لغة التجميع

مدخل إلى لغة التجميع (معالجات إنتل)

ترجمة: تواتي عمر

مستغانم، يوم 28 أغسطس 2002

السلام عليكم

رغبة في الإفادة والاستفادة، هاأنذا أقدم لكم مجموعة من الدروس البسيطة في لغة التجميع، ولقد قمت بترجمة هذه الدروس من الفرنسية إلى العربية بمبادرة شخصية، والهدف الأسمى من ورائها هو إيصال العلم إلى طالبيه بالعربي، وما أكثرهم. فقط أتمنى أن تنال رضاكم، وتحظى باهتماماتكم.

هذا ملخص بسيط للغة التجميع، وإن لم نقم بشرح وافي لكل المعلومات، إلا لأننا فرضنا علم الطالب ببعض الأساسيات حول هذه اللغة. ومع ذلك، فإننا لم ننسى تقديم بعض الشروحات للمبتدئين الذين لا يعلمون شيئا عن هذه اللغة.

الملخص:


- مدخل عام إلى لغة التجميع:
- معجم التجميع
- قائمة السجلات
- قائمة الرايات
- قائمة التعليمات
- قائمة المقاطعات
- أمثلة عامة


## *مدخل عام إلى لغة التجميع:*

معالج أي حاسب لا يفهم أية لغة، لا الباسكال ولا السي ولا الجافا، ولا حتى التجميع في حد ذاته، إنما يفهم شيئا واحدا، هو لغة الآلة. إنها قائمة من ثمانية أعداد تسمى البتات، وتكون مقدمة في النظام الست عشري، على شاكلة "B0h 12h". ومنه يمكننا القول أن التجميع هو إصدار يتوافق مع فهم البشر للغة الآلة.

المثال السابق يعطينا "mov al, 12h"، ومعناها نسخ القيمة 12h في السجل AL.

كما يمكنك ملاحظته، إنها اللغة البرمجية الأكثر قربا إلى المعالج (إلا في حالة قدرتك على البرمجة بلغة الآلة نفسها).

سؤال قد يتبادر إلى ذهنك: **فيم تستعمل؟** الأهمية الأولى تتمثل في السرعة، حيث أنها اللغة التي بفضلها يمكننا تحقيق البرامج الأكثر سرعة، وخاصة في ميدان الرسومات.

ثانيا، يمكنك الوصول إلى مقاطعات الدوس (interruptions)، والتي تسمح بالوصول المباشر إلى العتاد، كالفأرة أو الشاشة أو حتى بطاقة الفيديو. لأن المترجم (compiler) لا يعمل إلا على ترجمة البرنامج المكتوب بلغة يفهمها الإنسان إلى لغة التجميع، وهذه العملية يمكن القيام بها يدويا بعد تدريب طويل في هذا الميدان. كما يتيح لنا التجميع إمكانية معرفة ما يتوفر عليه ملفنا الثنائي (ملف تنفيذي exe. أو com. في بيئة مايكروسفت).

بعض المتمرسين في البرمجة وخاصة القراصنة، لا يستخدمون إلا التجميع في برامجهم، والسبب هو إنجاز برامج سريعة وصغيرة، وفي نفس الوقت تحقيق مرادهم من خلال قرصنة البرامج أو ببرمجة ما يرفضه المترجم كالفيروسات.

المختصون في البرمجة لا ينصحون أبدا بأن يقوم المبرمج بإنجاز برنامجه 100% بلغة التجميع، لأن الكود سيفقد مرونته وتكثر أخطاءه، مما يفقد السيطرة على برنامج يتكون من آلاف الأسطر، على اعتبار أن كل تعليمة تقع في سطر واحد. وإنما ينصح باستخدام التجميع لتحسين أداء بعض الدوال أو الإجراءات في لغة البرمجة التي تعتمدها (السي، الباسكال ...)، والتي تحتاج لسرعة قصوى، سواء لدوال تطلب لمرات عديدة، أو تلك المسؤولة عن القيام بوظائف معينة، كرسم خط، أو نسخ قطاع كامل من الذاكرة...الخ.

## معجم التجميع:

**المصرف**: هو برنامج يقوم بتحويل كود مصدري (ملف نصي، أما للتجميع، فالملف يكون ذو توسع asm.) إلى كود الآلة، يعني خلق ملف ثنائي (ملف ذو توسع obj. أو exe. أو com.).

**الست عشري (Hexadecimal)**: في نظام العد المتداول بين الناس، نستعمل الأرقام العربية، والمحصورة بين 0 والـ 9، إذن ما مجموعه 10 أرقام، والرقم الأكبر هو 9 ( 10-1). نقول إذن عن هذا النظام أنه نظام عشري decimal. فالعدد 451 يمكننا تفكيكه على شكل:

$$(1\times1) + (10\times5) + (100\times4) = 10\times1^{0} + 10\times5^{1} + 10\times4^{2}.$$

النظام الست عشري ينتهي عند 16، والسبب الذي دفع إلى ذلك هو تمثيل النظام الثنائي بأقل عدد ممكن من الأرقام. فكما نعلم جميعا، فالنظام الثنائي يتكون من رقمين فقط، وهما الـ 0 والـ 1، ولتكوين أي رقم آخر، فإننا نكتب متوالية من الأصفار والوحدات على شاكلة 01101001 وهذا الرقم يمثل 105 في نظام عدنا الذي نستعمله يوميا، ولكن لتمثيل أعداد كبيرة نحتاج إلى أصفار ووحدات كثيرة، مما يسبب العديد من الأخطاء ويضيع الوقت، ويعقد الفهم، لهذا جاء دور النظام الست عشري ليحل الإشكال، ويعوض كل أربعة أرقام ثنائية برقم ست عشري واحد.

01101001 = 1001 و 0110 = 9 و 6 = 69 في النظام الست عشري.

النظام الست عشري يبدأ من الـ 0 وينتهي عند الـ 15، ولكن الأرقام الستة الأخيرة (من 10 إلى 15) تتكون من وحدتين، وهذا ما سيخلط الأمور، لهذا تم تعويضها بحروف، كما يشير الجدول الآتي إلى ذلك:

| النظام العشري | النظام الثنائي | النظام الثماني | النظام الست عشري |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 3 | 3 |
| 4 | 100 | 4 | 4 |
| 5 | 101 | 5 | 5 |
| 6 | 110 | 6 | 6 |
| 7 | 111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

ملاحظة: يشار للرقم في النظام الست عشري بعدة رموز:

ففي لغة السي نشير إلى الرقم بالسابقة 0x، أما في لغة الباسكال، فيسبق الرقم رمز الدولار، وفي لغة التجميع يشار إلى الرقم أنه ست عشري من خلال اللاحقة h

Ox14 في لغة السي = $14 في لغة الباسكال = 14h في لغة التجميع.

ملاحظة: لغويا، العدد هو الكتابة الحرفية للرقم، والرقم هو الكتابة الرمزية للعدد.

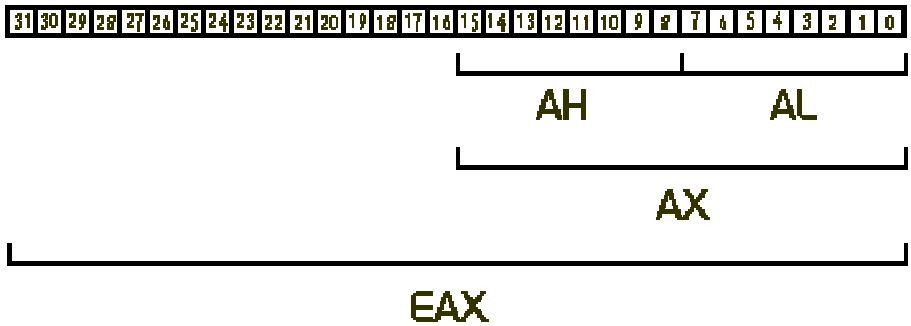
**البايت (Byte)**: البايت هو متوالية من ثمانية بتات (8 bits)، فإذا كان البايت مرمّزا ( signed) فإنه يمكنه أخذ قيمة محصورة بين –128 و +127 (128= $2^{8-1}$ = $2^{7}$ ، و127= $2^{(8-1)-1}$)، إذن تحتل القيم المحصورة 7 بتات فقط، والبت الأخير يستخدم للإشارة (السلب أو الإيجاب)، إما إذا كان البايت غير مرمّز فإنه بإمكانه أخذ قيمة محصورة بين 0 و 255، أي 256 حالة = $2^{8}$.

**الكلمة (Word)**: بما أن البايت يشغل 8 بتات، فإن الكلمة تشغل 16 بتا، وبالتالي يمكن القول أن الكلمة تحوي 2 بايت.

إذا كانت الكلمة مرمزة، فإنها قادرة على احتواء قيمة محصورة بين –32768 أي ($2^{15}$)، و 32767 أي ($2^{15}$ - 1)، أما إذا كانت غير مرمزة، فالقيمة المحتواة تكون محصورة بين 0 و65535 ($2^{16}$ -1).

**الكلمة المضاعفة (Double Word أو DWord)** : وتشغل الكلمة من هذا النوع قيم ذات 4 بايت. إذن إذا كانت مرمزة، فإنها تكون بين –2147483648 أي ($2^{31}$)، و + 2147483647 ($2^{31}$ –1)، أما إذا كانت غير مرمزة، فبإمكانها حصر قيمة تتراوح بين 0 و 4294967295.

**السجل (Register)**: هو فراغ ذاكري موجود بصفة فيزيائية في قلب المعالج ( Processor)، والذي تحفظ فيه القيم أثناء معالجتها. في البداية كانت السجلات تعمل على ثمانية بتات (بايت واحد)، وكانت تسمى آنذاكL? (حيث يمكن للرمز "? " أن يكون A أو B أو C أو D)، ثم تطورت السجلات إلى 16 بت (كلمة)، وصارت مسماة بـX? ، الشيء العجيب في هذا، هو أننا لو قمنا بتغيير قيمةL? فإننا نكون بذلك قد غيرنا الجزء المنخفض من السجل X? . بمعنى آخر،X? مكون من بايت ومنL? (ويقال أيضا من جزء مرتفع وجزء منخفض). ثم بعد وصول معالجات 386، تم الانتقال إلى سجلات 32 بت، وصارت بالتالي تسمى بـ E? X، وهنا إذا ما قمنا بتغيير قيمةL? أوX? فإننا نكون بذلك قد قمنا بتغيير الجزء E? X.




السجلات القاعدية هي السجلات AX، BX، CX و DX، وهي سجلات ذات 16 بت (= 2 بايت)، وأجزاءها المنخفضة (Low) هي AL، BL، CL و DL أما أجزاءها المرتفعة (High) فهي AH، BH، CH و DH.

إذا كان AX معدوما (null)، فهذا يعني أن AL =0 و AH =0 أيضا. وإذا ما قمنا بغيير قيمة AX فإننا نغير قيمتي AH و AL معا، أما لو قمنا بتغيير قيمة AL فهذا يعني أن قيمة AX تغيرت أما قيمة AH فمازالت كما هي. (الرسم السابق يوضح الصورة أكثر).

لنلاحظ هذا المثال:

al = 15
ah = 10
=> ax = al + ah*256 = 15 + 10*256 = 2575

لمزيد من التفاصيل راجع قائمة السجلات.

**القطاع والفرع (Segment & Offset)**: العناوين الذاكرية (مواقع البايتات على صفائح الذاكرة) هي معرفة بسجّلين، القطاع (الجزء المرتفع من العنوان)، وبالفرع (الجزء المنخفض من العنوان). نشير أن العنوان يحدد بـ Segment :[Offset] (حيث أن النقطتين "**:**" هما الفاصل، في حين يعتبر المجالان "**[**" و "**]**" غير إجباريين). مثال: DS :[DI]

مع وصول الـ 386، تطورت الفروع إلى 32 بت، من أجل الوصول إلى أقصى الذاكرة (16 ميغا بايت)، وظهرت بذلك الفروع بالسابقة E، مثل (ESI, EDI, ESP, EBP)، هذا بالنسبة لسجلات 32 بت، ونفس الشيء كان مع سجلات 16 بت، حيث كانت الفروع SI, DI, SP, SB. لمزيد من التفاصيل، راجع قائمة السجلات.

**المقاطعة (Interruption)**: يمكن وصف المقاطعة على أنها برنامج صغير مخزّن في الذاكرة، ويتم طلبه عدة مرات لأداء مهمة محدّدة ، وبوثيرة متزايدة أو متناقصة. يوجد على الأكثر 256 مقاطعة، فالأولى منها مدمجة في البيوس (BIOS، وهو برنامج ينفذ عند تشغيل الحاسب، والذي يسيّر العتاد: القرص الصلب، الوصول إلى الذاكرة ...)، مثل مقاطعات بطاقة الفيديو ولوحة المفاتيح.

بعض المقاطعات يتم طلبها بوثيرة ثابتة: المقاطعة 1Ch مثلا والتي هي عدّاد يزيد في القيمة 18.6 مرة/ثانية (أي يضيف 1 للقيمة المعنية). البعض الآخر يتم طلبه فقط في حالة ما إذا أردناها: المقاطعة 09h الخاصة بلوحة المفاتيح، يتم طلبها في كل مرة يقوم المستخدم بالضغط على زر ما أو عند تحريره.

**المكدس (Stack)**: في المكدس نقوم بإضافة القيم بفضل التعليمة PUSH، ونقوم بحذفها من خلال التعليمة POP. في قمّة المكدس نجد قيمة السجل [E]BP. ويمكننا بذلك معرفة ما هو موجود في الطابق X من خلال التعليمة "mov AX, [Bp-4]"، ولكن ليس لحذف هذه القيمة، لأننا لو قمنا بحذف قيمة من وسط المكدس، فإن هذه الأخيرة تتقهقر، وهذا ما يؤدي إلى توقف الحاسب عن العمل.

**الراية (Flag):** الرايات هي بتات موجودة بداخل المعالج. راجع قائمة الرايات.

## قائمة السجلات:

يوجد العديد من السجلات، ولكل سجّل دور محدد على الأقل (هناك بعض السجّلات لها عدة أدوار وعدة مهام).

**AL/AH/EAX**: هو السجل العام، والذي تتغير قيمته بسرعة أثناء عمل الحاسب.

**BL/BH/EBX**: هو أيضا سجل عام، يمكن استخدامه كفرع للذاكرة (Offset)، كمثال:

"mov al, byte ptr ds:[bx+10]".

**CL/CH/ECX**: عامة، يستخدم هذا السجل كعداد للحلقات (loops)، كمثال:

"mov ecx, 5; rep movsd"، ويعمل هذا المثال على نسخ 5 كلمات مضاعفة.

**DL/DH/EDX**: هذا الأخير هو أيضا سجّل عام، وهو إجباري عند العمل للوصول إلى المداخل(ports)، (المداخل هي واسطة للتواصل بين مختلف رقائق الحاسب، مثل المدخل 42h و 43h تستعمل لمراقبة مكبرات الصوت الداخلية).

**CS**: قطاع ذاكري مخصص للكود (Code Segment).

**DS**: قطاع ذاكري مخصص للبيانات (Data Segment).

**ES**: قطاع ذاكري.

**FS**: قطاع ذاكري آخر.

**GS**: قطاع ذاكري آخر.

**SS**: قطاع ذاكري خاص بالمكدس (Stack Segment).

**BP**: فرع (Offset) ذاكري، وغالبا هو نسخة من السجل SP، حيث يمكننا طرح قيمة من أجل قراءتها في المكدس. (لا يجب علينا تغيير محتوى SP).

**EDI/DI**: فرع ذاكري يستخدم من قبل ES (أو FS أو GS إذا ما تم تحديده)، كمثال:

"mov al, byte ptr gs:[10]"

**EIP/IP**: فرع ذاكري خاص بالكود (غير يمكن الوصول إليه مباشرة، لكن يمكن تغيير محتواه بطريقة مباشرة من خلال التعليمات: CALL, JMP أو J[case] ، حيث case هي حالة من حالات القفز التي سنراها لاحقا).

**ESI/SI**: فرع ذاكري يستخدم من قبل DS

**ESP/SP**: فرع ذاكري يستخدم من قبل المكدس (Stack).

الجدول الآتي يلخص أهم سجلات 16 بت، وحجم كل سجّل:

| *Register* | *Bits 7..0* | *Bits 15..8* | *Main Function* |
|---|---|---|---|
| **AX** | AL | AH | Accumulator |
| **BX** | BL | BH | Base |
| **CX** | CL | CH | Count |
| **DX** | DL | DH | Data |
| **SI** | - | - | Source Increment |
| **DI** | - | - | Destination Increment |
| **SP** | - | - | Stack Pointer |
| **BP** | - | - | Base Pointer |
| **CS** | - | - | Code Segment |
| **DS** | - | - | Data Segment |
| **SS** | - | - | Stack Segment |
| **ES** | - | - | Extra Segment |
| **IP** | - | - | Instruction Pointer |

قائمة الرايات Flags:

**AF**: (Auxiliary Flag): دليل حفظ إضافي.

**CF**: (Carry Flag): دليل حفظ.

**DR**: (Direction Flag): دليل اتجاه معالجة حزم الرموز.

**IF**: (Interrupt Flag): دليل تنفيذ المقاطعات، والمسماة بالمقنّعة.

**OF**: (Overflow Flag): دليل الفائض.

**PF**: (Parity Flag): دليل الزوجية: PF=0 يعني فردي، PF=1 يعني زوجي.

**SF**: (Sign Flag): دليل الإشارة: SF=0 يعني موجب، SF=1 يعني سالب.

**TF**: (Single Step Flag): دليل عملية تصحيح الأخطاء.

**ZF**: (Zero Flag): دليل القيمة المعدومة.

## قائمة التعليمات:

- LDS – LEA – J[case] - JMP – IRET – IN – CMPS – CMP – CALL – AND – ADD - OR – NOT – MUL - MOVZX – MOVS – MOV – LODS – LSS – LGS – LFS - LES – SUB – STOS – SCAS – SHR – SHL – SCAS – RET – REP – POP – PUSH - OUT XOR - TEST

المجالين "**{**" و "**}**" يحددان البارامترات. والتي قد تكون سجّلا أو منطقة ذاكرية أو قيمة فورية (Immediacy)، وهي في الغالب عدد.

اللاحقة "[B أو W أو D]" في نهاية التعليمة تحدد أسلوب معالجة التعليمة.

"B" : تعني هذه اللاحقة أن التعليمة من نوع 8 بت (Byte)، وهذا ما يجعل استعمال السجل AL ضروريا.

"W" : تعني هذه اللاحقة أن التعليمة من نوع 16 بت (Word). استعمال السجل AX ضروري.

"D" : تعني هذه اللاحقة أن التعليمة من نوع 32 بت (DWord) . استعمال السجل EAX ضروري.

التعليمة ADD:

```
ADD {destination},{source} // ADD {الوجهة},{القناع}
```

تقوم هذه التعليمة بإضافة قيمة المصدر لقيمة الوجهة، وينجم عن هذه العملية تغيير في محتوى الرايات، ويتم تخصيص ناتج الجمع للمعامل الأول.

قد يكون أحد المعاملين سجلا، كما قد يكون المعامل الثاني مرجع ذاكري (متغير أو عنوان ذاكري) أو immediate (رقم مباشر).

```
add ax, bx // => ax=ax+bx // (reg, reg)  كلا المعاملين سجّلا
add ax, 5 // => ax=ax+5 // (reg, imm)  المعامل الثني هو عدد مباشر
add ax, var// => ax=ax+var // (reg, mem)  المعامل الثني هو مرجع ذاكري
add ax, [si] // => ax=ax+[si] // (reg, imm)  المعامل الثني هو مرجع ذاكري
```

التعليمة AND:

```
AND {destination},{mask} // AND {الوجهة},{القناع}
```

لتبسيط هذه التعليمة، ووفقا جدول الحقيقة الذي سنطّلع عليه لاحقا، فإن الناتج صحيح (=1)، إذا كان كلا المعاملين صحيح، والناتج يكون خاطئا (=0) إذا كان أحد المعاملين خاطئا أو كلاهما.

وبمفهوم آخر أكثر وضوح، تقوم هذه التعليمة بجعل كل بت في الوجهة مساويا للصفر إذا كان القناع مساويا للصفر، في حين لا تغير بت الوجهة إذا كان القناع مساويا للواحد.

حسب لغات البرمجة المعروفة يمكننا تمثيل ذلك بـ:

```
Destination := Destination AND Mask ; // بلغة الباسكال
Destination &= Mask ; // بلغة السي
```

التعليمة CALL:

```
CALL {address}
```

تطلب الإجراء (procedure) أو الدالة (function) المتواجد في العنوان المحدد. فإذا سبقت التعليمة CALL عدة تعليمات من نوع PUSH، فهذا يعني أنها مجموعة من البارامترات تم تخزينها في المكدس (Stack). وفي هذه الحالة ، فإن الإجراء يبدأ بـ:

```
"push [e]bp ; mov [e]bp, [e]sp "
```

ويمكننا أيضا إيجاد قراءة للبارامترات على شكل:

```
" mov {Register}, [ebp – {Value}] "
```

وأخيرا لا يجب نسيان التعليمة "RET [{Value}]" في نهاية الإجراء.

التعليمة CMP:

```
CMP {a},{b}
```

تقوم هذه التعليمة بمقارنة المتغير a مع المتغير b. وبعدها نجد مباشرة قفزا مشروطا نحو فرع ذاكري آخر "jmp أو je أو jne ...، بصفة عامة j[case] {Offset}".

التعليمة CMPS:

```
CMPS [B/D/W]
```

هذه التعليمة تقارن كل بايت/كلمة/كلمة_مضاعفة من القطاع والفرع الذاكريين DS :ESI مع القطاع والفرع الذاكريين ES :EDI.

التعليمة IN:

```
IN {destination},{port}
```

هذه التعليمة تقرأ قيمة ذات 8 بت على المنفذ {port}، وتخزنه في الوجهة {destination}. أما السجل الوحيد المسموح باستعماله في هذه العملية فهو السجل DX.

التعليمة IRET:

```
IRET {value}
```

هذه التعليمة تسمح بمغادرة المقاطعة، وتستعمل فقط للبرامج المستقرة في الذاكرة، مثل سائق (driver) الفأرة.

التعليمة JMP:

```
JMP {offset}
```

هذه التعليمة تمكن من القفز إلى العنوان الذاكري المحدد بالفرع {offset}.

التعليمة J[case]:

```
J[case] {offset}
```

إذا كانت الحالة [case] صحيحة، فإن التعليمة تنفّذ، وينتقل العمل بذلك إلى العنوان المحدد في الفرع {offset}. الحالة [case] هي شرط متعلق بالرايات (flags)، ويتم تنفيذ الشرط وفق قواعد المنطق المعروفة.

الحالات الممكنة لـ case، بعد أخذ "CMP {a},{b} " كمثال:

غير مؤشرة not signed:

JA : يتم القفز إذا كان أكبر، في مثالنا b < a، إذا كانت الراية (الدليل) 0=ZF=CF, JAE
JNB, JNC : إذا كان أكبر أو يساوي (b=<a)، إذا كان 0 = CF

JB, JC : أصغر (b>a)، إذا كان 1=ZF=CF

JBE : أصغر أو يساوي (b=>a)، إذا كان OF<>CF و 1=ZF.

مؤشرة signed:

JG : أكبر (b<a)، إذا كان 0=ZF=SF

JGE : أكبر أو يساوي (b=<a)، إذا كان OF = SF

JL : أصغر (b>a)، إذا كان OF <> SF

JLE : أصغر أو يساوي (b=>a)، إذا كان OF<>SF و 1=ZF.

متساوية:

JE, JZ : يساوي (b=a)، إذا كان 1=ZF.

JNE, JNZ : يختلف عن (b <> a)، إذا كان 0=ZF.

التعليمة LEA:

```
LEA {destination},{source}
```

يتم هنا كتابة عنوان المصدر {source} في الوجهة {destination}.

هذه التعليمة تكافئ:

```
MOV {destination},Offset{source}
```

التعليمة LDS:

```
LDS {destination},{address}
```

نسخ عنوان {address} من نوع 32 بت في السجّل DS، (الذي هو قطاعه)، وفي {destination} من نوع 16 بت، (التي هي فرعه).

التعليمة LES:

```
LES {destination},{address}
```

نسخ عنوان {address} من نوع 32 بت في السجّل ES، (الذي هو قطاعه)، وفي {destination} من نوع 16 بت، (التي هي فرعه).

التعليمة LFS:

```
LFS {destination},{address}
```

نسخ عنوان {address} من نوع 32 بت في السجّل FS، (الذي هو قطاعه)، وفي {destination} من نوع 16 بت، (التي هي فرعه).

التعليمة LGS:

```
LGS {destination},{address}
```

نسخ عنوان {address} من نوع 32 بت في السجّل GS، (الذي هو قطاعه)، وفي {destination} من نوع 16 بت، (التي هي فرعه).

التعليمة LSS:

```
LSS {destination},{address}
```

نسخ عنوان {address} من نوع 32 بت في السجّل SS، (الذي هو قطاعه)، وفي {destination} من نوع 16 بت، (التي هي فرعه).

التعليمة LODS:

```
LODS [B/D/W]
```

نسخ البايت/الكلمة/الكلمة_المضاعفة ES :EDI في السجل EAX/AX/AL.

* التعليمة العكسية لها هي: "STOS [B/D/W] ".

التعليمة MOV:

```
MOV {destination},{source}
```

يتم هنا نسخ قيمة المصدر {source} في الوجهة {destination}.

التعليمة MOVS:

MOVS [B/D/W]

هذه التعليمة تنسخ البايت/الكلمة/الكلمة_المضاعفة من القطاع والفرع الذاكريين DS :ESI إلى القطاع والفرع الذاكريين ES :EDI.

التعليمة MOVZX:

MOVZX {destination},{source}

تقوم هذه التعليمة بتوسعة العدد الموجود في المصدر، والمشفر على 8 بت، ويتم تحويل الناتج إلى الوجهة بحجم 16 بت أو 32 بت، مثال:

MOVZX ax, al

يقوم هذا المثال بمحو الجزء المرتفع من السجل AX، (أي محو الجزء AH).

التعليمة MUL:

MUL {source}

تحدد هذه التعليمة جداء الوجهة المصرح بها من قبل في المصدر المشار إليه كبارامتر.

العددين (الوجهة والمصدر) يتم اعتبارهما غير مؤشرين (not signed). يمكن استعمال التعليمة JC {address} لتجربة الفائض.

كما يجب الإشارة إلى أن الوجهة يتم استخدامها بدلالة طول المصدر:

8 بت: الوجهة هي AX (العدد المضروب فيه هو AL).

16 بت : الوجهة هي DX :AX، هذا يعني أن AX يحوي الجزء المنخفض و DX يحوي الجزء المرتفع (العدد المضروب فيه هو AX).

32 بت : الوجهة هي EDX :EAX، هذا يعني أن EAX يحوي الجزء المنخفض و EDX يحوي الجزء المرتفع (العدد المضروب فيه هو EAX).

التعليمة NOT:

NOT {destination}

تسمح هذه التعليمة، والتي تعتبر أحادية وليست ثنائية (تحتاج لبارامتر واحد) بقلب بت الوجهة كما يشير إلى ذلك جدول الحقيقة.

التعليمة OR:

OR {destination},{mask}

تطبق هذه التعليمة " **أو** " منطقيا على الوجهة اعتمادا على القناع، فكل بت من الوجهة سيتم جعله مساويا للواحد إذا كان بت القناع مساويا للواحد، في حين يتم ترك بت الوجهة من دون تغيير إذا ما كان بت القناع معدوما. لاحظ جدول الحقيقة.

```
Destination := Destination or Mask ; // بلغة الباسكال
Destination |= Mask ; // بلغة السي
```

التعليمة OUT:

```
OUT {source},{port}
```

عكس تعليمة IN، فإن هذه التعليمة تكتب القيمة الموجودة في المصدر {source} (ذات 8 بت) على المنفذ {port}. أما السجل الوحيد المسموح باستعماله في هذه العملية فهو DX.

التعليمة PUSH:

```
PUSH {value}
```

تسمح هذه التعليمة بوضع القيمة {value} في المكدس {Stack}.

التعليمة POP:

```
POP {register}
```

تقوم هذه التعليمة بإخراج قيمة من المكدس ثم تخزنها في السجل {register}.

التعليمة REP:

```
REP {instruction}
```

تقوم REP بتكرار التعليمة {instruction} **نون** مرة، (حيث **نون** هو محتوى السجل ECX).

التعليمة RET:

```
RET {value}
```

تسمح هذه التعليمة بمغادرة الإجراء الساري.

إذا ما تم إرسال بارامترات إلى CALL، فإن [xxxx] هو عدد البايتات المرسلة الواجب إخراجها من المكدس.

التعليمة SHL:

```
SHL {register},{value}
```

تقوم هذه التعليمة بإزاحة ثنائية (binary) على جهة اليسار (L = Left) لمحتوى السجل {register} بمقدار {value} مرة، وبالتالي فإن البتات التي تظهر على اليمين يتم تعويضها بأصفار. مثال:

```
Mov al, 3;// تخزين القيمة 3 في السجل
Shl al, 2;// إزاحة مرتين لبتات العدد 3
```

ستكون نتيجة المثال إذن 12، لأن تمثيل 3 في الثنائي هو 00000011، إذن الإزاحة مرتين على اليسار تعطي 00001100، وهو العدد 12.

التعليمة SHR:

```
SHR {register},{value}
```

تقوم هذه التعليمة بإزاحة ثنائية (binary) على جهة اليسار (R = Right) لمحتوى السجل {register} بمقدار {value} مرة، وبالتالي فإن البتات التي تظهر على اليسار يتم تعويضها بأصفار. يمكن تطبيق نفس منهاج المثال السابق.

الإزاحة على اليسار تساوي الضرب في 2 نون مرة، والإزاحة على اليمين تساوي القسمة على 2 نون مرة،(حيث نون هو قيمة {value}).

التعليمة STOS:

```
STOS [B/D/W]
```

هذه التعليمة تنسخ EAX/AX/AL في البايت/الكلمة/الكلمة_المضاعفة الموجودة في القطاع والفرع الذاكريين ES :EDI (عكس التعليمة LODS).

التعليمة SUB:

```
SUB {destination},{source} // ADD {الوجهة},{القناع}
```

وبعكس عملية الجمع، فإن التعليمة sub تقوم بطرح قيمة المصدر من قيمة الوجهة، وينجم عن هذه العملية تغيير في محتوى الرايات، ويتم تخصيص النتيجة للسجل (المعامل الأول).

* تنطبق الخواص نفسها على كل من ADD، MOV و SUB.

التعليمة SCAS:

```
SCAS [B/D/W]
```

هذه التعليمة تقارن EAX/AX/AL بالبايت/الكلمة/الكلمة_المضاعفة الموجودة في القطاع والفرع الذاكريين ES :EDI (يمكن القول أنها تبحث عن قيمة في حزمة_رموز ( string)).

التعليمة TEST:

```
TEST {source},{mask}
```

تقوم هذه التعليمة تسمح باختبار بت خاص للمورد، وتعمل بالمقابل على تغيير الراية JZ، فإذا كان البتات فائضة أم لا.

بأسلوب آخر، تقوم التعليمة TEST {a},{a} إذا ما كان المتغير a معدوما أم لا.

التعليمة XOR:

```
XOR {destination},{mask}
```

تطبق هذه التعليمة " **أو المانع** = eXclusive OR" على الوجهة اعتمادا على القناع، فكل بت من الوجهة سيتم جعله مساويا للواحد إذا كان مخالفا لبت القناع ، ومساويا للصفر إذا كان موافقا لبت القناع. لاحظ جدول الحقيقة.

تقوم التعليمة XOR {a},{a} بجعل قيمة a معدومة، وتمتاز بسرعة أكبر من التعليمة

.MOV {a}, 0

```
Destination := Destination xor Mask ; // بلغة الباسكال
Destination ^ = Mask ; // بلغة السي
```

جدول الحقيقة:

| A | B | Not A | A AND B | A OR B | A XOR B |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |

## قائمة المقاطعات:

لقد وجدت هذه المعلومات من مصدرها المشار إليه، ومع أنها تظهر غامضة لأغلب المبرمجين، والمبتدئين، إلا أنني أنبهكم أن أغلب هذه المصطلحات الغريبة ستفهمون مدلولاتها بعد أن تطلعوا على مختلف الأمثلة التي سنسردها في نهاية هذه الدروس أو التي سنستقيها في الأيام المقبلة.

**01h** : تنفيذ برنامج ما في حالة خطوة خطوة (للسماح باكتشاف الأخطاء).

**02h** : مقاطعة غير مقنّعة.

**03h** : خطأ في الانفصال.

**04h** : خطأ نتيجة تعدّي الحجم، كمثال:

```
Mov al, 200;

Add al, 140;

==> 200+140 = 340 <==> 340>255.
```

**05h** : يتم من خلال هذه المقاطعة طبع الشاشة في شكل نص.

**08h** : ساعة تدور بمقدار 18.6 نقرة في الثانية.

**09h** : قراءة من لوحة المفاتيح. المفتاح مشفر بكود خاص بهذه اللوحة (scan code)، ويتم تحويله إلى كود آسكي بفضل المقاطعة 16h.

**0Bh** : تسيير المنفذ COM2.

**0Ch** : تسيير المنفذ COM1.

**10h** : تسيير بطاقة الفيديو.

**11h** : قائمة التعديلات (الذاكرة، عدد منافذ الكوم (COM)، المعالج الثانوي Co-Processor، ...)

**12h** : حجم الذاكرة المنخفضة Low (640 كيلو بايت كحد أقصى).

**13h** : تسيير مختلف الأقراص.

**14h** : تسيير واجهة السلسلة (منافذ الكوم، لاحظ المقاطعتين 0Bh و 0Ch).

**15h** : مقبض اللعب، أشرطة و التوب فيو (TopView).

**16h** : تحويل كود المفتاح (الملمس) المقروء من قبل المقاطعة 09h وتحويله إلى كود آسكي.

**17h** : تسيير الطابعة.

**18h** : الذاكرة الميتة القاعدية (basic rom).

**19h** : روتين تحميل الدوس.

**1Ah** : تسيير الساعة الحقيقية (real).

**1Bh** : مراقبة الضغط المشترك للمفتاحين CTRL+C.

**1Ch** : عداد نقرة/نقرة بسرعة الساعة 08h أي 18.6 هرتز، قيمته تحفظ في الموضع 0040h: 0060h

**1Dh** : جدول تهيئة (استهلال) الفيديو.

**1Eh** : جدول بارامترات الأقراص المرنة.

**1Fh** : جدول الرموز البيانية (Graphic).

**20h** : مقاطعة الدوس: نهاية برنامج من نوع com. (استعمال النوع exe. هو السائد حاليا)

**21h** : مقاطعة الدوس: دوال ومهام عامة (القرص الصلب، الساعة، الطباعة، الإخراج والإدخال من و إلى الشاشة أو الملفات، ...).

**22h** : مقاطعة الدوس: عنوان نهاية المعالجة (القرص الصلب، الساعة، ...).

**23h** : مقاطعة الدوس: مراقبة CTRL+PAUSE أو CTRL+BREAK.

**24h** : مقاطعة الدوس: خطأ قاتل في اتجاه المقاطعة.

**25h** : مقاطعة الدوس: قراءة مباشرة من قرص ما.

**26h** : مقاطعة الدوس: كتابة مباشرة على قرص معين.

**27h** : مقاطعة الدوس: برامج مستقرة في الذاكرة.

**28h** : نهاية برنامج بقي مستقرا في الذاكرة.

**2Fh** : مقاطعة للعديد من البرامج الفرعية (تسيير الشبكة، درايفر السيدي...).

## أمثلة عامة:

نواصل درسنا الأخير في لغة التجميع، وهذه المرة مع الأمثلة الخاصة بأغلب وأهم التعليمات المستخدمة في هذه اللغة. لكن قبل ذلك يلزمنا البرنامج أو البرامج المناسبة لتطبيق ذلك، لهذا أنصح المهتم بهذه الدروس أن يقوم بتنزيل المصرّف TASM (أو MASM) وبرنامج TLINK (أو LINK من مايكروسفت).
فيم يخص التطبيقات المنجزة فلقد جربتها بـ TASM و LINK (أي أخلطت بين الإصدارات)، والسبب هو أن منتج بورلاند TLINK يرسل رسائل الخطأ والتي مفادها أن العمل يجري خارج الذاكرة، وهذا ما يستلزم تعديلات أخرى نحن في غنى عنها، لهذا سنقوم بعملنا مستعملين البرنامجين TASM و LINK والذين ستجدهما كتوفرين معا في ملف مضغوط واحد.

عند كتابة البرنامج (الذي فرضناه FirstPrg)، قم بتنفيذ الخطوات الآتية:

```
Tasm FirstPrg // عملية تصريف البرنامج
link FirstPrg // عملية ربط تحكمات البرنامج
FirstPrg // تنفيذ البرنامج
```

## المثال الأول:

لفهم العملية جيدا، فإنه لدينا هذا البرنامج، والذي يقوم بطبع جملة Hello World على الشاشة:

```
;-----------------------------------------------------------------
;  The First Program                                              ;
;  بعد الفاصلة المنقوطة يمكننا إدراج تعليقات لتوضيح الفكرة
;  هذه التعليقات سيهملها المصرف لاحقا
;-----------------------------------------------------------------
; قطاع البيانات

data16  segment public       ; التصريح بقطاع البيانات

Hello   db 'Hello World!',13,10,'$' ; متغير

data16  ends

; ---------------------------------------------------------------
; قطاع الكود

code16  segment public   ; التصريح بقطاع الكود
        assume  cs:code16, ds:data16, ss:stack16

start16:
; تحضير قطاع البيانات
        mov ax, data16       ; ax = data16
        mov ds, ax           ; ds = ax
```

```
; طباعة نص على الشاشة
        mov dx, offset Hello ; hello فرع الذاكرة الخاص بالجملة
        mov ah, 09h          ; 09h تسمح بطباعة النص على الشاشة
        int 21h              ; مقاطعة الدوس

; إنهاء البرنامج
        mov al, 00h          ; ولا مشكلة
        mov ah, 4ch          ; 4ch تسمح بمغادرة البرنامج
        int 21h              ; مقاطعة الدوس

code16  ends                 ; نهاية النص

; ------------------------------------------------------------
; قطاع المكدس

stack16 segment stack
        db  200h dup (?)
stack16 ends

        end start16          ; start16 نهاية البرنامج
; ------------------------------------------------------------
```

*تحليل البرنامج:*

تشكل لنا القطاعات (segments) هيكل البرنامج. عامة يوجد قطاع الكود، قطاع البيانات، وقطاع المكدس (stack). قطاع الكود يستعمل للتعليمات (instructions)، قطاع البيانات للمتغيرات، وقطاع المكدس لتعليمات المكدس نفسه كتعليمتي push و pop.

التصريح العام للقطاع:

```
"Name" segment [public|stack]
        ......... data instructions
"Name" ends
```

التعليمة assume (تكلّف) تخبر المصرف TASM بالاستعمال الحالي لسجل القطاع (segment register).

Code16 هو قطاع خاص بالكود، إذن قطاع الكود (CS) متكلّف بـ code16.

Data16 هو قطاع خاص بالبيانات، إذن قطاع البيانات (DS) متكلّف بـ Data16.

Stack16 هو قطاع خاص بالمكدس، إذن قطاع المكدس (SS) متكلّف بـ Stack16.

عند الإقلاع يقوم الدوس بتحضير (تهيئة) CS و SS أما DS فلا يتم تحضيره. إذن يجب عليك تحضير DS أو ستتعرض للمفاجئات.

```
Adapt the data register
        Mov ax, data16  ; نستعمل ax لأننا لا نستطيع الكتابة
        Mov ds, ax      ; حاليا ds محضّر
```

يعتبر قطاع المكدس (SS) وقطاع البيانات (DS) متخصصين بالمعطيات (البيانات). لكن قطاع المكدس هو منطقة ذاكرية مؤقتة للتعليمات pop و push، لهذا فإنه لا يوجد إلا فرع ذاكري واحد.
قطاع البيانات يمكنه احتواء متغيرات وثوابت. في هذا المثال، 'hello' تعتبر ثابت على شكل حزمة رموز (string). في المثال قمنا بوضع فرع الذاكرة (offset) الخاص بهذه الجملة في dx لأن مقاطعة الدوس تحتاج لفرع الذاكرة حتى يمكنها طباعة النص على الشاشة. 'hello' هي المتغير الأول والذي يملك فرعا ذاكريا (أقرب مرادف للـ offset) معدوما.

```
Declaration of data:
"name"  db  ?              ; متغير من 8 بت: بايت
"name"  dw  ?              ; متغير من 16 بت: كلمة
"name"  dd  ?              ; متغير من 32 بت: كلمة_مضاعفة
"name"  db  ?,?            ; 2 بايت
"name"  dw  ?,?,?,?        ; 4 كلمات
"name"  db  'Hello'        ; سلسلة من البايتات
"name"  dw  32 dup (?)     ; 32 كلمة
```

جرب البرنامج ولاحظ النتيجة.
حاول تعديل بعض القيم أو تغييرها، لكن إحذر من الأعمال العشوائية.
إذا لم تفهم كل الذي قلناه، فلا تقلق، لأنه مع الوقت يكون بإمكانك تحقيق مرادك وبلغة التجميع.

هذه الدروس لم تكتمل، فللحديث بقية...