا حاولت أن تكتب برنامج مره واحده بدون تقسيم فلن تستطيع أن تكمل هذا البرنامج لأكثر من 64 ألف سطر فلن يستطيع العقل البشرى المحدود أن يستوعب هذه السطور دفعه واحده ولن تستطيع أبدا أن تقوم بتغيير بسيط في البرنامج بدون التأثير سلبا على بقية البرنامج (ظهور أخطاء منطقيه) أو إعادة كتابة البرنامج من جديد كلما حاولت عمل أي تعديل.

أما إذا كان لديك تقسيم منطقي للبرنامج خصوصا للأجزاء التي يستخدمها البرنامج بكثرة في عمله فيمكنك فقط تعديل هذا الجزء بدون إعادة كتابة البرنامج كله أو البحث عن كل الإمكان التي يجب تغييرها في الكود .

بالإضافة لذلك يكون ترجمة البرامج المقسمة إلى أجزاء صغيرة أسرع لان المعالج يستهلك مساحه اقل من الذاكرة للترجمة.

والآن السؤال هو كيف يمكن تنفيذ ذلك ؟

في كل لغات البرمجة الحديثة يمكن تقسيم البرنامج إلى أجزاء صغيرة عن طريق استخدم الإجراءات أو procedures وفي لغة الاسمبلى تسمى procs وبالنسبة للغة مثل السى فتسمى دوال functions أما لغة الباسكال فتسمى procedures وتتعامل كل لغة مع الإجراءات بطريقة مختلفة قليلا عن الأخرى .

مثال: في معالج الاسمبلى TASM يمكن يكون الإجراء بالشكل التالي:

```
PrintLine   proc    near
    mov     ah, 9
    int     21h
    ret             ; امر الرجوع من الاجراء
endp        PrintLine
```

و الآن لدينا إجراء يمكن النداء عليه لطبع سطر ويمكن استخدامه كما يلي:

```
mov dx, offset MyMessage
call    PrintLine
```

كما ترى فإن هذا مفيد جدا بدلا من تكرار كود الطباعة كلما أردت طباعة نصوص حرفيه

ملحوظة: رغم الفوائد العديدة للإجراءات إلا انه يمكن بسهوله التداخل بينهم أي اشترك معرف مثلا له نفس الاسم في الاثنين كما يمكن أن نرى من المثال التالي:

```
CompareByte proc        near
    Loop:
            inc ah
            cmp ah, 092h
            jne     Loop
            ret
endp        CompareByte
;----------------------
CompareWord proc        near
    Loop:
            inc ax
            cmp ax, 02942h
            jne     Loop
            ret
endp        CompareWord
;----------------------
```

من المثالين السابقين نجد أن في الإجراء CompareWord والإجراء CompareByte أننا قمنا باستخدام المعرف "Loop" مرتين فلن يعرف المعالج في هذه الحالة أي منهم سيقوم باستخدامه لذلك يعطى المعالج رسالة خطأ

لذلك من الطبيعي أن نقوم بعمل اختلاف في الأسماء بينهم كما يلي :

```
;----------------------
CompareByte proc        near
    CmpByteLoop1:
            inc ah
            cmp ah, 092h
            jne     CmpByteLoop1
            ret
endp        CompareByte
;----------------------
CompareWord proc        near
    CompareWordLoop1:
            inc ax
            cmp ax, 02942h
            jne     CompareWordLoop1
            ret
endp        CompareWord
;----------------------
```

في حالة البرامج كبيرة الحجم سيكون من المرهق جدا محاولة إيجاد طريقة للتمييز بين مختلف التعريفات لذلك يوجد طريقة سهلة لتنفيذ ذلك وهى استخدام المتجه IDEAL MODE وسوف نقوم باستخدامه في رأس نموذج الكود السابق بإضافة الكلمة ideal

بهذه الطريقة يعرف TASM أننا سوف نستخدم الحالة النموذجية للمعرفات وفى ما يلى توضيح ذلك في الأمثلة السابقة :

```
;----------------------
    COM_PROG  segment byte public
        ideal
        assume  cs:COM_PROG
        org     100h
    start:
```

```
;----------------------
```

وهذا يجعل طريقتنا في كتابة الإجراءات تختلف قليلا كما يتضح من الكود التالي:

```
proc        PrintLine
    mov     ah, 9
    int     21h
    ret             أمر الرجوع من الإجراء ;
endp        PrintLine
```

ملاحظة: قمنا هنا باستخدام الكلمتين proc وendp لتسهيل معرفة بداية ونهاية الإجراء لكن الميزة الأساسية هي استخدامنا للعلامتين @@ ويتضح ذلك من التغير في الكود الخاص بالإجراء CompareByte:

```
;----------------------
proc        CompareByte
    @@Loop:
            inc ah
            cmp ah, 092h
            jne     @@Loop
            ret
endp        CompareByte
;----------------------
proc        CompareWord
    @@Loop:
            inc ax
            cmp ax, 02942h
            jne     @@Loop
            ret
endp        CompareWord
;----------------------
```

الرمزين @@ يدلان على استخدام المدى المحلى بمعنى أن أي معرف يقع داخل الإجراء proc مع وجود الرمزين @@ أمام هـ فهو معرف فقط (محلى) داخل هذا الإجراء ولن يراه أي كود آخر في البرنامج .

ويعتبر أي شيء آخر غير محلى معرف عالميا Global بمعنى أنه متاح لجميع أجزاء الكود بالبرنامج

استخدام إجراءات أو تعريفات عالمية global مفيد فقط في حالات معينة مثل الألعاب فتصبح هذه الطريقة مفيدة جدا للحصول على إجمالي النقاط التي تم إحرازها خلال اللعب

وبالعكس فإن استخدام الطريقة المحلية تتميز بإمكانية كتابة برامج كبيرة الحجم بدون وجود تعريفات متكررة يمكن أن تؤثر على النتيجة المنطقية للبرنامج

وبالنظر إلى المثال السابق الذي فيه ينتظر البرنامج فيه إدخال احد قيمتين من المستخدم y أو n سنقوم بتعديل هذا البرنامج بحيث نستخدم الإجراءات فالهدف الأساسي هو تقليل حجم أجزاء الكود لسهولة اكتشاف الأخطاء وتعديل البرنامج كما يلي من النموذج العام:

```
; your code here
call input
call process
call output
; end program here
```

فيما يلي الكود الفعلي لهذه الطريقة :

```
;*****************************************
        ;
        ;    TASM KEYPRESS.ASM
        ;    TLINK /t KEYPRESS.OBJ
        ;
```

```
;*********************************************
        .model small
        .code
        .386
         ideal
         org    100h
start:  jmp    MAIN_PROGRAM
;----------------------
; Your Data Here
;----------------------
CopyMsg    db     'Any message here',0Dh,0Ah,'$'
PressEnter  db     0Dh,0Ah,'$'
;----------------------
; Your Procedures Here
;----------------------
proc PrintString          ;DX  طباعة النص الموجود بالمسجل
mov ah, 09h               ; كود الطباعه عن طريق الدوس
int 21h
ret
endp PrintString          ;
;----------------------
proc PrintChar            ;DL  طباعة الحرف الموجود بالمسجل
mov    ah,6          ;06h  طباعة حرف عن طريق امر الدوس
int    21h
ret
endp PrintChar            ;
;----------------------
proc disp_string          ; طباعة النص المخزن
mov dx, offset CopyMsg    ; اخبار الدوس اين يوجد النص
call PrintString
```

```
mov dx, offset PressEnter  ; اخبار الدوس عن مكان طباعة
                           بداية سطر جديد;          النص المسئول عن
call PrintString
ret
endp disp_string
;----------------------
proc GetInput              ; اختبار أدخال بيانات صحيحه  من
المستخدم
@@Loop:
;AL تخزين مفتاح من المستخدم في المسجل
mov   ah,8                 ; امر الدوس الخاص بالحصول على
مفتاح من المستخدم
int   21h
mov   bl, al ;bl تخزين المفتاح حتى يمكن استخدمه مره أخرى
مؤقتا في المسجل
                           ;CMP لان قيمة AL تتغير عند استخدام
المسجل
cmp   bl, 'Y'
je    @@Done
cmp   bl, 'N'
je    @@Done
cmp   bl, 'y'
je    @@Done
cmp   bl, 'n'
je    @@Done
; اطلاق صفاره عند الضغط على أي مفتاح اخر
mov   dl, 07h              ;dl نسخ كود الصفاره إلى المسجل
call  PrintChar
jmp @@Loop
```

```
@@Done:
اظهرالحرف الذي قام المستخدم بأدخاله ;
 mov     dl, bl
 call    PrintChar
 ret
 endp GetInput
;----------------------
MAIN_PROGRAM:
;---------------
; Your Code Here
;---------------
call disp_string
call GetInput
;---------------
mov al, bl        وضع AL ليكون هو كود الخروج من البرنامج;
قيمة المفتاح في المسجل
mov ah,4ch        الخروج للدوس ;
int 21h
end    start
```

استخدام معاملات سطور الأوامر

تابع المثال التالي:

```
;*********************************************
; Compile with:
;
;     TASM CLINE1.ASM
;     TLINK /t CLINE1.OBJ
;
; Test run it like this:
;
;     CLINE1 abcd efgh ijklmn
;     CLINE1 abcd EF$ ghi
;     CLINE1/abcdefg  ← note the '/' is a special case
;
; ever wonder when you type "tlink /t filename"
; exactly where is stores the "/t" and "filename"
; so you can use that knowledge in your own code?
;
;*********************************************
; just a simple com program here…
COM_PROG  segment byte public
      ideal
      assume  cs:COM_PROG
      org     100h
     start: jmp     MAIN_PROGRAM
;----------------------
; Your Data Here
;----------------------
         CommandLen  db      0
         CommandLine db      128 dup (0)

;----------------------
```

```
MAIN_PROGRAM:
;---------------
; Your Code Here
;---------------
وضع القيمه صفر للمسجل CX;
xor cx, cx
الحصول على طول النص;
mov cl, [80h]
تخزين طول النص;
mov [CommandLen], cl
إذا كان CX = 0 يتم القفز;
jcxz  @@Done
```

اى سطر أوامر يقوم المستخدم بإدخاله يتم تخذينه في 81h بالذاكرة لذلك تستطيع دوال الاسيمبلى أن تقوم بقراءة النص المخزن هناك للحصول على أي معاملات للأمر مثل tlink /t filename وهذا الأمر الذي يقوم بترجمة كود الأسيمبلى وربط مكتبات اللغة بالكود الخاص بك وسوف نقوم هنا بمحاولة الحصول على المعامل filename

لاحظ أن 80h يحتوى على الطول الحقيقي لسطر الأوامر ولا يمكن أن يزيد نص الأمر عن 128 Byte لان هذه هي المساحة المحجوزة فقط من قبل المعالج وهذا هو سبب تحديد حجم buffer ليكون 128 Byte

لاحظ أننا قمنا بتخزين نص الأمر في مكان مؤقت Buffer حتى يمكن الحصول على هذا النص مرة أخرى لأن العنوان المخزن به هذا النص يتم تغييره باستمرار

```
mov si, 81h     ;المسجل المصدر si يتم تخزين معامل الامر به
```

```
تحديد المسجل الهدف الذي سيتم النقل اليه;
mov di, offset CommandLine
تنفيذ عملية نسخ النص;
cld

@@Loop:
; AL إلى المسجل SI نسخ بايت من النص الموجود بالمسجل
lodsb

cmp al, ' '     ; (20 = اختبار إذا كان البايت عباره عن مسافه (كود
jne @@NotSpace      ; if not equal, إذا لم يساوى اقفز
mov al, 0           ; و الا قم بتخزين القيمه صفر
@@NotSpace:

; DI نسخ حرف من المسجل AL إلى النص الموجود بالمسجل
    stosb
انقاص قيمة CX والبدء في دوره أخرى للحلقه;
loop @@Loop
فى هذا الجزء من البرنامج نكون قد اكملنا عملية نسخ النص بالكامل;

inc di
وضع العلامة $ في آخر النص حتى يستطيع الدوس طباعته;
mov al, '$'
stosb

و يمكنك طباعة النص عن طريق الجزء القادم للتأكد من انه النص الصحيح;
mov dx, offset CommandLine
mov ah, 09h
```

```
int 21h
```

لاحظ هنا أننا قمنا بكتابة الكود ليتعامل مع الحرف $ حتى لا يختلط الأمر على المعالج في حالة إدخال سعر مثلا فسوف يقوم المعالج باعتبار أن هذا الحرف يعبر عن نهاية النص ولن نستطيع رؤية بقيته

وهذا يوضح أنه يمكننا أن نقوم بعمل أي شيء بهذه اللغة القوية كما يمكننا أن نقوم بطباعة أي نص بناء على أمر لاختبار البرنامج

ومن المفيد أيضا استخدام 0 الذي يعبر عن نهاية النص واستبداله بكود ascii 30 حتى لا يعتبر المعالج الرقم 0 عند إدخاله هو نهاية النص

ولتحسين عمل برنامج قمنا باختبار الحرف 09 وهو حرف tab وذلك ف حالة ما إذا ضغط المستخدم على المفتاح tab بدلا من المفتاح spacebar

```
@@Done:
```

و الآن بعد أن قمنا بتخزين النص الخاص بمعامل الأمر في الذاكرة يمكننا الآن أن نقوم بإنشاء داله تقوم بالتوصل إلى المعامل بعد وجود حرف مثل / أو –

```
;----------
الخروج للدوس;
mov ah,4ch
int 21h

ends COM_PROG

end    start
```

المزيد من الاسمبلى

لا تيأس إذا لم تستطع فهم الأمثلة السابقة 100 % أن أفضل طريقة يمكن بها أن تتعلم الاسيمبلى هو عن طريق النظر إلى بعض الأمثلة البسيطة للكود الذي يحتوى على تعليقات وافية من كاتب الكود ومن أفضل المصادر الذي يمكن الحصول منها على حل أي مشكلة هو الانترنت ولكن من المفيد أيضا مطالعة بعض الأمثلة الموجودة بالكتب الشهيرة مثل كتاب بيتر نورتن .

وسوف نتناول في هذا الفصل المواضيع التالية :

- التعامل مع النصوص
- التعامل مع الأرقام

التعامل مع النصوص

سنقوم هنا بشرح مثالين يقومان بطبع النصوص بطريقة مختلفة بحيث يمكنك استخدام هذه الأمثلة في أي مكان بالبرامج الخاصة بك

وسنقوم بادراج هذا في برنامج كامل حتى تستطيع أن ترى سهولة دمج الاكواد

- **باستخدام دوال الدوس DOS**

للنداء على روتين الطباعة :

```
message   db    'hello world','$'
mov       dx,   offset message
call      DisplayString
```

لاحظ هنا أن هذا الروتين صغير لان الدوس لا يهتم بالتفاصيل

سنقوم الآن بإخراج النص المخزن بالمسجل dx عن طريق أمر الطباعة ah=9

```
DisplayString:
   mov    ax,cs
   mov    ds,ax
```

```
mov    ah,9    ; دالة دوس
int    21h     ; النداء على مقاطع الدوس رقم 21
ret
```

- باستخدام دوال البايوس BIOS

هذه هي الطريقة الثانية لطباعة النصوص فبدلا من استخدام أمر الدوس int 21 سنقوم باستخدام أمر البايوس int 10 وسبب تعلمنا باستخدام دوال البايوس هو وجود العديد من البرامج لا تعتمد على الدوس وبدون هذا الكود يكون من المستحيل كتابة برامج لا تعمل على نظام التشغيل دوس مثل النظام لينيكس linux

للنداء على هذا الروتين نقوم بكتابة

```
message   db    'hello world','$'
mov       dx,   offset message
call      BiosDisplayString
```

الكود الخاص بالطباعة:

سنقوم الآن بإخراج النص المخزن في المسجل dx باستخدام الكود ah=14

```
BiosDisplayString:

        mov    si, dx  ; المسجل si مناسب للبايوس أكثر
        mov    ax, cs  ; استخدام المقطع الحالي
        mov    ds, ax  ; لعرض النص

bnxtchar:   lodsb      ; الحصول على الحرف الثاني من النص
```

```
        push    ax      ;تخزين القيمة ax حتى لا تتغير فيما بعد
المسجله

        cmp     al, '$' ; نهاية النص
        jz      endbprtstr
        pop     ax      ;ax الحصول على القيمة المخزنة في المسجل

        call    BiosDisplayChar

        jmp     bnxtchar

endbprtstr:     pop     ax      ;ax تمهيد قيمة المسجل

        ret

; لاحظ اننا لابد من عرض النص حرف حرف في كل مره
BiosDisplayChar:                ;al اخراج الحرف في المسجل

        mov     ah, 0Eh ; دالة البيوس المسئولة عن الطباعة
        xor     bx, bx
        xor     dx, dx
        int     10h     ; النداء على مقاطع البيوس
        ret
```

كما يوجد هناك أوامر أخرى غير الأمر int 10 لعرض النصوص تتضمن استخدام المسجل ah مع العناوين المقاطعة interrupt services وهى 09h و13h ويمكنك الحصول على شرح وافى لجميع المقاطع وأيضا أوامر ومسجلات الاسمبلى من الملف HelpPC

وتعتبر عناوين المقاطعة من الأشياء الهامة جدا عند فك حماية البرامج ذات الشاشات الافتتاحية المعروفة باسم nagscreens

التعامل مع الأرقام

سنتعلم هنا كيفية إظهار الأرقام بأي نظام عن طريق أوامر الاسمبلى

– للنداء على الروتين المسئول عن طباعة الرقم :

```
mov     ax, 0402h
call    DisplayWord
```

– كود الروتين المسئول عن الطباعة:

هذا الروتين يقوم بعرض رقم صحيح word مخزن بالمسجل AX كما يلى:

```
DigitBase     dw     010h  ; استخدام النظام السداسى عشر
```

إذا أردت طباعة الأرقام بالنظام العشري فقم باستبدال 010h بالرقم 0Ah;

```
DisplayWord    proc    near
        mov     si,offset DigitBase
        mov     di,offset TempNum
NextDgt:
        xor     dx,dx
        div     si
        add     dx,30h        ;ascii التحويل إلى النظام
            mov     [di],dl
            dec     di
          cmp     ax,0        ; اختبار إذا كان هناك حدود باقية في
                              الرقم يجب التعامل معها
        ja      NextDgt

            inc     di
```

```
        mov     dx,di
        mov     ah,9
        int     21h     ;dx يتم طباعة النص في المسجل
        retn
DisplayWord     endp
```

خلفية سريعة عن لغة C

من المفيد أيضا اكتساب بعض المهارات الخاصة بلغة السى وسنبدأ كالعادة بطبع رسالة "Hello World" ويتم ذلك عن طريق استخدام عبارة بسيطة تسمى printf وتستخدم في داخل برنامج السى كما يلي:

```
/* هذا السطر تعليق */
   main()
   {
     printf("Hello World!");
   }
```

أول سطر بالبرنامج السابق هو تعليق فكل لغة لها العلامة الخاصة بها لإدراج سطور التعليق ويمكن عن طريق العلامتين /* */ إدراج تعليق مكون من عدة سطور أما إذا أردت إدراج تعليق لسطر واحد فيمكنك استخدام العلامة ' بكل بساطه أول السطر

و بعض المعالجات الشهيرة مثل بورلاند سى وميكروسوفت فيجوال سى تستخدم العلامتين // للتعليق في سطر واحد

السطر التالي يحتوى الدالة ()main التي لا تحتوى على أي معاملات والدالة هي مثل الإجراءات تحتوى على كود يمكن النداء عليه مرات عديدة من أي مكان بالبرنامج وأحيانا تحتاج الدوال إلى معاملات parameters لكي يمكن النداء عليها فقد تتوقع الدالة أن يتم تمرير رقم أو قيمه معينه إليها حتى تقوم بإجراء معالجه من نوع معين لهذه القيمة الممررة ويمكن التعبير عنها بالصيغة التالية

Return-type Function-name(Parameter-list)

```
{
  /* your code here */
}
```

لاحظ أن يتم تحديد نوع القيمة التي سوف تقوم الدالة بإرجاعها قبل اسم الدالة كما بالصيغة السابقة ثم نقوم بالإعلان عن المعاملات بين القوسين

وكقاعدة عامة لا يمكن كسرها فأي برنامج مكتوب بلغة السى يجب أن يكون به داله اسمها main فنظام التشغيل يقوم بالبحث عن هذه الدالة حتى يقوم بتنفيذ تعليمات البرنامج

السطر التالي يحتوى على عبارة الطباعة

```
printf("Hello World!");
```

تعتبر هذه العبارة من العبارات القياسية في لغة السى وتعنى طباعة سطر من النصوص ويتم وضع النص داخل علامتي اقتباس وينتهي دائما أي سطر بلغة السى بالفاصلة المنقوطة ; التي تعنى نهاية السطر للمعالج وإلا يتوقف المعالج ويقوم بإظهار رسالة خطأ.

ويمكن عن طريق العبارة printf طباعة النصوص بعدة طرق مختلفة وبالنسبة للأقواس { } فهي ضرورية حتى يعرف المعالج أين تبدأ الدالة وأين تنتهي.

كما يمكن للعبارة printf طباعة نصوص خاصة عن طريق استخدام حروف تسمى escape characters وتؤدى مثلا لنقل مؤشر الكتابة إلى مكان آخر أو بدء سطر جديد وهكذا وفيما يلي مثال يوضح ذلك:

```
/* escape character program */
main()
{
  printf("Hey, look at me\n");
  printf("I'm learning\t C!");
}
```

لاحظ من المثال السابق أن العلامة n\ تؤدى إلى تحريك المؤشر في سطر جديد أما العلامة t\ تقوم بعمل مفتاح المحاذاة TAB فيقفز مؤشر الكتابة إلى مسافة ثم يتم طبع حرف C

سنقوم الآن بكتابة برنامج يقوم بطبع قيمه من متغير بطريقة بسيطة كما يلى:

```
/*   استخدام المتغيرات */
void printStuff();
main()
{
  printStuff();
}
void printStuff()
{
  int x;
  x = 10;
  printf("The variable x is equal to %d", x);
}
```

يتضح من المثال السابق كيف يتم النداء على الدالة printstuff() لطباعة قيمة المتغير x ولاحظ أننا قمنا أولا بالإعلان عن الدالة printstuff قبل الدالة main ووضع التعبير void أمام ها يعنى أن هذه الدالة لا تقوم بإرجاع أي شئ

لاحظ أننا قمنا بتعريف المتغير x عن طريق تحديد نوع المتغير وهو عدد صحيح بالتعبير int x ويوجد أنواع أخرى للمتغيرات ليست مجالنا الآن وبعد ذلك نقوم بوضع القيمة 10 في المتغير

بالنسبة للعبارة printf فيمكنها أن تحتوى على معاملين أو أكثر الأول دائما يحتوى نص وحروف الهروب الخاصة ومعنى وجود الحرف %d هو طبع قيمة

متغير من نوع عددي في هذا المكان ثم نقوم بوضع فاصله لفصل المعاملين ووضع المتغير المراد طبع قيمته .