بسم الله الرحمن الرحيم

هذا المقال يدين بوجوده إلى **Raymond Filiatreault**.

- سبب كتابة هذا المقال؟

الملل, الرغبة بتجاوز بعض الأخطاء الشخصية والتغاضي عنها, الحاجة إلى استنشاق بعض الهواء, ما شأنك أنت؟

مقدمة إلى FPU:

FPU اختصاراً لـ Floating Point Units التعليمات الموجهة للتعامل مع الأرقام الحقيقة والفاصلة العائمة فمن المعروف لدى مبرمجي الأسمبلي أنه من غير الممكن التصريح عن رقم معين يحوي على فاصلة عشرية والتعامل معه كأي قيمة أخرى عددية.. على سبيل المثال:

```
.data
fpoint        db      0.5

.code
mov   eax, fpoint
...
```

بالنتيجة عند القيام بتجميع هذه الأسطر ستحصل على التالي:

Address	Hex dump	ASCII
00402000	00 00 00 3F 00 00 00 00 00 00 00 00 00 00 00 00	...?.....
00402010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

هل تشاهد تلك القيمة؟ أنت تتساءل الآن "اللعنة, ما هذه القيمة؟!"

في الواقع لا أدري, من المفترض أن تكون مساوية لـ 0.5 لكن هذه لا تبدو لي 0.5 ... ولو قمت بتنفيذ السطر الذي كتبته أعلاه لأصبح EAX = 3F000000h وهذا خطأ!

لذا من الواضح أن التعامل مع أرقام الفاصلة العائمة والأرقام الحقيقة أمر لا يمكن أن يتم مباشرة عبر برنامج التجميع المستخدم ولا بد من استخدام تعليمات FPU.

الصورة العامة لعمل FPU:

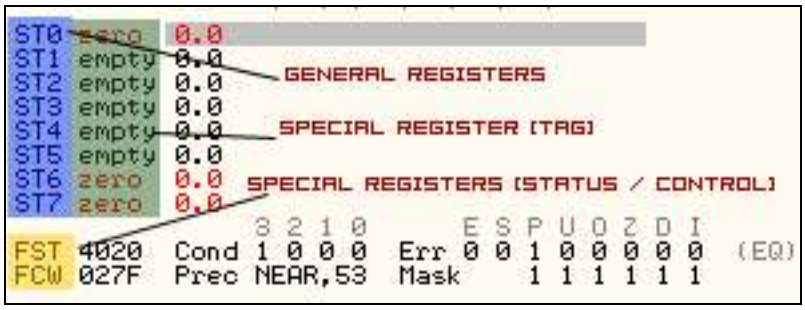
لربما تمر على مصطلح المعالج المساعد, وهو مجرد دلالة أخرى على هذه التعليمات, والآن لنتكلم قليلا عن هذه التعليمات.

في البداية يجب أن تعرف أن هذه التعليمات تتعامل مع الأرقام الحقيقية بطول 80bit حصراً أي REAL 10 ومن غير المجدي التصريح عن بعض الأرقام بأنها LONG REAL (64bit =

double) أو SHORT REAL (32bit = float) لأنه سيتم تحويلها آلياً إلى طول 80bit ولن تكسب من عمليتك أي فائدة تذكر, لربما يكون هذا الأمر من مساوئ FPU لأنها تتطلب المزيد من الذاكرة.

والآن بما أننا صورنا FPU بأنها تشبه المعالج, يمكن القول بأنها تملك عدداً من المسجلات العامة عددها 8 بطول 80bit لكل مسجل وبعض المسجلات ذات الاستخدام الخاص عددها 3 بطول 16bit لكل مسجل.

هذه المسجلات تظهر في Olly بالشكل التالي:



المسجلات من ST0 حتى ST7 هي المسجلات العامة التي يتم فيها تحميل القيم العددية للتعامل معها, بينما تختص المسجلات الثلاثة الباقية (Status word / Control word / Tag word) في تحديد حالات خاصة سواء تعطى من قبل المبرمج أو تحدد بعض خصائص القيم الموجودة في المسجلات العامة. سنأتي على التفصيل فيما يخص هذا لاحقا فلا تقلق.

المسجلات العامة:

هذه المسجلات من ST0 إلى ST7 يتم تحميلها بالقيم العددية لكن بصورة تختلف عما تعودنا عليه في مسجلات الأغراض العامة EAX,ECX,EBX,EDX والاختلافات هي بالشكل التالي للتبسيط:

FPU	CPU
طول المسجل 80bit حصرا ولا يمكن التعامل مع اجزاء هذا الطول	طول المسجل 32bit ويمكن التعامل مع 16bit منها او 8bit منها على حدة
لا يمكن تحديد مسجل معين لتحميله بقيمة	يمكن تحديد اي مسجل عام لتحميله بقيمة
تفريغ اي مسجل او تحميله بقيمة جديدة لا يمكن ان يتم مباشرة	تفريغ اي مسجل او تحميله بقيمة جديدة يمكن ان يتم مباشرة

لذلك نجد انه هناك بعض الصعوبة في التعامل مع FPU حيث انه يجب الانتباه الى بعض الامور كي لا نبدأ بارتكاب الاخطاء القاتلة اثناء كتابة كتلة معينة فيها.

فكيف يتم تحميل القيم في هذه المسجلات وكيف يقوم المعالج المساعد بعمله لتهيئتها؟

لنفرض اننا نريد تحميل قيمة 1 الى المسجلات العامة في FPU, ان هذا يتم باستعمال التعليمة fld1 فقط التي لا تملك بارامترات خاصة, عند تنفيذ هذه التعليمة فان المسجلات تبدو على الشكل:

```
ST0 valid 1.0000000000000000000
ST1 empty -UNORM BA1C 01050104 00000000
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 1.0000000000000000000
               3 2 1 0      E S P U O Z D I
FST 7820  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

لاحظ كيف تم تحميل المسجل ST0 بالقيمة 1 وكيف ان الـ Tag تحولت الى valid دلالة على ان القيمة صالحة للتعامل معها.

الان لنفرض اننا نريد تحميل قيمة جديدة هي 0.5 في المسجل ST4

في الواقع لا يمكن هذا, فكما ذكرت اعلاه, لا يمكن تحديد مسجل معين لتحميل قيمة فيه, كل ما نستطيع عمله هو تحميل القيمة والمعالج المساعد سيتولى مسؤولية وضعها في المسجلات.. لاحظ ما سيحدث عند تحميل هذه القيمة:

```
ST0 valid 0.5000000000000000000
ST1 valid 1.0000000000000000000
ST2 empty -UNORM D1D8 01050104 00000000
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 7020  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

انظر اين اصبحت القيمة الجديدة, يتم تحميل كل القيم في المسجل العلوي, ويشير ST0 دوما الى المسجل العلوي الذي يتم تدويره في البنية الداخلية للمعالج الى الخانة الفارغة التالية لذلك تجد ان القيمة اصبحت في ST0 وهو ما يشير الى الخانة او المسجل العلوي الحالي, بينما اصبح ST1 يشير الى الخانة او المسجل السابق والذي يحمل القيمة 1 التي قمنا بتحميلها للتو..

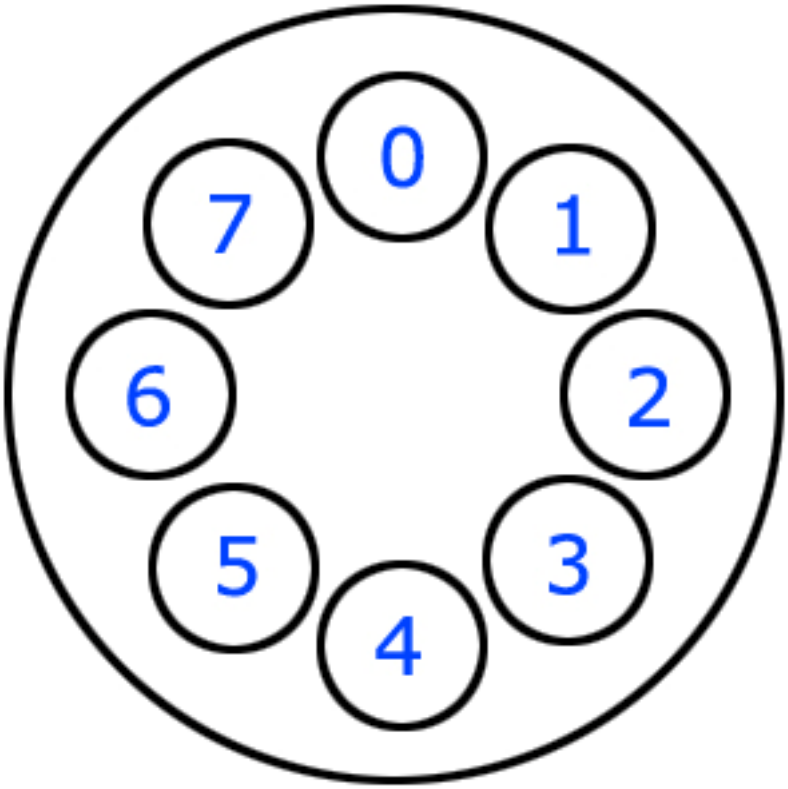
اذا, أي عملية تحميل للقيم الآن ستؤدي الى تحميل القيمة الجديدة في الخانة التي سيشير اليها ST0 (دوما) وستصبح القيمة 0.5 في الخانة التي يشير اليها ST1 والقيمة 1 في الخانة التي يشير اليها ST2

```
ST0 valid 177.38200378417968640
ST1 valid 0.50000000000000000000
ST2 valid 1.00000000000000000000
ST3 empty -UNORM BA1C 01050104 00000000
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 6820  Cond 1 0 0 0  Err 0 0 1 0 0 0 0 0  (EQ)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

هل ترى كيف تتم العملية؟

يجدر التنويه الان في هذه النقطة الى مسألة هامة, ماذا يحدث للقيمة الموجودة في الخانة العلوية التي يشير اليها ST0 لو اردنا التعامل مع القيمة التي تسبقها؟

في الواقع ما يحدث يمكن تشبيهه بماسورة Magnum

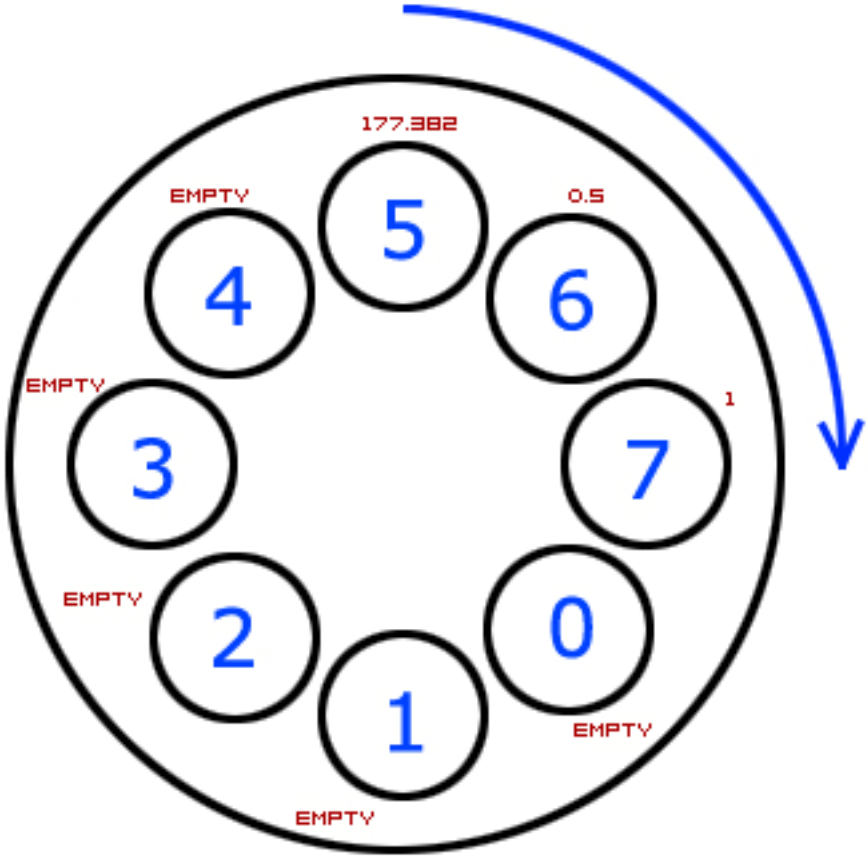


فعند تحميل قيمة ما في FPU فانه يتم اجراء عملية تدوير للخانات اولا (لا تخلط بين الارقام والدلالات ST0,...,ST7), فتعليمة fld1 تقوم بتدوير الشكل اعلاه خانة الى اليمين لتصبح الخانة 7 هي العلوية وليتم تحميلها بالقيمة 1

الخانة 7 هي العلوية لكن ما يكون هو ان ST0 تشير اليها (هذا ما قصدته بعدم الخلط بين الدلالات)

ثم يتم تحميل القيمة 0.5 بتدوير الخانات مرة اخرى خانة الى اليمين لتصبح الخانة العلوية هي 6 وليتم تحميلها بالقيمة بينما تأخذ الخانة 7 مكان الخانة 1 الحالي

ثم يتم تحميل القيمة 177.382 الى الخانة 5 بعد تدويرها لتكون هي الخانة العلوية ويكون الشكل النهائي:



الان لو اردنا التعامل مع القيمة 0.5 اي ما يشار اليها بواسطة ST1 فاننا نقوم بتدوير ماسورة الـ Magnum خانة واحدة الى اليسار عكس عقارب الساعة وبالتالي تبق القيم كما هي لكن تصبح المسجلات التي تدل عليها مختلفة:

قبل		بعد	
ST0	177.382	ST0	0.5
ST1	0.5	ST1	1
ST2	1	ST2	empty
ST3	empty	ST3	empty
ST4	empty	ST4	empty
ST5	empty	ST5	empty
ST6	empty	ST6	empty
ST7	empty	ST7	177.382

لذا فان ما يحدث هو القيمة التي تكون في الخانة العلوية المشار اليها دوما بـ ST0 تدور لتصبح في الخانة المشار اليها بـ ST7

وبنفس الطريقة, لو اردنا التعامل مع القيمة 1 فان المحتوى يصبح بالشكل:

قبل		بعد	
177.382	ST0	1	ST0
0.5	ST1	empty	ST1
1	ST2	empty	ST2
empty	ST3	empty	ST3
empty	ST4	empty	ST4
empty	ST5	empty	ST5
empty	ST6	177.382	ST6
empty	ST7	0.5	ST7

قد تستصعب فهم الامر لكنه سهل جدا عند كتابة التعليمات والتعامل مع قيمها لانك بكل بساطة لا تهتم بهذه التفاصيل حينها وانما يهمك ان تتعامل مع القيم الصحيحة في الوقت الصحيح وبما ان مختلف تعليمات FPU تتم على قيمتين فقط فكل ما ستتعامل معه هو المسجلين ST0, ST1 ونادرا ما تصل الى ST2 ☺

فقط تذكر, القيمة التي يتم تحميلها تصبح في ST0 والقيمة السابقة تدور لتصبح في ST1 وهكذا...

لكن, من الجدير بالذكر انه بالامكان طبعا تنفيذ العمليات الرياضية على اي مسجلات نريد دون الحاجة لجعل القيم الخاصة بها تكون في الخانات العلوية.. بمعنى, لو اردت تنفيذ عملية ضرب مثلا بين المسجل ST0, ST2 اي بين القيمتين 177.382 و 1 فان هذا ممكن دون الحاجة لجعل القيمة 1 تحتل المسجل ST1, الامر مشابه لما هو الحال عليه عند اجراء عمليات الضرب في CPU لكن مع اختلاف بسيط هو انك ملزم بوضع المضروب في EAX, المضروب به في EBX, النتيجة تكون في EAX بعد التنفيذ والباقي في EDX

في FPU, نتيجة عملية الضرب تكون دوما في الخانة العلوية (بغض النظر ما هي) التي يشار اليها دوما بـ ST0 ويمكن ان يكون المضروب والمضروب به في اي خانة نريد (جيد ☺) ولا يوجد باق, ببساطة لان FPU تسمح للارقام التي تحملها مسجلاتها ان تحوي على فاصلة عشرية خلافا لـ CPU (ممتاز ☺)

والان لنلق نظرة اعمق على المسجلات الخاصة الباقية...

المسجلات الخاصة Tag / Status / Control:

Tag word:

نبدأ بالبسيط منها وهو مسجل Tag او Tag word (word دلالة على انه بطول 16bit) من اسم المسجل فهو يزود كل مسجل عام بدلالة او لافتة خاصة تدل على المحتوى في ذلك المسجل.. يقسم المسجل Tag الى ثمانية اقسام ثانوية كل منها بطول 2bit ولكل منها دلالة خاصة, يعطي كل قسم دلالة لكل مسجل عام مقابل له (الشكل التالي يختلف مع ما في مقال Raymond لكن الشكلين للتوضيح لذا لا تعتقد ان احدهما خاطئ وتختلط عليك الامور)

TAG [0]	TAG [1]	TAG [2]	TAG [3]	TAG [4]	TAG [5]	TAG [6]	TAG [7]
ST0	ST1	ST2	ST3	ST4	ST5	ST6	ST7

00 = قيمة عددية صالحة لا تساوي الصفر

01 = قيمة عددية صالحة تساوي الصفر

10 = المسجل يحوي على قيمة خاصة

11 = المسجل لا يحوي على أي قيمة

لنستوعب الفكرة اكثر, لنفرض اننا الان قمنا بتهيئة مسجلات FPU, اي ستكون المسجلات الان فارغة من اي قيمة, بالتالي ستحمل جميع الاقسام القيمة 11:

1111111111111111b

قمنا الان بتحميل القيمة 1 الى المسجل ST0, بالتالي تدور الماسورة (مجازيا للتذكير) خانة الى اليمين, الخانة 7 تحمل الآن القيمة 1, المسجل ST0 ما زال يشير اليها (دوما, تذكر هذا), لكن الـ Tag الخاص بالخانة 7 هو Tag [7] لذا تغدو القيمة في مسجل Tag على النحو التالي:

0011111111111111b

لان القيمة التي تم تحميلها عددية صالحة لا تساوي الصفر, لنفرض الان اننا قمنا بتحميل القيمة العددية 0, القيمة ستتوضع في الخانة 6 وستصبح هذه الخانة هي العلوية وما زال ST0 يشير اليها وتصبح قيمة الـ Tag:

0100111111111111b

هل لاحظت ما جرى, ST[x] ثابت دوما, الخانات الداخلية للمسجلات العامة تدور عند تحميل قيم جديدة وتدور معها الخانات الخاصة بمسجل Tag, أتمنى ان يكون الشكل التالي أوضح:

01	00	11	11	11	11	11	11
TAG [6]	TAG [7]	TAG [0]	TAG [1]	TAG [2]	TAG [3]	TAG [4]	TAG [5]
ST0	ST1	ST2	ST3	ST4	ST5	ST6	ST7

ST0 ما زال يشير الى الخانة 6 والتي هي الخانة العلوية الآن

Tag[6] دارت مع تحميل القيمة الثانية الجديدة لتصبح ايضا الخانة العلوية بين الخانات الخاصة بمسجل Tag

01 هي قيمة الخانة Tag[6] والتي تشير الى ان القيمة الموجودة في المسجل العلوي الآن مساوية للصفر.

صورة للتوضيح من Olly, لاحظ ما هو مسجل الى جانب اسماء المسجلات العامة, empty, هذا يعني حسب ما تعلمناه ان القيمة الخاصة بالـ Tag [x] المقابل هي 11b

```
ST0 empty ---- TAG [0]
ST1 empty ---- TAG [1]
ST2 empty ---- TAG [2]
ST3 empty ---- TAG [3]
ST4 empty ---- TAG [4]
ST5 empty ---- TAG [5]
ST6 empty ---- TAG [6]
ST7 empty ---- TAG [7]
```

والصورة الحالية للقيم التي قمنا بتحميلها في مثالنا, لاحظ كيف تتغير عبارة الـ Tag بناء على القيمة التي تحملها الخانة المقابلة للمسجل العام...

```
ST0 zero  0.0            TAG [6] = 01
ST1 valid 1.000000000    TAG [7] = 00
ST2 empty -UNORM BA1C    TAG [0] = 11
ST3 empty 0.0            TAG [1] = 11
ST4 empty 0.0            TAG [2] = 11
ST5 empty 0.0            TAG [3] = 11
ST6 empty 0.0            TAG [4] = 11
ST7 empty 0.0            TAG [5] = 11
```

أعلم أن ما تفكر به الآن هو كيفية الاستفادة من هذا المسجل, او ما الغاية المرجوة منه بأي حال؟ في الواقع لا يمكن الوصول الى القيمة الموجودة في هذا المسجل بشكل مباشر للقراءة, ولا يمكن الكتابة فيه على الاطلاق.. فللوصول اليه للقراءة يجب اولا ان نقوم بحفظ حالة المسجلات في مكان ما في الذاكرة بطول 28byte باستخدام تعليمة fstenv والتي ستقوم بحفظ الكتلة التالية في الذاكرة

+0	WORD	**Control Word**
+2	WORD	(unused)
+4	WORD	**Status Word**
+6	WORD	(unused)
+8	WORD	**Tag Word**
+10	WORD	(unused)
+12	DWORD	**Instruction Pointer**
+16	WORD	**Code Segment**
+18	WORD	(unused)
+20	DWORD	**Operand Address**
+24	WORD	**Data Segment**
+26	WORD	(unused)
+28	TBYTE	**ST(0)**
+38	TBYTE	**ST(1)**
+48	TBYTE	**ST(2)**
+58	TBYTE	**ST(3)**
+68	TBYTE	**ST(4)**
+78	TBYTE	**ST(5)**
+88	TBYTE	**ST(6)**
+98	TBYTE	**ST(7)**

Fig.3.1 FPU Save State Format

لاحظ انه يمكن عندها قراءة القيمة الخاصة بالـ Tag word عند الازاحة 8 من بداية عنوان الذاكرة المخصص للكتلة وهو بطول بايتين كما تعلم.

هذا كل ما يتعلق بالمسجل Tag.

Control word:

يمكّن هذا المسجل المبرمج من انتقاء طريقة معينة للتعامل مع البيانات التي يتم تحميلها في المسجلات العامة, يقسم الى 16 خانة خلافا لـ Tag word,

			IC	RC		PC		IEM		PM	UM	OM	ZM	DM	IM
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

الخانات الفارغة هي خانات غير مستخدمة, نفترض انها reserved, ناتي الآن الى تفصيل مهمة كل خانة من الخانات الاخرى...

IC = **I**nfinity **C**ontrol

0 يشير الى ان يتم معاملة ∞+ و ∞- على انها غير مؤشرة بالتالي هي موجبة دوما (مجازياً موجبة)

1 يشير الى ان يتم معاملة قيمة اللانهاية وفقا لإشارتها.

RC = **R**ounding **C**ontrol

00 تدوير إلى الأقرب (الحالة الافتراضية)

01 تدوير إلى الأسفل

10 تدوير إلى الأعلى

11 ازالة الفاصلة العائمة من الرقم لو وجدت (أي اهمال كل ما يلي الفاصلة العائمة)

PC = **P**recision **C**ontrol

يحدد نمط الرقم الذي سيتم التحويل اليه (التعامل معه في المسجلات) بعد كل عملية رياضية

00 يعني 24bit اي REAL4 (short real = single precision)

01 غير مستخدم

10 يعني 53bit اي REAL8 (long real = double precision)

11 يعني 64bit اي REAL10 (الحالة الافتراضية) (temporary real = extended precision)

IEM = **I**nterrupt **E**nable **M**ask

لم يعد مستخدما في المعالجات الحديثة

وهو يتألف من 6bit جميعها تحمل القيمة 1 في الحالة الافتراضية, ولو كان احدها يحمل القيمة 0 فانه يطلب من FPU ان تقوم بمعالجة الاستثناء المتعلق بهذا الـ bit بصورة خاصة قبل ان يتم اعادة التحكم الى FPU

هذه البتات الست تشير الى الخانات الست المتبقية في Control word والمسماة: IM / DM / ZM / OM / UM / PM

لمزيد من المعلومات عنها... يرجى مراجعتها بنفسك فهي لم تعد مهمة كثيرا بشكل عام.

يمكن الوصول الى قيمة هذا المسجل بنفس الطريقة المتبعة مع المسجل السابق شرحه, لكن باستخدام تعليمة اخرى هي fstcw والتي تقوم بحفظ البيانات المتعلقة فيه في الذاكرة بحجم 16bit, طبعا!

سأتجاوز شرح الـ Status word في هذه المرحلة وتقسيماتها فهي مبعثة على الضجر جدا بصراحة, يكفي للوقت الحالي ان تعرف انها تتعلق بالاستثناءات التي تحصل وتشبه في عملها الى حد كبير الـ flags التي تعودنا عليها في الـ CPU.

سأقوم بذكر تأثير كل تعليمة من التعليمات على خانات هذا المسجل بعد شرح كل تعليمة وفي سياق الذكر سأوضح بعض الأمور المتعلقة بخانات هذا المسجل.

ايضا يمكن الوصول الى قيمة هذا المسجل عبر تعليمة هي fstsw.

بديهيات:

fstsw, fstcw, fstenv, الا تلاحظ انها اختصار لـ **f**pu **st**ore **env**ironment

fpu store control word, fpu store status word؟

لا تعتقد انك تحفظ ما يشبه الرموز تعليمات الاسمبلي التي نعرفها ليست بأفضل وما تعليمات الـ FPU الا اختصارات ومقاربات للكلمات الانكليزية أيضا!

إلى هنا أنهي الجزء الأول من هذا المقال على أمل أن أتابع ما بدأته غداً إن شاء الله تعالى.

السلام عليكم ورحمة الله وبركاته.

REKCAX

JUNE 1 2007 / 2:27AM