بسم الله الرحمن الرحيم

نتابع اليوم من حيث توقفنا في الجزء الأول من هذا المقال, وسنتكلم اليوم إن شاء الله عن أنواع البيانات التي يمكن التعامل بها في FPU ونمط العنونة الخاص بكل نوع.

وهي ثلاث انواع, integers, floating point, packed BCD

نبدأ بسم الله بأول نوع وهو الأرقام الصحيحة Integer Data Type:

وهي كما نعلم الارقام التي لا تملك كسورا عشرية وهي عبارة عن ارقام مؤشرة بمعنى انه من الممكن ان تكون موجبة فيكون البت السابع 0 او ان تكون سالبة فيكون البت السابع 1 (راجع مقالات عن الاسمبلي لتفهم هذه النقطة وتجد ما تحتاجه في المواضيع المثبته)

وللتذكير, الارقام السالبة يمكن الحصول عليها عن طريق المتمم الثنائي 2's complement واضافه 1 للقيمة الناتجة عن المتمم الثنائي... المتمم الثنائي طبعا يمكن الحصول عليه من خلال قلب البتات الخاصة بالرقم, فالـ 0 يغدو 1 والعكس صحيح.

الآن, من المعروف ان حجم هذا النوع من البيانات هو (18446744073709551615)

وبالتالي يكون مجاله (±9223372036854775807)

بناء على هذا فإنه يمكن عنونة WORD, DWORD, QWORD في هذا النمط من البيانات.

WORD range	(16bit)	$\pm(2^{15}-1) = \pm 32767$
DWORD range	(32bit)	$\pm(2^{31}-1) = \pm 2147483647$
QWORD range	(64bit)	$\pm(2^{63}-1) = \pm 9223372036854775807$

أما فيما يتعلق بطرق العنونة المتبعة مع الأرقام الصحيحة والمتعلقة بـ FPU فهي شبيهة بـ CPU مع اختلاف بسيط وهو انه لا يمكن تنفيذ تعليمات FPU على هذه القيم لو كانت موجودة في أحد المسجلات العامة الخاصة بـ CPU, حيث يشترط ان تكون قيم الارقام الصحيحة المراد معالجتها من قبل FPU في احد عناوين الذاكرة وكأمثلة على هذا:

```
fxxx  var_name
fxxx  var_name[24]
fxxx  var_name[ebx]
fxxx  word ptr [eax]
fxxx  dword ptr [esi+12]
fxxx  qword ptr [edi+ebx]
fxxx  dword ptr [ebp+8]
```

ومن اجل معالجة القيم الموجودة في المسجلات العامة الخاصة بـ CPU يمكن اتباع الطريقة التالية:

```
push  eax
fxxx  dword ptr [esp]
pop   eax
```

ولأسباب عديدة يفضل ان يتم استخدام تعليمة fwait التي تطلب من CPU الانتظار الى ان يتم تفريغ البت الذي يدل على حالة الانشغال الخاصة بـ FPU اثناء تنفيذ بعض تعليماتها لتجنب حدوث اي خطأ لو تم استخدام تعليمات CPU و FPU مختلطة.

أرقام الفاصلة العشرية Floating Point Data Types:

وهي الأرقام التي تحوي على كسور عشرية (فواصل عشرية) فيها والتي لا تستطيع تعليمات CPU معالجتها بشكل مباشر وانما تحتاج الى المعالجة من قبل تعليمات FPU للحصول على قيم صحيحة.

تمتلك 3 أحجام مثل الارقام الصحيحة لكن هذه الاحجام هي:

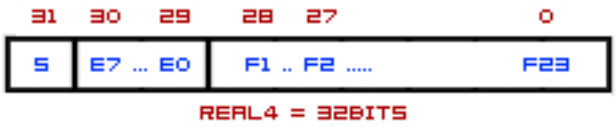
Note: aka = also known as

REAL4 (32bit, aka short real / single precision) $\pm(2^{32})$
REAL8 (64bit, aka long real / double precision) $\pm(2^{64})$
REAL10 (80bit, aka temporary real / extended precision) $\pm(2^{80})$

حاولت بشتى الوسائل ان اتجاوز النقطة المتعلقة بشرح تفصيلات بنية انماط البيانات التي يمكن تمثيلها للفاصلة العائمة لكن بعد قراءة المقطع المتعلق بها وجدت انه من الواجب علينا فهمها, أقله لزيادة معلوماتنا كمبر مجين من المفترض ان نلم بنمط البيانات التي نستطيع ان نتعامل بها بالاسمبلي... ام انا مخطئ؟

سأحاول تمثيل البيانات كما ورد في مقال Raymond وان اوجز واوضح ما وجدت من الضروري التفصيل فيه ان شاء الله.

REAL4:



S = Sign bit (0=positive, 1=negative)

بت الاشارة, وهو كما تعودنا في تمثيل اي رقم بالنظام الثنائي binary, 0 للقيم الموجبه و 1 للقيم السالبة (في النظام الثنائي لتمثيل الارقام يكون أول بت من اليسار)

E_n = biased exponent bits
F_n = fraction bits of the significant

E_n وهو عبارة عن مجموعة بتات تمثل قيمة تتعلق بالنظام العددي المستخدم وتشير الى عدد المرات التي يضرب فيها رقم بنفسه n مرة.. كمثال على هذا, 2^7, 10^5, حيث 5,7 هي الـ exponent (الأس).

للنمط REAL4 يكون عدد هذه البتات 8 بالتالي يمثلها بالنظام الست عشري بايت واحد فقط وهو القيمة 7Fh عندما يكون الأس 0 أي عدد مرات تكرار قيمة ما مرة واحدة ويكون التمثيل علميا:

x * y^0 = x

بالتالي, لو تم تمثيل رقم سالب, أصغر من الـ 1 فان قيمة هذا الأس تكون أصغر من 7Fh والعكس صحيح للأرقام أكبر من 1 الموجبة.

وللتنويه فان القيمة (11111111b) FFh للأس محجوزة لتمثيل الـ NAN (Not-A-Number) مثل اللانهاية.

وهناك تمثيل آخر عندما تكون جميع البتات (00000000b) أي 0h للأس, يطلق على القيمة تعبير denormalized وهي القيم الاصغر من 1 الذي يعتبر أصغر قيمة للارقام الحقيقية, سيتم تدعيم هذين التعبيرين بمزيد من الامثلة لاحقا.

أيضا يجب الانتباه بشكل ضروري الى ان اي قيمة صالحة للارقام الحقيقية يجب ان تبدأ بـ 1 أي أن تمثيل أي قيمة يبدأ بـ 1.0 ويتم إكمال الرقم من خلال الـ exponent فقط.

نعود الى موضوع الـ biased exponent, بالتالي لتمثيل اصغر قيمة للارقام الحقيقة التي هي +1.0:

{sign}	{exponent}	{fractions}
0	01111111 =7Fh	00000000000000000000000b

بالتالي يكون: (1.0 * 1^0) لتمثيل القيمة +1.0 وبالست عشري يكون: 3F800000h

هل تلاحظ أمر .. يثير الريبة .. مر معك سابقا ربما؟ تابع القراءة الآن وستدرك ما قد يكون قد فاتك لاحقا...

لتمثيل الرقم 2.0 (أي $1.0 * 2^1$):

{sign}	{exponent}	{fractions}
0	10000000 =7Fh+1	00000000000000000000000b

أما لو أردنا تمثيل -2.0 ($-1.0 * 2^1$):

{sign}	{exponent}	{fractions}
1	10000000 =7Fh+1	00000000000000000000000b

والآن لمزيد من التعقيد.. أقصد التوضيح.. قسمة 211- على 8 يكون الناتج... لا تخرج الآلة الحاسبة.. عيب p:

8 / 211- يمكن تمثيلها بالشكل التالي:

-211d = -11010011b = -1.1010011b * 2^7
Exponent, Number Base, 2^7 biased exponent
8d = 00001000b = 2^3
-211d / 8d = -1.1010011b * 2^7 / 2^3 = -1.1010011b * 2^4
Fractions

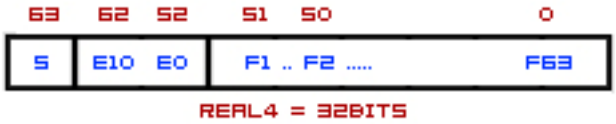
yup, damn it, but you have to understand it!

ولتمثيل الرقم النهائي بالـ REAL4:

{sign}	{exponent}	{fractions}
1	10000011 =7Fh+4	10100110000000000000000b

هل استوعبت الآن ما حصل؟ (لا تقلق, بقيت أدور حول نفسي حوالي الساعة حتى استوعبت ما حصل ثم لم أعد استوعب شيئا فقد غبت عن الوعي p:)

REAL8:



مشابه لـ REAL4 لكن بطول أكبر طبعا, نلاحظ ان البتات الخاصة بالـ exponent أصبحت 10 بدلا من 8 وتمثلها القيمة 3FFh في حال كان الأس صفري وتكون أكبر قيمة مسموحة لهذا النمط 7FFh والتي تكون محجوزة لـ NAN وبنفس الطريقة 0 من أجل denormal.

وكل ما تبقى مشابه لما ورد في شرح REAL4.

REAL10:

الاختلاف يكون في عدد بتات الـ exponent والتي تصبح 14بت والتي تمثلها القيمة 3FFFh والتي تملك حد أقصى هو 7FFFh من أجل تمثيل NANs.

أيضا البت 63 في هذا النمط يكون 1 والبتات الخاصة بالـ fractions تكون من البت 0 حتى 62.

طرق العنونة للأرقام الحقيقية:

كما نعلم فان REAL4 يمكن التصريح عنها في الاسمبلي على انها DWORD

REAL8 يصرح عنها على انها QWORD (Quadword)

REAL10 يصرح عنها على انها TBYTE (Terabyte)

وكأمثلة على طرق العنونة التي يجب ان تكون عبر عنوان ذاكرة.. لا تنسى

```
dword ptr [eax] ; EAX points to REAL4
dword ptr [esi+12] ; ESI points to an array of REAL4 values
                   ; ESI is the index of the array
qword ptr [edi+ebx] ; EDI or EBX points to an array of REAL8
values
tbyte ptr [edx] ; EDX points to REAL10 values
```

لاحظ كيف ان القيم كلها تمر عبر المسجلات التي تعنون البيانات في خانات ذاكرية وهذا شرط ضروري للتعامل بـ FPU.

تمثيل NANs:

وتشمل على اللانهاية (مؤشرة وغير مؤشرة), indefinite (غير معرّف), أخرى...

INFINITY:

تكون فيها البتات الخاصة بالـ exponent مساوية لـ 1

جميع البتات الخاصة بالـ fractions مساوية للصفر .

+INFINITY يكون فيها البت الخاص بالإشارة مساوي للصفر أي لانهاية موجبة.

-INFINITY يكون فيها البت الخاص بالإشارة مساوي للواحد أي لانهاية سالبة.

ويتم الحصول على هذا التمثيل من خلال:

1- قسمة عدد صحيح على الصفر (يقوم برفع استثناء zero divide exception)

2- الحصول على ناتج عملية ما أكبر من القيمة المسموحة (overflow exception)

3- محاولة حفظ قيمة في متغير بحجم أصغر من القيمة المراد تخزينها (overflow exception)

INDIFINITE:

تكون بتات الـ exponent مساوية لـ 1

جميع البتات الخاصة بالـ fractions مساوية للصفر فيما عدا أول بت.

بت الإشارة مساوي لـ 1.

يتم الحصول على هذا التمثيل من خلال:

1- استخدام قيمة غير معرّفه على أنها متحول.

2- استخدام مسجل فارغ على أنه متحول.

3- طرح لانهاية من لانهاية.

4- الجذر التربيعي لعدد سالب.

5- قسمة عدد صحيح على لانهاية.

Other NANs:
http://en.wikipedia.org/ > Search [NANs]

تمثيل denormalized numbers:

وهي الأرقام التي تعاني من فقدان للفاصلة العائمة عند معالجتها بالعمليات الرياضية (فقدان الفاصلة العائمة يعني أن الرقم يصبح أصغر مما كان عليه لو قمت بتربيعه مثلا)

ومن الواضح اننا نتكلم عن الارقام اصغر من 1

فعلى سبيل المثال ناتج تربيع القيمة 0.5 سيعطي النتيجة 0.25 (قيمة أصغر)

و 0.3^2 سيعطي 0.09 بالتالي نلاحظ كيف الـ fractions في تمثيل الـ REAL(x) يتم ازاحتها نحو اليمين.

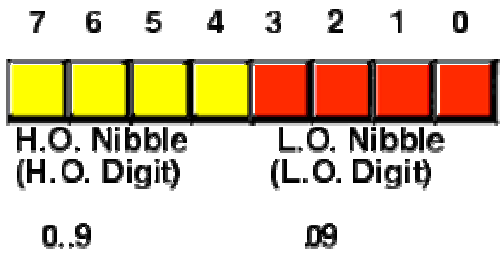
لن اعتمد المثال الوارد في مقال Raymond لأني أشعر أني لو بدأت باستخدام كل هذه الأمثلة لابتعدت عن هدف المقال الأساسي وهو FPU.

تمثيل Packed BCD:

BCD = Binary Coded Decimal

هممممم, ماذا استطيع ان اقول عنها باختصار...

يتالف هذا النمط من 8 خانات



كل خانة تستطيع ان تحمل قيمة عشرية من 0 < 9.

لكن في FPU فانه يتم اعتبارها على انها عدد صحيح مؤشر يملك حجم 80bit

بالتالي لتمثيل رقم مثل 211 فان التمثيل يكون بالشكل:

00000000000000000211h

ولتمثيل قيمة سالبة مثل 65536- فالتمثيل يكون بالشكل:

80000000000000065536h

لذلك لاحظ ان القيم السالبة تحمل القيمة 80h في الـ most significant byte
بينما يكون هذا البايت 0h من اجل القيم الموجبة.
وكما نعلم (او اصبحنا نعلم) فان هذا النمط من البيانات يتم حفظه في الذاكرة بتمثيل Big Endian (Motorola) وتصبح بالشكل:

36 55 06 00 00 00 00 00 00 80

اما عنونة هذا النمط من البيانات فيتم عبر تصريح TBYTE.

نتابع في الجزء الثالث ان شاء الله مع التعليمات التي تؤثر على البنية الداخلية لـ FPU والتي لا تتعلق بالعمليات الرياضية.