

الفريق العربي للهندسة العكسية

دورة ATRE لتعليم الأسمبلي للمبتدئين من الصفر
الدرس الخامس- تعليمات القفز - المتغيرات
allko

درس اليوم يتناول قسمين مهمين...تعليمات القفز إضافة الى المتغيرات...

I- jumping instructions

كما تعلم فإن المعالج يقوم بتنفيذ الأوامر بالترتيب...لكن إن أردنا نقل التنفيذ الى جزء آخر في البرنامج فيمكننا عمل ذلك بواسطة أوامر القفز...تقسم هذه الأوامر الى قفز مشروط conditional jump وقفز غير مشروط unconditional jump .
الجدول التالي يوضح أهم أوامر القفز المشروط وهناك الكثير غيرها سنذكرها لاحقاً...

Instruction	Description	Affected flags
JZ	اقفز اذا كانت النتيجة صفر	ZF=1 → jump
JNZ	اقفز اذا كانت النتيجة لا تساوي صفر.	ZF=0 → jump
JE	اقفز اذا كانت النتيجة صفر	ZF=1 → jump
JNE	اقفز اذا كانت النتيجة لا تساوي صفر.	ZF=0 → jump
JC	اقفز اذا ال MSB أخرج 1 للخارج (carry out) او احتاج لواحد من الخارج (borrow)	CF=1 → jump
JNC	اقفز اذا ال MSB لم تخرج 1 للخارج (carry out) او لم يحتاج لواحد من الخارج (borrow)	CF=0 → jump

انتبه الى أن jz تعمل مثل عمل je ، و jnz تعمل مثل عمل jne ...
والآن دعنا نوضح مفهوم ال carry وال borrow قليلاً...هذه الجزئية من الدرس مقتبسة - بتصرف - من أحد الدروس للأخ afeef من الفريق العربي للبرمجة ، فله الشكر.
أنظر الجدول التالي لتعرف الحالات الأربعة الممكنة لمفهوم ال carry :

1 st number	2 nd number	sum(S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

كما ترى ففي حالات الجمع الثلاثة الأولى (الصفوف الثلاثة الأولى) فإن ال carry يساوي 0 ، لكن الحالة الرابعة عند جمع 1 مع 1 فالناتج هو اثنان أي 10 إذن خانة المجموع (sum) يساوي 0 وال carry يساوي 1 (بالعامية نقول : باليد واحد ، وهذه بالأسمبلي نقول عنها : carry)

والآن دعنا نرى ما الذي يحدث عند جمع العددين 87H+93H

87H=10000111

83H=10010011

					C	C	C		Carry
	1	0	0	0	0	1	1	1	1 st number
	1	0	0	1	0	0	1	1	2 nd number
1	0	0	0	1	1	0	1	0	result
ignored	MSB							LSB	

النتيجة هي 1A h = 00011010 ... والآن ما هي قيمة الراية ZF (Zero Flag) ؟ النتيجة ليست بصفر (واضح !!!) اذن ZF=0 .

والآن ما هي قيمة الراية SF (Sign Flag) ؟ بالتأكيد 0 ، لأن النتيجة موجبة...كيف عرفت؟ راجع الدرس الأول...أنظر الى الـ MSB كما ترى إنه 0 مما يدل على أن العدد موجب.

ماذا عن قيمة الراية CF (Carry Flag) ؟ أنظر الى الخانة الأخيرة في كلا العددين (الـ MSB) ، كلاهما قيمتهما 1 ، وقد قلت للتو أنه في حالة جمع 1 و 1 فالنتيجة 10 ، وكما ترى نضع الـ 0 أما الـ 1 فينقل للخانة التالية (هذا الـ 1 يظهر باللون الأحمر و أسفل منه مكتوب ignored يعني تم تجاهله) لكننا هنا تجاهلناه ولم نعتبره جزءا من النتيجة...لماذا؟ هذا يعتمد على التعليمة...فتعليمة كهذه

MOV BL , 83H

ADD BL , 87H

تظهر لنا أن النتيجة ستخزن في BL ومعلوم لنا أن حجمه يتسع لـ 8 بتات فقط ، اذن النتيجة ستقتصر على 8 بتات والبت الأخير يذهب الى Carry Flag .

وهذا ما يسمى بمفهوم الـ overflow.

أما تعليمة كهذه

MOV BX , 83H

ADD BX , 87H

فلن يحدث فيها overflow أي أن البت الأخير (1) الذي تجاهلناه في المرة السابقة ، لن نتجاهله هنا بل سنعتبره جزءا من الحل (لأن BX يتسع لـ 16 بت أي لا مشكلة)...أي أن الناتج هو 10A h = 100011010 . والآن لتتعرف على مفهوم الـ borrow .

1 st number	2 nd number	sub(S)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

لن أقوم بالمزيد من التوضيح كي لا يخرج الدرس عن غايته الرئيسية...

مثال / دعنا نرى هذا الكود :

CMP BL,2

JE allko

MOV DX,44

JMP at4re

allko:

MOV BL ,7

at4re:

NOP

في البداية يتم مقارنة BL مع القيمة 2 ، ثم هناك قفزة مشروطة JE ، فإذا تساوا ، سينتقل التنفيذ الى allko ، و allko هو عبارة عن label . كما ترى يمكن وضع label في أي مكان من البرنامج وذلك ببساطة بكتابة الاسم الذي تريد متبوعا بنقطتان رأسيتان.

ماذا ان لم يتساوا؟ لن تنفذ القفزة وبالتالي سيتابع المعالج تنفيذ الأوامر بشكل طبيعي... الأمر التالي الذي سينفذ في حالة عدم المساواة هو mov dx,44 . يليه قفزة غير مشروطة الى ال label المسمى at4re .

إذن في حالة المساواة سنصل الى تعليمة mov bl,7 ويتم تنفيذها. ثم ننتقل الى التعليمة NOP لتنفيذها.

أما في حالة عدم المساواة سنصل الى تعليمة nop ويتم تنفيذها.

تعليمة nop هي اختصار لـ no operation أي أنها لا تقوم بعمل شيء.

مثال / دعنا نرى هذا الكود ..وظيفته هي نقل القيمة 25H الى تسع خلايا ذاكرة (من 401200 الى 401208) ...هناك عدة طرق لعمل هذه العملية منها الطريقة التالية :

```
MOV EAX,25H
MOV ECX,8
BEGINNING:
MOV BYTE PTR DS:[401200+ECX],EAX
DEC ECX
JNZ BEGINNING
RET
```

في البداية نضع القيمة 25H في أحد المسجلات وليكن EAX ثم نضع القيمة 8 في المسجل ECX (ليس شرطاً لكن جرت العادة أن نستخدم ECX كعداد) ثم هناك LABEL باسم BEGINNING ... يليه أمر النقل MOV ، لاحظ أن التوجيه (Directive) المستخدم هو BYTE PTR لأن القيمة التي نود نقلها (25 h) هي بحجم بايت . ثم نقوم بإنقاص قيمة العداد (ECX) بمقدار 1 وبالتالي فإن القيمة 25h سيتم نقلها أول مرة الى 401208 + 8 = 401200 ثم هناك قفزة مشروطة فإذا وصلت قيمة ECX الى الصفر فإن ZF=1 (بالتالي نكون قد نقلنا القيمة الى الأماكن التسعة) ولن يتحقق شرط القفز وسنكمل الى الأمر الذي بعده.. لكن بالطبع بعد أول عملية نقل وتنفيذ DEC ECX لأول مرة ، فإن ECX=7 اذا الشرط تحقق وبالتالي سنقفز الى BEGINNING

ونقوم بعملية النقل مرة أخرى وهكذا...الى ان نقوم بعملية النقل في المرة الأخيرة عندها 401200+0=401200 ونكون قد انتهينا إذا ← ECX=0 إذا ← ZF=1 إذا ← الشرط لم يتحقق إذا ← ننفذ الأمر التالي وهو RET وهذا يعني RETURN أي إعادة السيطرة (control) الى نظام التشغيل.

بقي أن نشير الى جزء آخر من تعليمات القفز وتستخدم بكثرة...انتبه الى أنه في هذه التعليمات فاننا نتعامل مع أعداد **ذات إشارة** إما سالبة و إما موجبة . هذا يعني أن هذه الأوامر تعتبر آخر بت (أي MSB) هو بت الإشارة فإذا كان مساوياً لـ 0 فالعدد موجب وإذا كان مساوياً لـ 1 فالعدد سالب.

CMP OPERAND 1 , OPERAND 2

instruction	description
JG	JMP IF OP1>OP2 (JMP IF GREATER)
JNG	JMP IF OP1 <= OP2
JL	JMP IF OP1<OP2 (JMP IF LESS THAN)
JNL	JMP IF OP1>=OP2
JGE	JMP IF OP1>=OP2 (JMP IF GREATER THAN OR EQUAL)
JNGE	JMP IF OP1<OP2
JLE	JMP IF OP1<=OP2
JNLE	JMP IF OP1>OP2

أما هذه التعليمات فهي خاصة بالأعداد الموجبة فهي تعتبر أن آخر بت (أي MSB) هو جزء من العدد وليس دالا على الإشارة.

CMP OPERAND 1 , OPERAND 2

instruction	description
JA	JMP IF OP1>OP2 (JMP IF above)
JNA	JMP IF OP1 <= OP2
JB	JMP IF OP1<OP2 (JMP IF BELOW)
JNB	JMP IF OP1>=OP2
JAE	JMP IF OP1>=OP2 (JMP IF ABOVE OR EQUAL)
JNAE	JMP IF OP1<OP2
JBE	JMP IF OP1<=OP2
JNBE	JMP IF OP1>OP2

دعنا نوضح الأمر قليلا...
تأمل المقطع التالي :

```
MOV AL , -1
CMP AL,0
JG SECOND
FIRST:
MOV ECX,3
RET
SECOND:
XOR ECX,ECX
RET
```

إن **1-** يتم تمثيلها بالنظام الثنائي (بحجم بايت) كالتالي : 1111 1111 .
هناك عملية مقارنة بين ال **1-** وبين ال 0 ، واضح أن الشرط لن يتحقق فال **1-** ليست أكبر من 0 ، إذا لن تنفذ القفزة وبالتالي سنتوجه الى تعليمة MOV ECX,3 فيصبح ECX=3 ثم هناك تعليمة RET مما يعني إرجاع السيطرة الى النظام وإنهاء البرنامج .
لكن ألق نظرة على الكود التالي...نفس السابق لكن استخدمت JA بدلا من JG .

```
MOV AL , -1
CMP AL,0
JA SECOND
FIRST:
MOV ECX,3
RET
SECOND:
XOR ECX,ECX
RET
```

إن **1-** يتم تمثيلها بالنظام الثنائي (بحجم بايت) كالتالي : 1111 1111 لكن بالنسبة لتعليمة JA فهذه لا تعني **1-** لأننا هنا نتعامل مع أعداد موجبة إذن البت الأخيرة MSB ليست لتدل على الإشارة بل هي جزء من الرقم...إذن لو حولنا القيمة 1111 1111 الى عشري سنجدها تساوي 255 ...
والآن هل 255 أكبر من 0 ؟ بكل تأكيد...إذا فقد تحقق الشرط...وسننتقل الى التعليمة XOR ECX,ECX التي ستجعل ECX=0 ثم هناك تعليمة RET للعودة للنظام..

إذن في حالة تعليمة JG حصلنا على ECX=3 ، لكن في حالة JA حصلنا على ECX=0

II- variables

المتغير هو مكان في الذاكرة له اسم وحجم معينين. يتم تعريف المتغيرات في مقطع `data`. من البرنامج .
أنظر الى الجدول التالي :

db	Define byte	1 byte = 8 bit
dw	Define word	2 byte = 16 bit
dd	Define double word	4 byte = 32 bit

- يتم تعريف متغير باسم `allko` (مثلا) و بحجم `word` (مثلا) وبقيمة ابتدائية `4445h` (مثلا) كالتالي :
`Allko dw 4445h`

- ماذا لو أردنا تعريف نفس المتغير لكن لا نريد إعطائه أي قيمة ابتدائية ؟ ستكون الصيغة كالتالي :
`Allko dw ?`
علامة الاستفهام تشير الى عدم وجود قيمة ابتدائية..لاحظ أنه اذا لم تعط قيمة ابتدائية للمتغير فعليك تعريفه في مقطع `data?`

- أيضا يمكنك تعريف مصفوفة (array) كالتالي :
`xxx dd 7,5,4,8,1`
هنا قمنا بتعريف مصفوفة مكونة من 5 عناصر وكل عنصر بحجم `dword` كما هو واضح.
و للوصول الى العنصر الثالث مثلا يمكننا عمل التالي :
`Add xxx[2],400`
لاحظ أن أول عنصر هو `xxx[0]` والثاني `xxx[1]` وهكذا...

- لنفرض أننا أردنا حجز 4 بايتات من الذاكرة ونريد وضع القيمة الابتدائية 450 في كل بايت ، ونريدهم جميعا تحت نفس الاسم...هذا ما علينا فعله :
`Alalme db 450,450,450,450`
لكن هناك طريقة أسهل كالتالي :
`Alalme db 4 dup (450)`
يمكننا عمل نفس الشيء لكن دون إعطاء قيمة ابتدائية...كالتالي:
`Alalme db 4 dup (?)`

- أما النصوص فيمكن تعريفها كالتالي :
`X db 'A'`
لاحظ أنه يجب تعريف الأحرف والنصوص (strings) كمتغيرات بحجم بايت ، أي يجب استخدام `db` .
ولاحظ أن الحرف أو الـ `string` توضع بين بين علامتي تنصيص مفردتين .

- تعريف `string` لا يختلف عن تعريف حرف...لاحظ :
`msg db 'allko is da best',0`
الفرق الوحيد هو وجود الصفر (NULL) في النهاية ليدل على انتهاء الـ `string` ...إن كنت قد تعاملت مع أي من لغات المستوى العلوي من قبل فأنت تعرف عن ماذا أتحدث.

- أمر آخر يجب عليك معرفته ، هو أنه بدلا من كتابة الحرف `A` (أو أي حرف آخر) يمكنك استخدام القيمة المقابلة له في جدول ASCII ...
`Y db 65`

الى هنا يكون درس اليوم قد انتهى. أتمنى منكم الدعاء لي بالتوفيق بالامتحانات :

Assignment:

I- What is the value of the SF , CF , ZF and OF flags after the following code is carried out?

```
MOV BX , 81E1 h
ADD BX , 0C0FD h
```

II- what is the value of EAX after the following code is carried out ?

```
MOV BX , -5
CMP BX , 5
JA SECOND
FIRST:
MOV EAX,3
RET
SECOND:
AND EAX,0
RET
```

III- Rewrite the following code without changing what it do.

```
MOV EAX,25H
MOV ECX,8
BEGINNING:
MOV BYTE PTR DS:[401200+ECX],EAX
DEC ECX
JNZ BEGINNING
RET
```

IV - define a word-sized variable , named "allko" , with a 8100 h initail value.

V - define a word-sized variable , named "allko2" , without an initail value.

VI – define an array of character , a,b,c and d .

VII – define a string of your choice.

References :

- 1- www.arabtem2000.com
- 2- www.emu8086.com

Arab Team for Reverse Engineering

αλλκο
August - 2006