التعريف بدوال الـAPI

كلمة API هي إختصار Application Programming Interface أي واجهة برمجة التطبيقات ومعناها أن أحد التطبيقات يحتوي علي مجموعة إجراءات يمكن تصديرها للتطبيقات الأخري بحيث يمكن لهذه التطبيقات أن تستدعيها ويشير مصطلح واجهة برمجة التطبيقات إلي مكتبات الربط الديناميكي التي توجد مع كل نسخه ويندوز والتي تحتوي علي العديد من الإجراءات التي يستخدمها المبرمجون عند كتابة برامجهم. فمثلاً برنامج "الفيجوال بيسيك" توجد فيه إمكانية إستدعاء الإجراءات الخارجية الموجوده في مكتبات الربط الديناميكي الموجوده في الويندوز ودائماً ما تنتهي مكتبات الربط الديناميكي بالإمتداد " dll. " وسوف أذكر أهم هذه المكتبات :-

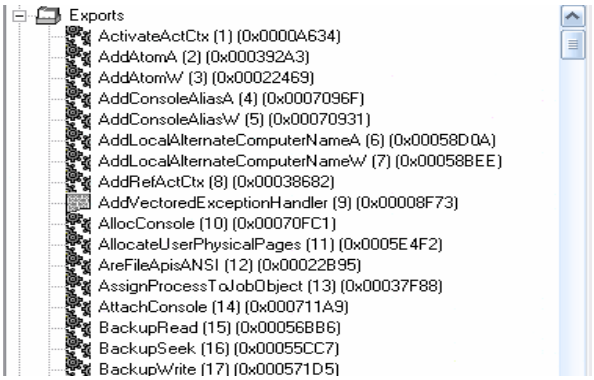
Kernel32 : الإجراءات المتعلقة بتشغيل البرامج، وتنظيم الذاكرة، والإنتقال بين البرامج، والتعامل مع موارد النظام.

User32 : الإجراءات التي تتعامل مع النوافذ مثل: صنع النوافذ وإظهارها وإخفائها والإجراءات التي تتعامل مع الرسائل التي ترسلها هذه المكتبه للتطبيقات الأخري والإجراءات الخاصه بالقوائم.

Gdi32 : الإجراءات الخاصه بالرسم والصور والعرض علي الشاشة والطابعات وإجراءات التعامل مع الخطوط

advapi32 : الإجراءات الخاصة بالتعامل مع سجلات النظام مثل كتابه مفتاح أو تحرير مفتاح أو حذف مفتاح.

ولو إستخدمنا برنامج PEBrowse ثم إختارنا ملف Kernel32.dll ثم نذهب إلي قسم الـExport سوف تري هذه الدوال :



كما تري فهذه هي الدوال التي يتم تصديرها للتطبيقات الأخري أو Visual C++ أو لغة الدلفي أما الفيجوال بيسيك فيستخدم ملفات معروفة مثل msvbvm60.dll وهذا الملف للإصدار

السادس من الفيجوال بيسيك أما الإصدار الخامس فيستخدم ملف msvbvm50.dll ولو مثلا وقفنا علي دالة lstrcmp سوف تري في الجهة المقابلة هذه البيانات :

```
;*****************************************************************************
; *** lstrcmp (935) ***
SYM:lstrcmp
0x7C81EE79: 8BFF            MOV     EDI,EDI
0x7C81EE7B: 55              PUSH    EBP
0x7C81EE7C: 8BEC            MOV     EBP,ESP
0x7C81EE7E: 53              PUSH    EBX
0x7C81EE7F: 8B5D08          MOV     EBX,DWORD PTR [EBP+0x8]
0x7C81EE82: 56              PUSH    ESI
0x7C81EE83: 57              PUSH    EDI
0x7C81EE84: 8B7D0C          MOV     EDI,DWORD PTR [EBP+0xC]
0x7C81EE87: 6AFF            PUSH    0xFF
0x7C81EE89: 57              PUSH    EDI
0x7C81EE8A: 6AFF            PUSH    0xFF
0x7C81EE8C: 53              PUSH    EBX
0x7C81EE8D: BE00000040      MOV     ESI,STATUS_OBJECT_NAME_EXISTS; ERR:(0x40000000)
0x7C81EE92: 56              PUSH    ESI
0x7C81EE93: E86DB5FEFF      CALL    GetThreadLocale             ; (0x7C80A405)
0x7C81EE98: 50              PUSH    EAX
0x7C81EE99: E8F5E3FEFF      CALL    CompareStringA              ; (0x7C80D293)
0x7C81EE9E: 85C0            TEST    EAX,EAX
0x7C81EEA0: 0F8489170200    JZ      0x7C84062F                  ; (*+0x2178F)
0x7C81EEA6: 83C0FE          ADD     EAX,0xFE                    ; <==0x7C840643(*+0x2
0x7C81EEA9: 5F              POP     EDI                         ; <==0x7C840675(*+0x2
0x7C81EEAA: 5E              POP     ESI
0x7C81EEAB: 5B              POP     EBX
0x7C81EEAC: 5D              POP     EBP
0x7C81EEAD: C20800          RET     0x8
;*****************************************************************************
0x7C84062F: 6AFF            PUSH    0xFF                        ; <==0x7C81EEA0(*-0x2
0x7C840631: 57              PUSH    EDI
0x7C840632: 6AFF            PUSH    0xFF
0x7C840634: 53              PUSH    EBX
0x7C840635: 56              PUSH    ESI
0x7C840636: E82AC2FCFF      CALL    GetSystemDefaultLCID        ; (0x7C80C865)
0x7C84063B: 50              PUSH    EAX
0x7C84063C: E852CCFCFF      CALL    CompareStringA              ; (0x7C80D293)
0x7C840641: 85C0            TEST    EAX,EAX
0x7C840643: 0F855DE8FDFF    JNZ     0x7C81EEA6                  ; (*-0x2179D)
0x7C840649: 85DB            TEST    EBX,EBX
0x7C84064B: 743F            JZ      0x7C84068C                  ; (*+0x41)
0x7C84064D: 85FF            TEST    EDI,EDI
0x7C84064F: 7433            JZ      0x7C840684                  ; (*+0x35)
0x7C840651: 8BF7            MOV     ESI,EDI
0x7C840653: 8BC3            MOV     EAX,EBX
0x7C840655: 8A10            MOV     DL,DWORD PTR [EAX]          ; <==0x7C840671(*+0x1
0x7C840657: 8ACA            MOV     CL,DL
0x7C840659: 3A16            CMP     DL,DWORD PTR [ESI]
0x7C84065B: 751D            JNE     0x7C84067A                  ; (*+0x1F)
0x7C84065D: 84C9            TEST    CL,CL
0x7C84065F: 7412            JZ      0x7C840673                  ; (*+0x14)
0x7C840661: 8A5001          MOV     DL,BYTE PTR [EAX+0x1]
0x7C840664: 8ACA            MOV     CL,DL
0x7C840666: 3A5601          CMP     DL,BYTE PTR [ESI+0x1]
0x7C840669: 750F            JNE     0x7C84067A                  ; (*+0x11)
0x7C84066B: 40              INC     EAX
0x7C84066C: 40              INC     EAX
```

فكما تري فهذه السطور البرمجية الخاصة بهذه الدالة وكذلك الحال بالنسبة لكل الدوال. وإذا قمت بإختيار أي من الملفات التي شرحناها بالأعلي سوف تري الدوال المستخدمة لها ويوجد أيضاً مسمي أخر لهذه الدوال وهو IAT وهي إختصار **I**mport **A**ddresses **T**abels أي قائمة العناوين المستوردة والمقصود بهذا المسمي أن هذه العناوين تم إستيرادها من الملفات Advapi32.dll Gdi32.dll , User32.dll , Kernel32.dll لذلك يعتبر مسمي API و IAT مجازاً مسمي واحد.

شرح موجز عن لغة الأسمبلي

الأسمبلي هي لغة برمجة تتكون من سلسلة من التعليمات المتتابعة كل تعليمة فيها تحول إلي تعليمة مقابلة بلغة الآلة.

وتعتبر لغة الأسمبلي حالياً من اللغات الغير شعبية لدي المبرمجين فكثير من المبرمجين يستخدمون اللغات عالية المستوي مثل لغة السي والفيجوال بيسيك ولكن لغة الأسمبلي من أقوي اللغات في التعامل مع عتاد الكمبيوتر ويمكن إنشاء برنامج صغير بالأسمبلي فميزة هذه اللغة أنها خالية من الأخطاء وسريعة وتمتاز أيضاً بالمرونه ولا يمكن كتابة برنامج كبير بها لأن الكود سيفقد مرونته وتكثر أخطائه مما يفقد السيطرة علي برنامج مكتوب من آلاف الأسطر ولكن يمكن كتابة برنامج صغير ودمجة مع لغة الـC مثلاً أو كتابة ملف DLL يتم تصديرة إلي لغات أخري.

ويوجد إختلاف بين برامج الدوس وبرامج الويندوز فبرامج الدوس تستخدم المقاطعات كوظائف وقد تسأل ما هي **المقاطعة** ؟ فمثلاً إذا كنت جالس أمام الكمبيوتر تعمل وجاء والدك يناديك فإنك سوف تقطع وقتك وتذهب من أمام الكمبيوتر إلي والدك لكي تنفذ ما يريده ثم تحضر لإستكمال عملك وهذه هي المقاطعة فالمعالج يكون مشغول في أمر ما ونطلب منه نحن أن ينفذ لنا طلباً ما **فيقطع** المعالج الأمر الذي كان مشغول فيه ثم يحضر لينفذ لنا طلبنا ثم يذهب لإستكمال الامر الذي كان مشغول فيه ، أما برامج الويندوز تستخدم دوال الـAPI مثل MessageBoxA و CreateWindowEx .

في نظام العد المتداول بين الناس نستخدم أصابعنا العشره لكي نعد عليها فلو بدأنا بالرقم 0 سوف ننتهي بالرقم 9 ولذلك يطلق علي هذا النظام " النظام العشري " نسبه إلي إستخدام أصابعنا العشره ويسمي أيضاً هذا النظام بالـDecimal ويوجد نظام أخر يطلق عليه Hexadecimal وينتهي هذا النظام عند الرقم 16 والهدف من ذلك تمثيل النظام الثنائي بأقل عدد من الأرقام فنحن نعلم أن النظام الثنائي يتكون من رقمين فقط هما 0 , 1 فكتابه رقم معين يأخذ عدد كبير من الأرقام الثنائية فمثلاً الأرقام هذه : 10001001 = 89 في النظام الست العشري أو من الممكن أن نطلق عليه النظام السداسي عشر ويبدأ هذا النظام بالرقم 0 وينتهي بالرقم 15 ولكن نظراً لأن الأرقام من 10 إلي 15 تتكون من وحدتين أي عددين فقد تم التعويض عنها بحروف كما في الجدول التالي :

النظام العشري (Decimal)	النظام الست العشري (Hexadecimal)
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

يشار للرقم في نظام الست العشري بالحرف h.

البايت (Byte) : البايت يساوي 8 بت وهذه هي أصغر وحدة قياس فمثلاً حرف A يمثل بايت واحد والرقم المقابل له في نظام الـDecimal هو 65 وفي نظام الـHexadecimal هو 41 وبذلك يطلق علي هذا النظام نظام الـ8 بت لأنه يتكون من 8 بت أي بايت واحد.

الكلمة (word) : بما أن البايت يساوي 8 بت فإن الكلمة تساوي 16 بت لأن الكلمه علي الأقل 2 بايت ويطلق علي هذا النظام نظام الـ16 بت لأنه يتكون من 16 بت أي 2 بايت أي كلمه واحده.

الكلمة المضاعفة (DWord أو Double Word) : تشغل الكلمه من هذا النوع 4 بايت لذلك يطلق عليها الكلمة المضاعفة والـ4 بايت يمثلون 32 بت لذلك يطلق علي هذا النظام نظام الـ32 بت.

أساسيات الأسمبلي

المُسجّلات (Registers) :-

مُسجّلات الأغراض العامة (General Purpose Registers) :

وهي عبارة عن كل من المُسجّلات AX و BX و CX و DX طول كل منها 16 بت أي كلمة أي 2بايت البايت اليساري فيهما يعرف بالعلوي (High) أما البايت الأيمن فيهما فيعرف بالمنخفض(Low) فمثلاً المُسجّل AX يتألف من مُسجّلين العلوي وهو AH والمنخفض وهو AL تم توسيع المُسجّلات في معالجات الـ32 بت مع بقاء المُسجّلات نفسها ولكن كل منها أصبح جزء من مُسجّل موسع بطول 32بت وهي EAX,EBX,ECX,EDX ، أي أن المُسجّل EAX هو بطول 32بت وكجزء منه هناك المُسجّل AX بطول 16بت والذي يتألف هو الآخر من مُسجّلين هما AL و AH بطول 8 بت لكل منهما ولو لاحظت فالمسجلات في نظام الـ32 بت يتم إضافة لها حرف "E" وهذا الحرف هو إختصار لـExtended أي توسيع المسجلات لتصبح في نظام الـ32 بت وإليك المثال التالي :-

EAX = EA 78 23 BB (32 – bit)

AX = EA 78 23 BB (16 – bit)

AH = EA 78 23 BB (8 – bit)

AL = EA 78 23 BB (8 – bit)

فكما تري فإن مسجل الـEAX وهو في نظام الـ32 بت يساوي القيمة السابقة والمسجل AX يساوي الـ2 بايت وهم يمثلون كلمة وهو في نظام الـ16 بت أما المسجل AH فيساوي البايت اليساري والمسجل AL فيساوي البايت الأيمن

15	7	0	
AH	AL		AX
BH	BL		BX
CH	CL		CX
DH	DL		DX

المُسجّل AX (Accumulator Register) :

هذا المُسجّل كان من أهم المُسجّلات في معالجات الـ8 بت القديمة جداً حيث كانت تجري من خلاله كل العمليات الرياضية والمنطقية ولذلك كان يسمى بمُسجّل المركم لتراكم النواتج فيه.

المُسجّل BX (Base Register) :

هو المُسجّل الوحيد من بين مُسجّلات الأغراض العامة الذي يمكن إستخدامه كدليل(INDEX) ، يمكن إستخدام هذا المُسجّل للعمليات الرياضية والمنطقية.

المُسجّل CX (Counter Register) :

يستخدم عادة كعداد ويستخدم هذا المُسجّل بشكل خاص مع تعليمة التكرار LOOP حيث يعمل كعداد لها وبالطبع يمكن استخدامه في العمليات الرياضية والمنطقية.

المُسجّل DX (Data Register) :

يفضل إستخدام هذا المُسجّل لتخزين المعطيات في عمليات الدخل والخرج والمقاطعات وبالطبع فأنه يمكن إستخدامه كباقي المُسجّلات في العمليات الرياضية والمنطقية.

<u>مسجلات الأقسام (Segment Registers) :</u>

Code Segment -CS: يحمل هذا المُسجّل عنوان بداية القسم الخاص بالشفرة في البرنامج.

Data Segment -DS : يحمل هذا المُسجّل عنوان بداية قسم البيانات في البرنامج.

Stack Segment -SS : يحمل هذا المُسجّل عنوان بداية قسم المكدسة في البرنامج.

Extra Segment -ES : يحمل هذا المُسجّل عنوان بداية قسم إضافي يمكن أن يستعمل هذا القسم الإضافي كقسم بيانات آخر.

المُسجّلات الدليلية (Index Registers) :

Source Index -SI : يستخدم هذا المُسجّل في التأشير على النص المصدر وذلك لأجراء العمليات التي تتعامل مع النصوص.

Destination Index -DI : يستخدم هذا المُسجّل في التأشير على النص الهدف وذلك لأجراء العمليات التي تتعامل مع النصوص.

Instruction Pointer - IP : يحتوي المُسجّل IP على إزاحة التعليمية التالية التي ستنفذ ، أي أن المُسجّل عبارة عن مؤشر إلي التعليمية التالية الموجودة في مقطع الشفرة CS-Code Segment المنفذ حالياً.

مسجلات المكدسة (Stack Registers) :

Base Pointer -BP : يعمل هذا المُسجّل على تسهيل الوصول إلي الوسيطات(البارمترات) والتي تحتوي على عناوين ومعطيات والتي دفعت PUSH بشكل مؤقت إلي المكدسه عند استدعاء روتينات فرعية من البرامج.

Stack Pointer -SP : يحتوي المُسجّل SP على كلمة الذاكرة الحالية التي ستعالج في المكدس. هذا المُسجّل يعدل آلياً بواسطة المعالج مع عملية دفع PUSH أو سحب POP في المكدس ليشير دوماً إلي قمة المكدس.

الرايات أو الأعلام (Flags) : هي بتات موجوده داخل المعالج وهذه قائمة الرايات :

CF : قيمة هذا العلم ١ إذا كان نتيجة أخر عملية كبيرة جداً علي الهدف (في الأعداد التي بدون إشارة فقط) وإليك هذا المثال

```
Mov ch, 250
Add ch, 100
```

بما أن المُسجّل ch 8 بت أي بايت واحد فإن أقصي قيمة يتحملها 256 وبما أن القيمة في المُسجّل ch هي 250 ثم إضفنا لها 100 فإن النتيجة أكبر من الهدف لذلك العلم CF سوف يساوي ١.

PF : ببساطة يساوي هذا العلم ١ إذا كان عدد الوحايد في ناتج آخر عملية رياضية أو منطقية زوجياً وبصفر إذا كان فردياً ، فمثلاً لو كان جواب آخر عملية = ٠٠١٠٠٠١٠ فإن العلم سوف يساوي ١ لأن عدد البتات التي تحوي وحايد تساوي ٢ وهو عدد زوجي أما إذا كان الجواب مثلاً يساوي ١١١٠٠٠٠٠ فإن العلم يصفر لأن عدد البتات التي تحوي وحايد يساوي ٣ وهو عدد فردي.

AF : يضبط العلم ١ إذا تسببت آخر عملية رياضية أو منطقية حمل من البت الثالثة إلي البت الرابعة أو استلاف من البت الرابعة إلي البت الثالثة.

ZF : يساوي هذا العلم ١ إذا كانت نتيجة آخر عملية رياضية أو منطقية تساوي صفر.

SF : يضبط هذا العلم ١ إذا كان ناتج آخر عملية رياضية أو منطقية سالب ويصفر إذا كان موجب.

OF : هو نفس علم الحمل لكن مع العمليات ذي الإشارة أي أنه يضبط إذا كان ناتج آخر عملية أكبر أو أصغر من حدود الهدف.

أوامر الاسمبلي أو التعليمات

لغة الأسمبلي تستخدم مجموعة من التعليمات لكي تنفيذ أمر ما والمعالج يعمل علي تحويل قيم التعليمات إلي hexcodes لكي يقرأها المعالج ويقوم بتنفيذها أما اللغات عالية المستوي مثل لغة C يتم تحويلها إلي لغة الأسمبلي ثم إلي لغة الـBinary Code والتعليمة تنقسم إلي قسمين : رمز العملية opcode = operation code والمتحولات operands ، ورمز العملية هو جزء من التعليمة كالجمع والطرح والضرب مثل ADD و SUB و MUL أما المتحولات فهي عبارة عن المعطيات التي سوف تُعالج بواسطة المعالج وهي الجزء الأخر من التعليمة فمثلا :

ADD CX, DX

رمز العملية هو الجزء ADD أما المتحولات هي نتيجة إضافة قيمة المسجل DX إلي قيمة المسجل CX ولذلك يعتبر DX متحول المصدر وCX متحول الهدف أي أن المتحولات هي الجزء الأخر من التعليمة. وإليك أهم التعليمات :

التعليمة ADD

ADD EAX, EDX

هذه التعليمة تضيف قيمة المسجل EDX إلي قيمة المسجل EAX مع ثبات قيمة EDX

ADD EAX, EDX = (EDX + EAX), EDX

التعليمة MOV

MOV Destination, Source

هذه التعليمة تستخدم لنقل قيمة من مكان إلي أخر (أو بالأصح نسخ هذه القيمة) هذا المكان ممكن أن يكون مسجل أو موقع معين في الـMemory فمثلاً :

MOV EDX, ECX

هذه التعليمة تستخدم لنقل محتويات المسجل ECX إلي المسجل EDX ولابد من أن يكون نفس الحجم أي لا ينفع أن تكون التعليمة كالتالي :

MOV AL, ECX

فالتعليمة السابقة تحاول نسخ قيمة DWORD(32–bit) إلي BYTE(8–bit) وهذا غير ممكن فلابد من أن يكون نفس الحجم.من الممكن الحصول علي قيمة محددة في الـmemory ثم وضعها في مسجل معين وإليك الجدول التالي كمثال :

offset	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	40	41	42
data	0D	0A	50	32	44	57	25	7A	5E	72	EF	7D	FF	AD	C7

فمثلاً لو كتبنا :

MOV EAX, DWORD PTR [0000003Ah]

التعليمة السابقة تعني وضع قيمة 3Ah وهو موقع معين في الـmemory داخل المسجل EAX وبما إننا نستخدم DWORD(32–bit) لذلك سوف تصبح قيمة المسجل EAX 725E7A25 ربما لاحظت أننا عكسنا هذه القيم فكما تري في الجدول السابق أن القيم هي 257A5E72 ولكن نحن عكسناها لتصبح 725E7A25 ويرجع السبب إلي ان الـMemory تستخدم هذه الهيئة في التعامل مع hexcode فالبايت الأيمن هو الذي يستخدم أولاً ثم تليه بعد ذلك بقية البايتات ولكن من الناحية اليمني.

التعليمة CALL

CALL (Procedure)

تعمل هذه التعليمة علي نداء بعض الإجراءات للتأكد من شئ معين.

التعليمة RET

RET (Return from procedure)

العوده من الإجراءات الذي كان يناديها أمر النداء (CALL) ثم الإنتقال إلي التعليمة التي تلي أمر النداء.

التعليمة CMP

CMP EAX, EDX

ومن الممكن أن تكون EDX بقيمة المُسجّل EAXهذه التعليمة تعمل علي مقارنة قيمة المُسجّل المقارنة بين نصوص وأحياناً تكون هذه التعليمة مقارنة بين السيريال الصحيح و السيريال الخاطئ.

التعليمة SUB

SUB Destination, Source

(Destination = Destination – Source)

كما تري فهذه التعليمة تعمل علي طرح الـDestination من الـSource وتُخزن قيمة الطرح في الـDestination مع بقاء قيمة الـSource كما هي.

التعليمة MUL

MUL Destination, Source

(Destination = Destination × Source)

كما تري فهذه التعليمة هي عملية ضرب الـDestination في الـSource مع بقاء قيمة الـSource كما هي.

التعليمة DIV

DIV Source **(** EAX = EAX / Source, EAX = Remainder (الباقي) **)**

وهذا بيان نتيجة القسمة :-

Source Size	القسمة	نتيجة القسمة تُخزن في البايت الأدني	باقي نتيجة القسمة تُخزن في البايت الأعلي
Byte(8-bit)	ax / Source AH AL	البايت الأيمن AL	البايت اليساري AH
Word(16-bit)	dx:ax / Source	ax	dx
Dword(32-bit)	edx:eax / Source	eax	Edx

كما تري فإن حاصل القسمة يوضع في البايت الأدني ألا وهو البايت الأيمن ونتيجة القسمة في البايت الأعلي ألا وهو البايت اليساري.

التعليمة INCREMENT و DECREMENT

التعليمة INCREMENT تعمل علي زيادة المسجل أو موقع معين في الـmemory بمقدار "١" . والتعليمة DECREMENT تعمل علي نقصان المسجل أو موقع معين في الـmemory بمقدار "١" .

INC reg -----→ reg = reg + 1

DEC reg ----→ reg = reg -1

INC DWORD PTR [103405] --→ سوف يتم زيادة القيمة بمقدار "١"

DEC DWORD PTR [103405] --→ سوف يتم نقصان القيمة بمقدار "١"

التعليمة NOP

هذه التعليمة لا تساوي شيئاً والغرض منها شغل مساحة فقط بدون فائدة.

التعليمة XCHG

EAX = 237h

ECX = 978h

XCHG EAX, ECX

EAX = 978h

ECX = 237H

هذه التعليمة تعمل علي تبادل قيم المسجلات فيما بينها أو بين قيمة مسجل وموقع معين في الـmemory.

بعض التعليمات مثل AND , OR , XOR , NOT

instruction	AND				OR				XOR				NOT	
Source Bit	0	0	1	1	0	0	1	1	0	0	1	1	0	1
Destination Bit	0	1	0	1	0	1	0	1	0	1	0	1	X	X
Output Bit	0	0	0	1	0	1	1	1	0	1	1	0	1	0

التعليمة AND تجعل قيمة البت الناتج "١" إذا كان قيمة الـSource والـDestination تساوي "١" .التعليمة OR تجعل قيمة البت الناتج "١" إذا كانت قيمة الـSource أو الـDestination تساوي "١" .التعليمة XOR تجعل البت الناتج يساوي "١" إذا كانت قيمة الـSource والـDestination مختلفين أما إذا كان متساويين فإن قيمة البت الناتج تساوي صفر.التعليمة NOT تعكس أو تقلب قيمة الـSource فإذا كان قيمة الـSource تساوي "٠" فسوف يصبح قيمة البت الناتج تساوي "١".

تعليمات القفز

تعليمة القفز غير المشروط :-

JMP Target

تقوم هذه التعليمة بالقفز إلي مكان ما. ولكن عملية القفز غير مشروطة بأمر ما بمعني أن عملية القفز سوف تتم في كل الاحوال وغير متبوعة بشرطاً ما.

تعليمات القفز المشروطة :-

حالة الراية أو العلم	Jump IF	الأمر
CF or ZF = 0	Above / not below or equal	JA/JNBE
CF = 0	Above or equal / not below	JAE/JNB
CF = 1	Below / not above or equal / carry	JB/JNAE/JC
CF or ZF = 1	Below or equal / not above	JBE/JNA
ZF = 1	Equal / zero	JE/JZ
CF = 0	No carry	JNC
ZF = 0	Not equal / not zero	JNE/JNZ
OF = 0	Not overflow	JNO
PF = 0	No parity / parity odd	JNP/JPO
SF = 0	No sign	JNS
OF = 1	Overflow	JO
SF = 1	sign	JS
ZF = 0 and SF = OF	Greater / not less not equal	JG/JNLE
SF = OF	Greater or equal / not less	JGE/JNL
SF <> OF	Less / not greater or equal	JL/JNGE
ZF = 1 or SF <>OF	Less or equal / not greater	JLE/JNG

وتوجد تعليمات أخري ولكنها ليست مهمه لنا وبذلك نكون قد إنتهينا من شرح التعليمات المهمه بالنسبة لنا.