

١ المقاطعات Interrupts

المقاطعات هي طريقة لإيقاف المعالج بشكل مؤقت من تنفيذ عملية ما (Current Process) والبدء بتنفيذ أوامر أخرى . وكمثال على ذلك هو عند الضغط على أي حرف في لوحة المفاتيح فان هذا يولد مقاطعة (Interrupt) تأتي كإشارة الى المعالج بأن يوقف ما يعمل عليه حالياً ويحفظ كل القيم التي يحتاجها لكي يستطيع مواصلة ما تم قطعه ، وفي حالة وجود دالة للتعامل مع هذه المقاطعة (مقاطعة لوحة المفاتيح) وتسمى دالة معالجة المقاطعة (Interrupt Handler) أو دالة خدمة المقاطعة (Interrupt Service Routine) فان التنفيذ ينتقل اليها تلقائياً ، و يتم فيها معالجة هذه المقاطعة (مثلاً يتم قراءة الحرف الذي تم ادخاله من متحكم لوحة المفاتيح ومن ثم إرساله الى متغير في الذاكرة) وعندما تنتهي دالة معالجة المقاطعة من عملها فان المعالج يعود ليكمل تنفيذ العملية التي كان يعمل عليها. والمقاطعات إما تكون مقاطعات عتادية (Hardware Interrupt) وتصدر من عتاد الحاسب أو تكون برمجية (Software Interrupt) وتصدر من خلال البرامج عن طريق تعليمة `int n`. كذلك هناك مقاطعات يصدرها المعالج نفسه عند حدوث خطأ ما (مثلاً عن القسمة على العدد صفر أو عند حدوث Page Fault) وتسمى هذه المقاطعات بأخطاء المعالج أو استثناءات المعالج (Exceptions) ويجب معالجة هذه الأخطاء (Error Handler) لأنها توقف عمل النظام في حالة لم تتوفر دالة لمعالجتها.

١.١ المقاطعات البرمجية Software Interrupts

المقاطعات البرمجية هي مقاطعات يتم اطلاقها من داخل البرنامج (عن طريق الأمر `int n`) لنقل التنفيذ الى دالة أخرى تعالج هذه المقاطعة (Interrupt handler)، وغالباً ما تستخدم هذه المقاطعات في برامج المستخدم (Ring3 user mode) للاستفادة من خدمات النظام (مثلاً للقراءة والكتابة في أجهزة الإدخال والإخراج حيث لا توجد طريقة أخرى لذلك في نمط المستخدم).

١.١.١ المقاطعات في النمط الحقيقي

في النمط الحقيقي عندما يتم تنفيذ أمر المقاطعة (وهو ما يسمى بطلب تنفيذ المقاطعة (Interrupt Request) وتختصر بـ IRQ) فان المعالج يأخذ رقم المقاطعة المطلوب تنفيذها ويذهب بها الى جدول المقاطعات (Interrupt Vector Table) ، هذا الجدول يبدأ من العنوان الحقيقي `0x0` وينتهي عند العنوان `0x3ff` ويحوي كل سجل فيه على عنوان دالة معالجة المقاطعة (IR) والتي يجب تنفيذها لتخدم المقاطعة المطلوبة. حجم العنوان هو أربع بايت وتكون كالتالي:

جدول ١.١ : Interrupt Vector Table

Description	Interrupt Number	Base Address
Divide by 0	0	0x000
Single step (Debugger)	1	0x004
Non Maskable Interrupt (NMI) Pin	2	0x008
Breakpoint (Debugger)	3	0x00C
Overflow	4	0x010
Bounds check	5	0x014
Undefined Operation Code	6	0x018
No coprocessor	7	0x01C
Double Fault	8	0x020
Coprocessor Segment Overrun	9	0x024
Invalid Task State Segment (TSS)	10	0x028
Segment Not Present	11	0x02C
Stack Segment Overrun	12	0x030
General Protection Fault (GPF)	13	0x034
Page Fault	14	0x038
Unassigned	15	0x03C
Coprocessor error	16	0x040
Alignment Check (486+ Only)	17	0x044
Machine Check (Pentium/586+ Only)	18	0x048
Reserved exceptions	19-31	0x05C
Interrupts free for software use	32-255	0x068 - 0x3FF

- Byte 0: Low offset address of IR.
- Byte 1: High offset address of IR.
- Byte 2: Low Segment address of IR.
- Byte 3: High Segment Address of IR.

ويتكون الجدول من 256 مقاطعة (وبحسبة بسيطة يكون حجم الجدول هو 1024 بايت وهي ناتجة من ضرب عدد المقاطعات في حجم كل سجل)، بعض منها محجوز والبعض الآخر يستخدمه المعالج والبقية متروكة لمبرمج نظام التشغيل لدعم المزيد من المقاطعات. وبسبب أن الجدول يتكون فقط من عناوين لدوال معالجة المقاطعات فإن هذا يمكننا من وضع الدالة في أي مكان على الذاكرة ومن ثم وضع عناونها داخل هذا السجل (يتم هذا عن طريق مقاطعات البايوس)، والجدول ١.١ يوضح IVT والمقاطعات الموجودة فيه.

٢.١.١ المقاطعات في النمط المحمي

في النمط المحمي يستخدم المعالج جدولاً خاصاً يسمى بجدول واصفات المقاطعات (Interrupt Descriptor Table) ويختصر ب IDT ، هذا الجدول يشابه جدول IVT حيث يتكون من 256 واصفة كل واصفة مخصصة لمقاطعة ما (إذاً الجدول يحوي 256 مقاطعة) ، حجم كل واصفة هو 8 بايت تحوي عنوان دالة معالجة المقاطعة (IR) و نوع الناخب (selector type: code or data) في جدول GDT الذي تعمل عليه دالة معالجة المقاطعة ، بالإضافة الى مستوى الحماية المطلوب والعديد من الخصائص توضحها التركيبة التالية.

- Bits 0-15:
 - Interrupt / Trap Gate: Offset address Bits 0-15 of IR
 - Task Gate: Not used.
- Bits 16-31:
 - Interrupt / Trap Gate: Segment Selector (Useually 0x10)
 - Task Gate: TSS Selector
- Bits 31-35: Not used
- Bits 36-38:
 - Interrupt / Trap Gate: Reserved. Must be 0.
 - Task Gate: Not used.
- Bits 39-41:
 - Interrupt Gate: Of the format 0D110, where D determines size
 - * 01110 - 32 bit descriptor
 - * 00110 - 16 bit descriptor
 - Task Gate: Must be 00101
 - Trap Gate: Of the format 0D111, where D determines size
 - * 01111 - 32 bit descriptor
 - * 00111 - 16 bit descriptor
- Bits 42-44: Descriptor Privilege Level (DPL)
 - 00: Ring 0
 - 01: Ring 1
 - 10: Ring 2
 - 11: Ring 3

- Bit 45: Segment is present (1: Present, 0: Not present)
- Bits 46-62:
 - Interrupt / Trap Gate: Bits 16-31 of IR address
 - Task Gate: Not used

والمثال التالي يوضح انشاء واصفة واحدة بلغة التجميع حتى يسهل تتبع القيم ، وسيتم كتابة مثال كامل لاحقاً بلغة السي.

Example ١.١: Example of interrupt descriptor

```

١
٢ idt_descriptor:
٣     baseLow      dw    0x0
٤     selector     dw    0x8
٥     reserved     db    0x0
٦     flags        db    0x8e          ; 010001110
٧     baseHi       dw    0x0

```

المتغير الأول baseLow هو أول 16 بت من عنوان دالة معالجة المقاطعة IR ويكمل الجزء الآخر من العنوان المتغير baseHi وفي هذا المثال العنوان هو 0x0. بمعنى أن دالة تستخدم المقاطعة ستكون في العنوان 0x0. وبما أن دالة معالجة (تخدم) المقاطعة تحوي شفرة برمجية للتنفيذ وليست بيانات (Data) فإن قيمة المتغير selector يجب أن تكون 0x8 للإشارة إلى ناخب الشفرة (Code Selector) في جدول الواصفات العام (GDT). أما المتغير flags فإن قيمته هي 010001110b دلالة على أن الواصفة هي 32-bit وأن مستوى الحماية هو الحلقة صفر (Ring0).

وبعد أن يتم انشاء أغلب الواصفات بشكل متسلسل (في أي مكان على الذاكرة) ، يجب أن ننشئ جدول IDT وهذا يتم عن طريق حفظ عنوان أول واصفة في متغير وليكن idt_start وعنوان نهاية الواصفات في المتغير idt_end ومن ثم انشاء مؤشراً يسمى idt_ptr والذي يجب أن يكون في صورة معينة بحيث يحفظ عنوان بداية الجدول ونهايته :

Example ١.٢: Value to put in IDTR

```

١ idt_ptr:
٢     limit dw idt_end - idt_start ; bits 0-15 is size of idt
٣     base dd idt_start          ; base of idt

```

هذا المؤشر يجب أن يتم تحميله إلى المسجل IDTR (وهو مسجل داخل المعالج) عن طريق تنفيذ الأمر `lidt [idt_ptr]` بالشكل التالي.

بعد تنفيذ هذا الأمر فإن جدول المقاطعات سيتم استبداله بالجدول الجديد والذي نجد عنوانه بداخل المسجل idtr ، وهذا الأمر لا يُنفذ إلا إذا كانت قيمة العلم (CPL flag) هي صفر.

وعند حدوث أي مقاطعة فإن المعالج ينهي الأمر الذي يعمل عليه و يأخذ رقم المقاطعة ويذهب به الى جدول IDT (عنوان هذا الجدول يتواجد بداخل المسجل IDTR) ، وبعد ذلك يقوم بحساب مكان الوصفة بالمعادلة $int_num * 8$ وذلك بسبب أن حجم كل واصفة في جدول IDT هو 8 بايت. وقبل أن ينقل التنفيذ الى دالة معالجة المقاطعة فإنه يجب أن يقوم بعملية حفظ للمكان الذي توقف فيه حتى يستطيع أن يتابع عمله عندما تعود دالة معالجة المقاطعة . ويتم حفظ الأعلام EFLAGS ومسجل مقطع الشفرة CS ومسجل عنوان التعليمات التالية IP في المكس (Stack) الحالي ، وفي حالة حدوث خطأ ما فإنه يتم دفع شفرة الخطأ (Error Code) الى المكس أيضا. وشفرة الخطأ هي بطول 32-bit وتتبع التركيبة التالية.

- Bit 0: External event
 - 0: Internal or software event triggered the error.
 - 1: External or hardware event triggered the error.
- Bit 1: Description location
 - 0: Index portion of error code refers to descriptor in GDT or current LDT.
 - 1: Index portion of error code refers to gate descriptor in IDT.
- Bit 2: GDT/LDT. Only use if the descriptor location is 0.
 - 0: This indicates the index portion of the error code refers to a descriptor in the current GDT.
 - 1: This indicates the index portion of the error code refers to a segment or gate descriptor in the LDT.
- Bits 3-15: Segment selector index. This is an index into the IDT, GDT, or current LDT to the segment or gate selector bring referenced by the error code.
- Bits 16-31: Reserved.

وعندما تنتهي دالة معالجة المقاطعة من عملها فإنه يجب أن تنفذ الأمر `iret` أو `iretd` حتى يتم ارجاع القيم التي تم دفعها الى المكس (قيم الأعلام FLAGS). وبالتالي يُكمل المعالج عمله.

٣.١.١ أخطاء المعالج

خلال تنفيذ المعالج للأوامر فإنه ربما يحدث خطأ ما مما يجعل المعالج يقوم بتوليد استثناء يعرف باستثناء المعالج ، ويوجد له عدة أنواع:

- الخطأ Fault: عندما تعمل دالة معالجة هذا النوع من الاستثناء فربما يتم اصلاح هذا الخطأ ، وعنوان العودة الذي يتم دفعه الى المكس هو عنوان الأمر الذي تسبب في هذا الخطأ.
- الخطأ Trap: عنوان العودة هو عنوان التعليمات التي تلي الأمر الذي تسبب في الخطأ.

جدول ٢.١ : x86 Processor Exceptions Table

Description	Class	Interrupt Number
Divide by 0	Fault	0
Single step	Trap/Fault	1
Non Maskable Interrupt (NMI) Pin	Unclassed	2
Breakpoint	Trap	3
Overflow	Trap	4
Bounds check	Fault	5
Unvalid OPCode	Fault	6
Device not available	Fault	7
Double Fault	Abort	8
Coprocessor Segment Overrun	Abort	9
Invalid Task State Segment	Fault	10
Segment Not Present	Fault	11
Stack Fault Exception	Fault	12
General Protection Fault	Fault	13
Page Fault	Fault	14
Unassigned	-	15
x87 FPU Error	Fault	16
Alignment Check	Fault	17
Machine Check	Abort	18
SIMD FPU Exception	Fault	19
Reserved	-	20-31
Available for software use	-	32-255

● الخطأ Abort: لا يوجد عنوان للعودة ، ولن يكمل البرنامج عمله بعد انتهاء دالة معالجة الخطأ.

والجدول ٢.١ يوضح أخطاء المعالج والمقاطعات التي يقوم بتوليدها.

ويجدر بنا الوقوف على ملاحظة كئنا قد ذكرناها في الفصول السابقة وهي إلغاء المقاطعات (بواسطة الأمر cli) عند الانتقال الى النمط المحمي حتى لا يتسبب في حدوث خطأ General Protection Fault وبالتالي توقف النظام عن العمل وسبب ذلك هو أن عدم تنفيذ الأمر cli يعني أن المقاطعات العادية مفعلة وبالتالي أي عتاد يمكنه أن يرسل مقاطعة الى المعالج لكي ينقل التنفيذ الى دالة تخدمها . وعند بداية الانتقال الى النمط المحمي فان جدول المقاطعات IDT لم يتم انشائه وأي محاولة لاستخدامه سيؤدي الى هذا الخطأ. أحد المتحكمات التي ترسل مقاطعات الى المعالج بشكل ثابت هو متحكم Programmable Interval Timer وتختصر بـ PIT وهي تمثل ساعة النظام System Timer بحيث ترسل مقاطعة بشكل دائم الى المعالج والذي بدوره ينقل التنفيذ الى دالة تخدم هذه المقاطعة . وبسبب أن جدول المقاطعات غير متواجد في بداية المرحلة الثانية من محمل النظام وكذلك لا توجد دالة لتخدم هذه المقاطعة فان هذا يؤدي الى توقف النظام ،

لذلك يجب إيقاف المقاطعات العادية لحين إنشاء جدول المقاطعات وكتابة دوال معالجة المقاطعات. كذلك توجد مشكلة أخرى لبعض المقاطعات العادية حيث أنها تستخدم نفس أرقام المقاطعات التي يستخدمها المعالج للإستثناءات وحلها هو بإعادة برمجة الشريحة المسؤولة عن استقبال الاشارات من العتاد وتحويلها الى مقاطعات وارسالها الى المعالج ، هذه الشريحة تسمى Programmable Interrupt Controller وتختصر ب PIC ويجب إعادة برمجتها وتغيير ارقام المقاطعات للأجهزة التي تستخدم أرقاماً متشابهة. وفيما يلي سيتم إنشاء جدول المقاطعات (IDT) باستخدام لغة السي وتوفير ال 256 دالة لمعالجة المقاطعات وحاليا سيقصر عمل الدوال على طباعة رسالة ، وقبل ذلك سنقوم بإنشاء جدول الواصفات العام (GDT) مجددا (أي سيتم الغاء الجدول الذي قمنا بإنشائه في مرحلة الاقلاع) وبعد ذلك سنبدأ في برمجة متحكم PIC وإعادة ترقيم مقاطعات الأجهزة وكذلك برمجة ساعة النظام لارسال مقاطعة بوقت محدد.

٤.١.١ إنشاء جدول الواصفات العام GDT

الهدف الرئيسي في نواة نظام التشغيل هي المحمولة على صعيد المنصات ، وهذا ما أدى الى اعتماد فكرة طبقة HAL والتي يقبع تحتها كل ما يتعلق بعتاد الحاسب وادارته وكل ما يجعل النظام معتمداً على معمارية معينة أيضاً نجده تحت طبقة HAL ، و جدول الواصفات العام – كما ذكرنا في الفصول السابقة- يحدد ويقسم لنا الذاكرة الرئيسية كأجزاء قابلة للتنفيذ وأجزاء تحوي بيانات وغيرها ، ونظراً لأن إنشاء هذا الجدول يعتمد على معمارية المعالج والأوامر المدعومة فيه فإنه يجب ان يقع تحت طبقة HAL^٢ وهذا يعني أن نقل النظام الى معمارية حاسوب آخر يتطلب فقط إعادة برمجة طبقة HAL . بداية سنبدأ بتصميم الواجهة العامة لطبقة HAL ويجب أن نراعي أن تكون الواجهة مفصولة تماماً عن التطبيق حتى يتمكن أي مطور من إعادة تطبيقها لاحقاً على معمارية حاسوب آخر.

Example ١.٣: include/hal.h:Hardware Abstraction Layer Interface

```

١
٢ #ifndef HAL_H
٣ #define HAL_H
٤
٥ #ifndef i386
٦ #error "HAL is not implemented in this platform"
٧ #endif
٨
٩ #include <stdint.h>
١٠
١١ #ifdef _MSC_VER
١٢ #define interrupt __declspec(naked)
١٣ #else

```

^٢ من منظور آخر هذه الجداول (GDT, LDT and IDT) هي جداول للمعالج لذلك يجب أن تكون في طبقة HAL.

```

١٤ #define interrupt
١٥ #endif
١٦
١٧ #define far
١٨ #define near
١٩
٢٠
٢١ /* Interface */
٢٢
٢٣ extern int _cdecl hal_init();
٢٤ extern int _cdecl hal_close();
٢٥ extern void _cdecl gen_interrupt(int);
٢٦
٢٧
٢٨ #endif // HAL_H

```

والياً واجهة طبقة HAL مكونة من ثلاث دوال تم الإعلان عنها بأنها extern وهذا يعني أن أي تطبيق (Implementation) لهذه الواجهة يجب أن يُعرّف هذه الدوال. الدالة الاولى هي hal_init() والتي تقوم بتهيئة العتاد وجدول المعالج بينما الدالة الثانية hal_close() تقوم بعملية الحذف والتحرير وأخيراً الدالة gen_interrupt والتي تم وضعها لغرض تجربة إرسال مقاطعة برمجية والتأكد من أن دالة معالجة المقاطعة تعمل كما يرام. نعود بالحدوث الى جدول الواصفات العام (GDT) ^٣ حيث سيتم انشائه بلغة السي وهذا ما سيسمح لنا باستخدام تراكيب عالية للتعبير عن الجدول و المؤشر مما يعطي وضوح ومقروئية أكثر في الشفرة. وسوف نحتاج الى تعريف ثلاث دوال^٤:

- الدالة i386_gdt_init: تقوم بتهيئة واصفة خالية وواصفة للشفرة ولليانات وكذلك انشاء مؤشر الجدول.
- الدالة i386_gdt_set_desc: دالة تهيئة الواصفة حيث تستقبل القيم وتعينها الى الواصفة المطلوبة.
- الدالة gdt_install: تقوم بتحميل المؤشر الذي يحوي حجم الجدول وعنوان بدايته الى المسجل GDTR.

والشفرة التالية توضح كيفية انشاء الجدول^٥.

Example ١.٤: hal/gdt.cpp:Install GDT

^٣ راجع ??.

^٤ لغرض التنظيم والتقسيم لا أكثر ولا أقل.

^٥ راجع شفرة النظام لقراءة ملف الرأس hal/gdt.h.


```

١
٢ #include <string.h>
٣ #include "gdt.h"
٤
٥ static struct gdt_desc _gdt[MAX_GDT_DESC];
٦ static struct gdtr _gdtr;
٧
٨
٩ static void gdt_install();
١٠
١١
١٢ static void gdt_install() {
١٣ #ifdef _MSC_VER
١٤     _asm lgdt [_gdtr];
١٥ #endif
١٦ }
١٧
١٨ extern void i386_gdt_set_desc(uint32_t index, uint64_t base, uint64_t
    limit, uint8_t access, uint8_t grand) {
١٩
٢٠     if ( index > MAX_GDT_DESC )
٢١         return;
٢٢
٢٣     // clear the desc.
٢٤     memset((void*)&_gdt[index], 0, sizeof(struct gdt_desc));
٢٥
٢٦     // set limit and base.
٢٧     _gdt[index].low_base = uint16_t(base & 0xffff);
٢٨     _gdt[index].mid_base = uint8_t((base >> 16) & 0xff);
٢٩     _gdt[index].high_base = uint8_t((base >> 24) & 0xff);
٣٠     _gdt[index].limit = uint16_t(limit & 0xffff);
٣١
٣٢     // set flags and grandularity bytes
٣٣     _gdt[index].flags = access;
٣٤     _gdt[index].grand = uint8_t((limit >> 16) & 0x0f);
٣٥     _gdt[index].grand = _gdt[index].grand | grand & 0xf0;
٣٦ }
٣٧
٣٨ extern gdt_desc* i386_get_gdt_desc(uint32_t index) {
٣٩     if ( index >= MAX_GDT_DESC )
٤٠         return 0;

```

```
٤١     else
٤٢         return &_gdt[index];
٤٣     }
٤٤
٤٥ extern int i386_gdt_init() {
٤٦
٤٧     // init _gdtr
٤٨     _gdtr.limit = sizeof(struct gdt_desc) * MAX_GDT_DESC - 1;
٤٩     _gdtr.base = (uint32_t)&_gdt[0];
٥٠
٥١     // set null desc.
٥٢     i386_gdt_set_desc(0,0,0,0,0);
٥٣
٥٤     // set code desc.
٥٥     i386_gdt_set_desc(1,0,0xffffffff,
٥٦         I386_GDT_CODE_DESC|I386_GDT_DATA_DESC|I386_GDT_READWRITE|
            I386_GDT_MEMORY,    // 10011010
٥٧         I386_GDT_LIMIT_HI|I386_GDT_32BIT|I386_GDT_4K           //
            11001111
٥٨
٥٩ );
٦٠
٦١     // set data desc.
٦٢     i386_gdt_set_desc(2,0,0xffffffff,
٦٣         I386_GDT_DATA_DESC|I386_GDT_READWRITE|I386_GDT_MEMORY,    //
            10010010
٦٤         I386_GDT_LIMIT_HI|I386_GDT_32BIT|I386_GDT_4K           // 11001111
٦٥ );
٦٦
٦٧     // install gdtr
٦٨     gdt_install();
٦٩
٧٠     return 0;
٧١ }
```

٢.١ متحكم المقاطعات القابل للبرمجة Programmable Interrupt Controller

السبب الرئيسي في تعطيل المقاطعات العتادية عند الانتقال الى النمط المحمي (PMode) هو بسبب عدم توفر دوال لمعالجة المقاطعات في تلك اللحظة ، وحتى لو قمنا بتوفير ال ٢٥٦ دالة لمعالجة المقاطعات فان هنالك مشكلة استخدام نفس رقم المقاطعة لأكثر من غرض ، فمثلا مؤقتة النظام PIT التي ترسل مقاطعات بشكل دائم تستخدم المقاطعة رقم ٨ والتي هي أيضا أحد استثناءات المعالج ، لذلك في كلتا الحالت سيتم استدعاء دالة تخدم واحدة وهو شيء مرفوض تماماً. لذلك الحل الوحيد هو بإعادة برمجة المتحكم المسؤول عن استقبال الإشارات من متحكمات العتاد وتعيين أرقام مختلفة بخلاف تلك الأرقام التي يستخدمها المعالج للأخطاء والاستثناءات ، هذا المتحكم (انظر الشكل ١.١) وظيفته هي استقبال إشارات من متحكمات العتاد ومن ثم يقوم بتحويلها الى أرقام مقاطعات تُرسل بعد ذلك الى المعالج الذي يقوم بنقل التنفيذ إليها ، ويعرف هذا المتحكم بمتحكم PIC اختصاراً ل Programmable Interrupt Controller ويعرف أيضا بالإسم 8259A ، وفي هذا البحث سنستخدم المسمى متحكم PIC.

شكل ١.١: متحكم المقاطعات القابل للبرمجة 8259A



١.٢.١ المقاطعات العتادية Hardware Interrupts

قبل أن نبدأ في الدخول في تفاصيل متحكم PIC يجب إعطاء نبذة عن المقاطعات العتادية حيث ذكرنا أنها مقاطعات تختلف عن المقاطعات البرمجية من ناحية أن مصدرها يكون من العتاد وليس من برنامج ما ، وهذا ما أدى الى ظهور لقب مسير للأحداث (Interrupt Driven) على أجهزة الحاسب. حيث قدما لم يكن هناك طريقة للتعامل مع العتاد إلا باستخدام حلقة برمجية (loop) على مسجل ما في متحكم العتاد حتى تتغير قيمته دلالة على أن هناك قيمة أو نتيجة قد جاءت من العتاد ، هذه الطريقة في التخاطب مع العتاد تسمى Polling^٦ وهي تضيق وقت المعالج في انتظار قيمة لا يُعرف هل ستظهر أم لا وقد تم إلغائها في التخاطب مع العتاد حيث الان أصبح أي متحكم عتاد يدعم إرسال الإشارات (وبالتالي المقاطعات) الى المعالج والذي قد يعمل على عملية أخرى ، وهكذا تم الاستفادة من وقت المعالج وأصبح التخاطب هو غير متزامن (Asynchronous) بدلاً من متزامن (Synchronous). وعندما يبدأ الحاسب في الإقلاع فان نظام البايوس يقوم بترقيم عتاد الحاسب وإعطاء رقم مقاطعة لكل متحكم وبسبب تكرار هذه الأرقام فانه يجب تغييرها لأرقام أخرى وهذا

^٦وتسمى أيضا ب Busy Waiting.

جدول ٣.١: مقاطعات العتاد لحواسيب x86

وصف	رقم المقاطعة	رقم المشبك (الدبوس)
المؤقتة Timer	0x08	IRQ0
لوحة المفاتيح	0x09	IRQ1
يُربط مع متحكم PIC ثانوي	0x0a	IRQ2
المنفذ التسلسلي ٢	0x0b	IRQ3
المنفذ التسلسلي ١	0x0c	IRQ4
منفذ التوازي ٢	0x0d	IRQ5
متحكم القرص المرن	0x0e	IRQ6
منفذ التوازي ١	0x0f	IRQ7
ساعة ال CMOS	0x70	IRQ8/IRQ0
CGA vertical retrace	0x71	IRQ9/IRQ1
محجوزة	0x72	IRQ10/IRQ2
محجوزة	0x73	IRQ11/IRQ3
محجوزة	0x74	IRQ12/IRQ4
وحدة FPU	0x75	IRQ13/IRQ5
متحكم القرص الصلب	0x76	IRQ14/IRQ6
محجوزة	0x77	IRQ15/IRQ7

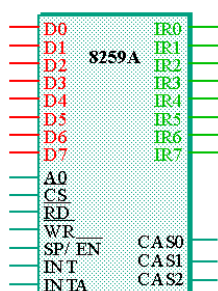
يتم بسهولة في النمط الحقيقي وذلك باستخدام مقاطعات البايوس أما في النمط المحمي فيجب أن نقوم بالتخاطب المباشر مع المتحكم الذي لديه أرقام المقاطعات ومن ثم تغييرها . والجدول ٣.١ يوضح أرقام المقاطعات لمتحكمات الحاسب.

٢.٢.١ برمجة متحكم PIC

متحكم PIC يستقبل إشارات (Signals) من متحكمات العتاد والتي تكون موصولة به ومن ثم يقوم بتحويلها الى أرقام مقاطعات لكي يقوم المعالج بنقل التنفيذ الى دالة تخدمها ، ويراعي متحكم PIC أولية متحكمات العتاد ، فمثلا لو تم إرسال إشارتين في نفس الوقت الى متحكم PIC فان المتحكم سوف يراعي الأولوية ويقوم بإرسال رقم مقاطعة العتاد ذو الأولوية أولا وبعد أن تنتهي دالة تخدم المقاطعة يقوم المتحكم بإرسال الرقم الآخر . ونظراً لتعقيدات بناء المتحكم فانه يتعامل فقط مع ٨ أجهزة مختلفة (أي ٨ مقاطعات IRQ) وهذا ما أدى مصنعى الحاسب الى توفير متحكم PIC آخر يعرف بالمتحكم الثانوي (Secondary/Slave PIC) . المتحكم الرئيسي (Primary PIC) يوجد داخل المعالج ويرتبط مع المتحكم الثانوي والذي يتواجد في الجسر الجنوبي (SouthBridge) .

مشابك المتحكم PIC's Pins

تعتبر مشابك المتحكم هي طريقة ارسال البيانات من المتحكم الى المعالج (أو الى متحكم رئيسي) ، ونظراً لان كل مشبك لديه وظيفة محددة فانه يجب دراسة هذه المشابك ولكن لن نفصل كثيراً حيث أن الموضوع متشعب ويخص دراسي المنطق الرقمي (Digital Logic). ويوضح الشكل ٢.١ هذه المشابك.



شكل ٢.١: مشابك متحكم PIC

حيث أن المشابك D0-D7 هي لإرسال البيانات الى متحكم PIC أما المشابك CAS0, CAS1, CAS2 تستخدم للتخاطب بين متحكمات PIC الرئيسية والثانوية ، والمشبك INT يرتبط مع مشبك للمعالج وهو INTR كذلك المشبك INTA يرتبط مع مشبك المعالج INTA وهذه المشابك لها العديد من الفوائد حيث عندما يقوم المعالج بتنفيذ أي مقاطعة فانه يقوم بتعطيل قيم العلمين IF and TF وهذا ما يجعل مشبك المعالج INTR يغلق مباشرة وبالتالي لا يمكن لمتحكم PIC إرسال أي مقاطعة عبر مشبكه INT حيث أن الجهة المقابلة لها تم غلقها وبالتالي لا يمكن لمقاطعة أن تقطع مقاطعة أخرى وإنما يتم حجبها في مسجل

داخل PIC الى أن ينتهي المعالج من تنفيذ المقاطعة والعودة بإشارة (تسمى إشارة نهاية المقاطعة End Of Interrupt) تدل على أن المقاطعة قد انتهت. أخيراً ما يهمنا في هذه المشابك هي مشابك IR0...IR7 وهي مشابك ترتبط مع متحكمات العتاد المراد استقبال الإشارات منه عند حدوث شيء معين (الضغط على حرف في لوحة المفاتيح مثلاً) ويمكن لهذه المشابك أن ترتبط مع متحكمات PIC أخرى ولا يوجد شرط ينص على وجوب توفر متحكمين PIC وإنما يمكن ربط كل مشبك من هذه المشابك الثمانية مع متحكم PIC وهكذا سيتواجد ٨ متحكمات تدعم حتى ٢٥٦ مقاطعة عتادية مختلفة. ويجب ملاحظة أن متحكم العتاد الذي يرتبط بأول مشبك IR0 لديه الأولوية الأولى في التنفيذ وهكذا على التوالي.

مسجلات متحكم PIC

يحتوي متحكم PIC على عدة مسجلات داخلية وهي:

- **مسجل الأوامر (Command Register):** ويستخدم لإرسال الأوامر الى المتحكم ، وهناك عدد من الأوامر مثل أمر القراءة من مسجل ما أو أمر ارسال إشارة EOI.
- **مسجل الحالة (Status Register):** وهو مسجل للقراءة فقط حيث تظهر عليه حالة المتحكم.
- **مسجل طلبات المقاطعات (Interrupt Request Register):** يحفظ هذا المسجل الأجهزة التي طلبت تنفيذ مقاطعتها وهي بانتظار وصول إشعار (Acknowledges) من المعالج ، والجدول ٤.١ يوضح بتات هذا المسجل.

جدول ٤.١: مسجل IRR/ISR/IMR

IRQ Number (Slave controller)	IRQ Number (Primary controller)	Bit Number
IRQ8	IRQ0	0
IRQ9	IRQ1	1
IRQ10	IRQ2	2
IRQ11	IRQ3	3
IRQ12	IRQ4	4
IRQ13	IRQ5	5
IRQ14	IRQ6	6
IRQ15	IRQ7	7

جدول ٥.١: عناوين المنافذ لمتحكم PIC

الوصف	رقم المنفذ
Primary PIC Command and Status Register	0x20
Primary PIC Interrupt Mask Register and Data Register	0x21
Secondary (Slave) PIC Command and Status Register	0xA0
Secondary (Slave) PIC Interrupt Mask Register and Data Register	0xA1

وفي حالة كانت قيمة أي بت هي ١ فهذا يعني أن متحكم العتاد بانتظار الإشعار من المعالج.

- مسجل الخدمة (In Service Register (ISR)): يدل على المسجل على أن طلب المقاطعة قد نجح وأن الإشعار قد وصل لكن لم تنتهي دالة تخدم المقاطعة من عملها.

- مسجل (Interrupt Mask Register (IMR)): يحدد هذا المسجل ما هي المقاطعات التي يجب تجاهلها وعدم ارسال إشعار لها وذلك حتى يتم التركيز على المقاطعات الأهم.

والجدول ٥.١ يوضح عناوين منافذ المسجلات في حواسيب x86.

برمجة متحكم PIC

لبرمجة متحكم PIC وإعادة ترقيم المقاطعات فإن ذلك يتطلب إرسال بعض الأوامر إلى المتحكم بحيث تأخذ هذه الأوامر نمط معين تُحدّد بها عمل المتحكم. وتوجد أربع أوامر تهيئة يجب إرسالها لتهيئة المتحكم تعرف ب Initialization Control Words وتختصر بأوامر تهيئة ICW ، وكذلك توجد ثلاث أوامر تحكم في عمل متحكم PIC تعرف ب Operation Control Words وتختصر ب OCW . وفي حالة توفر أكثر من متحكم PIC على النظام فيجب أن تُرسل أوامر التهيئة إلى المتحكم الآخر كذلك. الأمر الأول ICW1 وهو أمر التهيئة

جدول ٦.١: الأمر الأول ICW1

رقم البت	القيمة	الوصف
0	IC4	إرسال الأمر ICW4
1	SNGL	هل يوجد متحكم PIC واحد
2	ADI	تأخذ القيمة صفر في حواسيب x86
3	LTIM	نمط عمل المقاطعة
4	1	بت التهيئة
5	0	تأخذ القيمة صفر في حواسيب x86
6	0	تأخذ القيمة صفر في حواسيب x86
7	0	تأخذ القيمة صفر في حواسيب x86

الرئيسي والذي يجب إرساله أولاً إلى المتحكم الرئيسي والثانوي ويأخذ ٧ بتات ويوضح الجدول ٦.١ هذه البتات ووظيفة كل بت.

حيث أن البت الأول يحدد ما إذا كان يجب إرسال أمر التحكم ICW4 أم لا وفي حالة كان قيمة البت هي ١ فإنه يجب إرسال الأمر ICW4 أما البت الثاني فغالباً يأخذ القيمة صفر دلالة على أن هناك أكثر من متحكم PIC في النظام ، والبت الثالث غير مستخدم أما الرابع فيحدد نمط عمل المقاطعة هل هي Level Triggered أم Mode Edge Triggered Mode ، أما البت الخامس فيجب أن يأخذ القيمة ١ دلالة على أننا سنقوم بتهيئة متحكم PIC وبقية البتات غير مستخدمة في حواسيب x86. والشفرة ١.٥ توضح إرسال الأمر الأول إلى متحكم PIC الرئيسي والثانوي.

Example ١.٥: Initialization Control Words 1

```

١ ; Setup to initialize the primary PIC. Send ICW 1
٢ mov al, 0x11 ; 00010001
٣ out 0x20, al
٤
٥ ; Send ICW 1 to second PIC command register
٦ out 0xA0, al

```

الأمر الثاني ICW2 يستخدم لإعادة تغيير عناوين جدول IVT الرئيسية للطلبات المقاطعات IRQ وبالتالي عن طريق هذا الأمر يمكن أن نغير أرقام المقاطعات لل IRQ إلى أرقام أخرى . ويجب أن يرسل هذا الأمر مباشرة بعد الأمر الأول كذلك يجب أن يتم اختيار أرقاماً غير مستخدمة من قبل المعالج حتى لا تقع في نفس المشكلة السابقة (وهي أكثر من IRQ يستخدم نفس رقم المقاطعة وبالتالي لديهم دالة تخديم واحدة). والمثال ١.٦ يوضح كيفية تغيير أرقام IRQ لمتحكم PIC الرئيسي والثانوي بحيث يتم استخدام أرقام المقاطعات ٣٢-٣٩ للمتحكم الأول والأرقام من ٤٠-٤٧ للمتحكم الثانوي وهي أرقاماً خالية لا يستخدمها المعالج وتقع مباشرة بعد آخر مقاطعة للمعالج الذي يستخدم ٣٢ مقاطعة بدءاً من الصفر وانتهاءً بالمقاطعة ٣١.

جدول ٧.١: الأمر الثالث للمتحكم الرئيسي ICW3 for Primary PIC

رقم البت	القيمة	الوصف
0-7	S0-S7	رقم IRQ التي يتصل بها المتحكم الثانوي

جدول ٨.١: الأمر الثالث للمتحكم الثانوي ICW3 for Slave PIC

رقم البت	القيمة	الوصف
0-2	ID0	رقم IRQ التي يتصل بها مع المتحكم الرئيسي
3-7	3-7	محمولة

Example ١.٦: Initialization Control Words 2

```

١ ; send ICW 2 to primary PIC
٢ mov al, 0x20
٣ out 0x21, al
٤ ; Primary PIC handled IRQ 0..7. IRQ 0 is now mapped to interrupt
   number 0x20
٥
٦
٧ ; send ICW 2 to secondary PIC
٨ mov al, 0x28
٩ out 0xA1, al
١٠ ; Secondary PIC handles IRQ's 8..15. IRQ 8 is now mapped to use
    interrupt 0x28

```

الأمر الثالث ICW3 يستخدم في حالة كان هناك أكثر من متحكم PIC حيث يجب أن نحدد رقم طلب المقاطعة IRQ التي يستخدمها المتحكم الثانوي للتخاطب مع المتحكم الرئيسي. وفي حواسيب x86 غالباً ما يستخدم IRQ2 لذا يجب إرسال هذا الأمر الى المتحكم، لكن كل متحكم يتوقع الأمر بصيغة معينة يوضحها الجدولان ٧.١ و ٨.١.

ويجب إرسال الأمر بحسب الصيغة التي يقبلها مسجل البيانات للمتحكم ، فمتحكم PIC الرئيسي يستقبل رقم IRQ على شكل ٧ بت بحيث يتم تفعيل رقم البت المقابل لرقم IRQ وفي مثالنا يرتبط المتحكم الرئيسي مع الثانوي عبر IRQ2 لذلك يجب تفعيل قيمة البت ٢ (أي يجب إرسال القيمة 0000100b وهي تعادل 0x4) بينما المتحكم الثانوي يقبل رقم IRQ عن طريق إرسال قيمته على الشكل الثنائي وهي ٢ (وتعادل بالترميز الثنائي 010) وبقية البتات محمولة (انظر جدول ٨.١) ، والمثال ١.٧ يوضح كيفية إرسال الأمر الثالث الى المتحكمين.

Example ١.٧: Initialization Control Words 3

جدول ٩.١: الأمر الرابع ICW4

رقم البت	القيمة	الوصف
0	uPM	يجب تفعيل هذا البت في حواسيب x86
1	AEOI	جعل المتحكم يقوم بإرسال إشارة EOI
2	M/S	If set (1), selects buffer master. Cleared if buffer slave.
3	BUF	If set, controller operates in buffered mode
4	SFNM	تأخذ القيمة صفر في حواسيب x86
5-7	0	تأخذ القيمة صفر في حواسيب x86

```

١ ; Send ICW 3 to primary PIC
٢ mov al, 0x4 ; 0x04 => 0100, second bit (IR line 2)
٣ out 0x21, al ; write to data register of primary PIC
٤
٥ ; Send ICW 3 to secondary PIC
٦ mov al, 0x2 ; 010=> IR line 2
٧ out 0xA1, al ; write to data register of secondary PIC

```

الأمر الرابع ICW4 هو آخر أمر تحكم يجب إرساله الى المتحكمين ويأخذ التركيبة التي يوضحها جدول ٩.١.

وفي الغالب لا يوجد حوجة لتفعيل كل هذه الخصائص ، فقط أول بت يجب تفعيله حيث يستخدم مع حواسيب x86 . والمثال ١.٨ يوضح كيفية إرسال الأمر الرابع الى المتحكم PIC الرئيسي والثانوي.

Example ١.٨: Initialization Control Words 4

```

١ mov al, 1 ; bit 0 enables 80x86 mode
٢
٣ ; send ICW 4 to both primary and secondary PICs
٤ out 0x21, al
٥ out 0xA1, al

```

وبعد إرسال هذه الأوامر الأربع تكتمل عمليةتهيئة متحكم PIC الرئيسي والثانوي ، وفي حالة حدوث أي مقاطعة من متحكم لعتاد ما ، فإن أرقام المقاطعات التي سترسل الى المعالج هي الأرقام التي قمنا بتعيينها في الأمر الثاني (وتبدأ من ٣٢ الى ٤٧) وهي تختلف بالطبع عن الأرقام التي يستخدمها المعالج. وبخصوص أوامر التحكم الثلاث OCW فلن نحتاج اليها جميعاً وسيتم الحديث عن الأمر الثاني OCW2 نظراً لأنه يجب أن يُرسل دائماً بعد أن تنتهي دالة تقديم المقاطعة من عملها وذلك حتى يتم السماح لبقية المقاطعات أن تأخذ دوراً لمعالجتها. والجدول ١٠.١ يوضح البتات التي يجب إرسالها الى مسجل التحكم . وبهنا البتات ٥-٧ حيث أن قيمهم تحدد بعض الخصائص التي يوضحها الجدول ١١.١. والمثال ١.٩ يوضح كيفية إرسال إشارة نهاية

جدول ١٠.١: أمر التحكم الثاني OCW2

الوصف	القيمة	رقم البت
Interrupt level upon which the controller must react	L0/L1/L2	0-2
محجوزة	0	3-4
End of Interrupt (EOI)	EOI	5
Selection	SL	6
Rotation option	R	7

جدول ١١.١: أمر OCW2

Description	EOI Bit	SL Bit	R Bit
Rotate in Automatic EOI mode (CLEAR)	0	0	0
Non specific EOI command	1	0	0
No operation	0	1	0
Specific EOI command	1	1	0
Rotate in Automatic EOI mode (SET)	0	0	1
Rotate on non specific EOI	1	0	1
Set priority command	0	1	1
Rotate on specific EOI	1	1	1

عمل دالة تستخدم المقاطعة (EOI) حيث يجب ضبط البتات لإختيار Non specific EOI command.

Example ١.٩: Send EOI

```

١ ; send EOI to primary PIC
٢
٣ mov al, 0x20 ; set bit 4 of OCW 2
٤ out 0x20, al ; write to primary PIC command register

```

كيف تعمل مقاطعات العتاد

عندما يحتاج متحكم أي عتاد لفت انتباه المعالج الى شيء ما فأول خطوة يقوم بها هي إرسال إشارة الى متحكم PIC (وعلى سبيل المثال سنفرض أن هذا المتحكم هو متحكم المؤقتة PIT والتي ترتبط بالمشبك IRO) هذه الإشارة ترسل عبر مشبك IRO ، حينها يقوم متحكم PIC بتسجيل طلب المتحكم IRQ في مسجل يسمى مسجل طلبات المقاطعات (Interrupt Request Register) ويعرف اختصاراً بمسجل IRR . هذا المسجل بطول ٨ بت كل بت فيه يمثل رقم IRQ ويتم تفعيل أي بت عند طلب مقاطعة من المتحكم ، وفي مثالنا سيتم تفعيل البت 0 بسبب أن المؤقتة ترتبط مع IRO. بعد ذلك يقوم متحكم PIC بفحص مسجل Interrupt

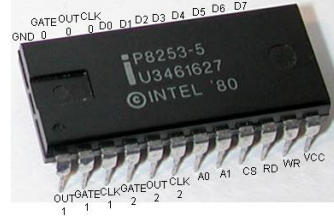
Mask Register ليتأكد من أنه لا توجد هناك مقاطعة ذات أولوية أعلى حيث في هذه الحالة على المقاطعة الجديدة أن تنتظر حتى يتم تخدم كل المقاطعات ذات الأولوية. وبعد ذلك يُرسل PIC إشارة إلى المعالج من خلال مشبك INTA لأخبار المعالج بأن هناك مقاطعة يجب تنفيذها. وهنا يأتي دور المعالج حيث يقوم بالإنتهاء من تنفيذ الأمر الحالي الذي يعمل عليه ومن ثم يقوم بفحص قيمة العلم IF حيث في حالة كانت غير مفعلة فإن المعالج سوف يتجاهل طلب تنفيذ المقاطعة، أما إذا وجد المعالج قيمة العلم مفعلة فإنه يقوم بإرسال إشعار (Acknowledges) عبر مشبك INTR إلى متحكم PIC الذي بدوره يستقبلها من مشبك INTA ويضع رقم المقاطعة ورقم IRQ في المشابك D0-D7 ، وأخيراً يفعل قيمة البت ٠ في مسجل In Service Register دلالة على أن مقاطعة المؤقتة جاري تنفيذها. وعندما يحصل المعالج على رقم المقاطعة فإنه يقوم بوقف العملية التي يعمل عليها ويحفظ قيم مسجل الأعلام ومسجل CS and EIP وإذا كان المعالج يعمل في النمط الحقيقي فإنه يأخذ رقم المقاطعة ويذهب بها كدليل إلى جدول المقاطعات IVT حيث يجد عنوان دالة تخدم المقاطعة ومن ثم ينقل التنفيذ إليها ، أما إذا كان المعالج يعمل في النمط المحمي فإنه يأخذ رقم المقاطعة ويذهب بها إلى جدول واصفات المقاطعات حيث يجد دالة تخدم المقاطعة. وعندما تنتهي دالة تخدم المقاطعة من عملها فإنها يجب أن ترسل إشارة EOI حتى يتم تفعيل المقاطعات مجدداً.

٣.١ المؤقتة Programmable Interval Timer

المؤقتة هي شريحة Dual Inline Package (DIP) تحوي ثلاث عدادات (Counters or Channels) تعمل كمؤقتات لإدارة ثلاث أشياء (انظر الشكل ٣.١). العداد الأول ويُعرف بمؤقت النظام (System Timer) وظيفته إرسال طلب مقاطعة (IRQ0) إلى متحكم PIC وذلك لتنفيذ مقاطعة ما كل فترة محددة ، هذه الفترة يتم تحديدها عند برمجة هذه المؤقتة ويُستفاد من هذه المؤقتة في عملية تزامن العمليات وتوفير بنية تحتية لمفهوم تعدد العمليات والمسالك (Multitask and Multithread) حيث أن الفترة التي تقوم بها مؤقتة النظام لإصدار طلب المقاطعة سيكون هو الوقت المحدد لأي عملية (Process) موجودة في طابور العمليات (Process Queue) وبعد ذلك تُرسل العملية إلى آخر الصف في حالة لم تنتهي من عملها بعد ويبدأ المعالج في تنفيذ العملية التالية تحت نفس الفترة المحددة. أما العداد الثاني فيستخدم في عملية تنعش الذاكرة الرئيسية (RAM refreshing) حتى تحافظ على محتوياتها من الفقدان أثناء عمل الحاسب ويجدر بنا ذكر أن هذه المهمة قد أُحيلت إلى متحكم الذاكرة (Memory Controller) وأصبحت هذه المؤقتة لا تستخدم في العادة. أما العداد الأخير فيستخدم في عملية إرسال الصوت إلى سماعات الحاسب^٧ (PC Speaker).

^٧ لا يُقصد بهذه كرت الصوت وإنما يوجد في كل حاسب سماعات داخلية تستخدم في إصدار الصوت والغمات وأحد استخداماتها لإصدار رسائل الخطأ بعد عملية فحص الحاسب (POST) في مرحلة الإقلاع.

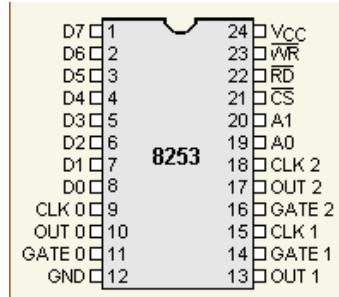
شكل ٣.١: المؤقتة القابلة للبرمجة 8253



١.٣.١ برمجة المؤقتة PIT

مؤخرًا تم نقل المؤقتة من اللوحة الأم (MotherBoard) كشريحة DIP مستقلة إلى الجسر الشمالي (SouthBridge). وسوف نركز على برمجة العداد الأول وهو مؤقت النظام حيث أنه يوفر الدعم العتادي اللازم للنظام حتى يدعم تعدد العمليات والمسالك.

مشابك المؤقتة PIT's Pins



شكل ٤.١: مشابك المؤقتة PIT

تُرسل الأوامر والبيانات إلى المؤقتة وذلك عبر مسار البيانات (Data Bus) حيث يرتبط هذا المسار مع مشابك البيانات في المؤقتة وهي ٨ مشابك D0...D7 وتمثل ٨ بتات. وعند إرسال بيانات إلى المؤقتة (عملية كتابة) فإن مشبك الكتابة WR يأخذ قيمة منخفضة دلالة على أن هناك عملية إرسال بيانات إلى المؤقتة وكذلك في حالة قراءة بيانات من المؤقتة فإن مشبك القراءة RD يأخذ قيمة منخفضة دلالة على أن هناك عملية قراءة من المؤقتة. ويتحكم في مشبك القراءة والكتابة مشبك CS حيث تحدد قيمته تعطيل أو تفعيل عمل المشبكين السابقين ، ويرتبط مشبك CS مع مسار العناوين (Address Bus) بينما يرتبط مشبك القراءة والكتابة مع مسار التحكم (Control Bus). وتحدد قيمة المشبكين A0,A1 -واللذان يرتبطان مع مسار العناوين- المسجلات المطلوب الوصول إليها داخل المؤقتة. أما المشابك (CLK, OUT, and GATE) فهي لكل عداد بداخل المؤقتة أي بمعنى أنه توجد ثلاث مشابك من كل واحدة منهم ، ويعتبر المشبكين (CLK (Clock Input) and GATE) مشابك إدخال للعداد بينما المشبك (OUT) مشبك إخراج حيث يستخدم لربط العداد مع العتاد فمثلا مشبك الإخراج في العداد الأول (مؤقتة النظام) يرتبط مع متحكم PIC حيث من خلاله تستطيع مؤقتة النظام إرسال طلب المقاطعة (IRQ0) إلى متحكم PIC والذي يقوم بتحويل الطلب إلى المعالج لكي ينفذ دالة التخدم.

مسجلات المؤقتة PIT

توجد ٤ مسجلات بداخل المؤقتة PIT ، ثلاث منها تستخدم للعدادات (الأول والثاني والثالث) حيث من خلالها يمكن قراءة قيمة العداد أو الكتابة فيه ، وطول مسجل العداد هو ١٦ بت . وبسبب أن مشابه البيانات التي تربط المؤقتة ومسار البيانات هي من الطول ٨ بت فإنه لن يتمكن من إرسال البيانات بهذا الشكل . لذلك يجب استخدام مسجل آخر وهو مسجل التحكم (Control Word) بحيث قبل إرسال بيانات أو قراءة بيانات من أي عداد فإنه يجب إرسال الأمر المطلوب إلى مسجل التحكم وبعد ذلك يتم إرسال البيانات أو قراءتها. والجدول ١٢.١ يوضح هذا المسجلات وعنوان منافذ الإدخال والإخراج المستخدمة للتعامل معها ، ويجب ملاحظة قيم خط القراءة والكتابة وخط العنوان (A0,A1) حيث تؤثر قيمهم في تحديد نوع العملية المطلوبة (قراءة أم كتابة ورقم العداد). وتوضح التركيبة التالية ماهية البتات المستخدمة في مسجل التحكم

جدول ١٢.١ : مسجلات المؤقتة PIT 8253

اسم المسجل	رقم المنفذ	خط RD	خط WR	خط A0	خط A1	الوظيفة
Counter 0	0x40	1	0	0	0	كتابة إلى المسجل 0 قراءة المسجل 0
Counter 1	0x41	1	0	0	1	كتابة إلى المسجل 1 قراءة المسجل 1
Counter 2	0x42	1	0	1	0	كتابة إلى المسجل 2 قراءة المسجل 2
Control Word	0x43	1	0	1	1	كتابة Control Word لا توجد عملية

(وهو مسجل بطول ٨ بت) حيث يجب إرسال قيم معينة حتى يتمكن من القراءة أو الكتابة في عداد ما.

- Bit 0: (BCP) Binary Counter
 - 0: Binary
 - 1: Binary Coded Decimal (BCD)
- Bit 1-3: (M0, M1, M2) Operating Mode. See above sections for a description of each.
 - 000: Mode 0: Interrupt or Terminal Count
 - 001: Mode 1: Programmable one-shot
 - 010: Mode 2: Rate Generator
 - 011: Mode 3: Square Wave Generator
 - 100: Mode 4: Software Triggered Strobe
 - 101: Mode 5: Hardware Triggered Strobe
 - 110: Undefined; Don't use

- 111: Undefined; Don't use
- Bits 4-5: (RL0, RL1) Read/Load Mode. We are going to read or send data to a counter register
 - 00: Counter value is latched into an internal control register at the time of the I/O write operation.
 - 01: Read or Load Least Significant Byte (LSB) only
 - 10: Read or Load Most Significant Byte (MSB) only
 - 11: Read or Load LSB first then MSB
- Bits 6-7: (SC0-SC1) Select Counter. See above sections for a description of each.
 - 00: Counter 0
 - 01: Counter 1
 - 10: Counter 2
 - 11: Illegal value

والمثال ١٠.١٠ يوضح كيفية برمجة عداد مؤقت النظام لإرسال طلب مقاطعة كل 100Hz (كل ١٠ milliseconds) ، وهذا يتم عن طريق إرسال أمر التحكم أولاً ومن ثم إرسال الوقت المطلوب الى العداد المطلوب.

Example ١٠.١٠: PIT programming

```

١          ; COUNT = input hz / frequency
٢
٣  mov dx, 1193180 / 100 ; 100hz, or 10 milliseconds
٤
٥  ; FIRST send the command word to the PIT. Sets binary counting,
٦  ; Mode 3, Read or Load LSB first then MSB, Channel 0
٧
٨  mov al, 110110b
٩  out 0x43, al
١٠
١١  ; Now we can write to channel 0. Because we set the "Load LSB first
    then MSB" bit, that is
١٢  ; the way we send it
١٣
١٤  mov ax, dx
١٥  out 0x40, al ;LSB
١٦  xchg ah, al
١٧  out 0x40, al ;MSB

```

٤.١ توسعة طبقة HAL

طبقة HAL تبعد نواة النظام من التعامل المباشر مع العتاد وتعمل كواجهة أو طبقة ما بين النواة والعتاد ، وفيها نجد تعريفات العتاد. وسيتم إضافة أوامر برمجة متحكم PIC التي تقوم بإعادة تعيين أرقام المقاطعات بداخل هذه الطبقة وكذلك سيتم إضافة شفرة برمجة المؤقتة و التي تحدد الوقت اللازم للمؤقتة لكي تقوم بارسال طلب المقاطعة (IRQ0) .

١.٤.١ دعم PIC

في القسم ٢.٢.١ تم عرض متحكم PIC وكيفية برمجته بالتفصيل ، وفي هذا القسم سيتم تطبيق ما تم عرضه على نواة نظام إقرأ. ويوجد ملفين لمتحكم PIC الأول هو ملف الرأس (hal/pic.h) الذي يحوي الإعلان عن الدوال وكذلك الثوابت والثاني هو ملف التطبيق (hal/pic.cpp) الذي يحوي على تعريف تلك الدوال. والمثال ١.١١ يعرض ملف الرأس الذي يغلف العديد من الأرقام والعناوين في صورة ثوابت (باستخدام الماكرو) بحيث تزيد من مقروئية ووضوح الشفرة^٨.

Example ١.١ : hal/pic.h: PIC Interface

```
١ // PIC 1 Devices IRQ
٢ #define I386_PIC_IRQ_TIMER      0
٣ #define I386_PIC_IRQ_KEYBOARD  1
٤ #define I386_PIC_IRQ_SERIAL2   3
٥ #define I386_PIC_IRQ_SERIAL1   4
٦ #define I386_PIC_IRQ_PARALLEL2 5
٧ #define I386_PIC_IRQ_DESKETTE  6
٨ #define I386_PIC_IRQ_PARALLEL1 7
٩
١٠ // PIC 2 Devices IRQ
١١ #define I386_PIC_IRQ_CMOSTIMER  0
١٢ #define I386_PIC_IRQ_CGARETRACE 1
١٣ #define I386_PIC_IRQ_AUXILIRY  4
١٤ #define I386_PIC_IRQ_FPU        5
١٥ #define I386_PIC_IRQ_HDC        6
١٦
١٧ // Operation Command Word 2 (OCW2)
١٨ #define I386_PIC_OCW2_MASK_L1   1
١٩ #define I386_PIC_OCW2_MASK_L2   2
٢٠ #define I386_PIC_OCW2_MASK_L3   4
```

^٨إرجع الى القسم ٢.٢.١ لمعرفة وظيفة هذه القيم الثابتة.

```
٢١ #define I386_PIC_OCW2_MASK_EOI      0x20
٢٢ #define I386_PIC_OCW2_MASK_SL      0x40
٢٣ #define I386_PIC_OCW2_MASK_ROTATE  0x80
٢٤
٢٥
٢٦ // Operation Command Word 3 (OCW3)
٢٧ #define I386_PIC_OCW3_MASK_RIS     1
٢٨ #define I386_PIC_OCW3_MASK_RIR     2
٢٩ #define I386_PIC_OCW3_MASK_MODE    4
٣٠ #define I386_PIC_OCW3_MASK_SMM     0x20
٣١ #define I386_PIC_OCW3_MASK_ESMM    0x40
٣٢ #define I386_PIC_OCW3_MASK_D7      0x80
٣٣
٣٤
٣٥ // PIC 1 port address
٣٦ #define I386_PIC1_COMMAND_REG       0x20
٣٧ #define I386_PIC1_STATUS_REG        0x20
٣٨ #define I386_PIC1_IMR_REG           0x21
٣٩ #define I386_PIC1_DATA_REG          0x21
٤٠
٤١
٤٢ // PIC 2 port address
٤٣ #define I386_PIC2_COMMAND_REG       0xa0
٤٤ #define I386_PIC2_STATUS_REG        0xa0
٤٥ #define I386_PIC2_IMR_REG           0xa1
٤٦ #define I386_PIC2_DATA_REG          0xa1
٤٧
٤٨ // Initializing Command Word 1 (ICW1) Mask
٤٩ #define I386_PIC_ICW1_MASK_IC4      0x1
٥٠ #define I386_PIC_ICW1_MASK_SNGL     0x2
٥١ #define I386_PIC_ICW1_MASK_ADI      0x4
٥٢ #define I386_PIC_ICW1_MASK_LTIM     0x8
٥٣ #define I386_PIC_ICW1_MASK_INIT     0x10
٥٤
٥٥
٥٦ // Initializing Command Word 4 (ICW4) Mask
٥٧ #define I386_PIC_ICW4_MASK_UPM      0x1
٥٨ #define I386_PIC_ICW4_MASK_AEOI     0x2
٥٩ #define I386_PIC_ICW4_MASK_MS       0x4
٦٠ #define I386_PIC_ICW4_MASK_BUF      0x8
٦١ #define I386_PIC_ICW4_MASK_SFNM     0x10
```



```

٦٢
٦٣
٦٤ // Initializing command 1 control bits
٦٥ #define I386_PIC_ICW1_IC4_EXPECT      1
٦٦ #define I386_PIC_ICW1_IC4_NO          0
٦٧ #define I386_PIC_ICW1_SNGL_YES        2
٦٨ #define I386_PIC_ICW1_SNGL_NO         0
٦٩ #define I386_PIC_ICW1_ADI_CALLINTERVAL4  4
٧٠ #define I386_PIC_ICW1_ADI_CALLINTERVAL8  0
٧١ #define I386_PIC_ICW1_LTIM_LEVELTRIGGERED 8
٧٢ #define I386_PIC_ICW1_LTIM_EDGETRIGGERED 0
٧٣ #define I386_PIC_ICW1_INIT_YES          0x10
٧٤ #define I386_PIC_ICW1_INIT_NO           0
٧٥
٧٦ // Initializing command 4 control bits
٧٧ #define I386_PIC_ICW4_UPM_86MODE        1
٧٨ #define I386_PIC_ICW4_UPM_MCSMODE       0
٧٩ #define I386_PIC_ICW4_AEOI_AUTOEOI      2
٨٠ #define I386_PIC_ICW4_AEOI_NOAUTOEOI    0
٨١ #define I386_PIC_ICW4_MS_BUFFERMASTER   4
٨٢ #define I386_PIC_ICW4_MS_BUFFERSLAVE     0
٨٣ #define I386_PIC_ICW4_BUF_MODEYES        8
٨٤ #define I386_PIC_ICW4_BUF_MODENO         0
٨٥ #define I386_PIC_ICW4_SFNM_NESTEDMODE    0x10
٨٦ #define I386_PIC_ICW4_SFNM_NOTNESTED     0
٨٧
٨٨
٨٩ extern uint8_t i386_pic_read_data(uint8_t pic_num);
٩٠ extern void i386_pic_send_data(uint8_t data, uint8_t pic_num);
٩١ extern void i386_pic_send_command(uint8_t cmd, uint8_t pic_num);
٩٢ extern void i386_pic_init(uint8_t base0, uint8_t base1);

```

وتحتوي الواجهة ٤ دوال منها دالتان للقراءة والكتابة من مسجل البيانات ودالة لإرسال الأوامر الى مسجل التحكم والدالة الأخيرة هي لتهيئة المتحكم وهي الدالة التي يجب استدعائها. والمثال ١.١٢ يوضح تعريف هذه الدوال.

Example ١.١٢: hal/pic.cpp: PIC Implementation

```

١ uint8_t i386_pic_read_data(uint8_t pic_num) {
٢     if (pic_num > 1)
٣         return 0;

```

```
٤
٥  uint8_t reg = (pic_num == 1)?I386_PIC2_DATA_REG:I386_PIC1_DATA_REG;
٦  return inportb(reg);
٧ }
٨
٩ void i386_pic_send_data(uint8_t data,uint8_t pic_num) {
١٠     if (pic_num > 1)
١١         return;
١٢
١٣     uint8_t reg = (pic_num == 1)?I386_PIC2_DATA_REG:I386_PIC1_DATA_REG;
١٤     outportb(reg,data);
١٥ }
١٦
١٧ void i386_pic_send_command(uint8_t cmd,uint8_t pic_num) {
١٨
١٩     if (pic_num > 1)
٢٠         return;
٢١
٢٢     uint8_t reg = (pic_num == 1)?I386_PIC2_COMMAND_REG:
        I386_PIC1_COMMAND_REG;
٢٣     outportb(reg,cmd);
٢٤ }
٢٥
٢٦
٢٧
٢٨ void i386_pic_init(uint8_t base0,uint8_t base1) {
٢٩
٣٠     uint8_t icw = 0;
٣١
٣٢     disable_irq();      /* disable hardware interrupt (cli) */
٣٣
٣٤     /* init PIC, send ICW1 */
٣٥     icw = (icw & ~I386_PIC_ICW1_MASK_INIT) | I386_PIC_ICW1_INIT_YES;
٣٦     icw = (icw & ~I386_PIC_ICW1_MASK_IC4) | I386_PIC_ICW1_IC4_EXPECT;
٣٧     /* icw = 0x11 */
٣٨
٣٩     i386_pic_send_command(icw,0);
٤٠     i386_pic_send_command(icw,1);
٤١
٤٢     /* ICW2 : remapping irq */
٤٣     i386_pic_send_data(base0,0);
```

```
٤٤ i386_pic_send_data(base1,1);
٤٥
٤٦ /* ICW3 : irq for master/slave pic*/
٤٧ i386_pic_send_data(0x4,0);
٤٨ i386_pic_send_data(0x2,1);
٤٩
٥٠ /* ICW4: enable i386 mode. */
٥١ icw = (icw & ~I386_PIC_ICW4_MASK_UPM) | I386_PIC_ICW4_UPM_86MODE ;
      /* icw = 1 */
٥٢ i386_pic_send_data(icw,0);
٥٣ i386_pic_send_data(icw,1);
٥٤
٥٥ }
```

٢.٤.١ دعم PIT

Example ١.١٣: hal/pit.h: Pit Interface

```
١ #define I386_PIT_COUNTER0_REG      0x40
٢ #define I386_PIT_COUNTER1_REG      0x41
٣ #define I386_PIT_COUNTER2_REG      0x42
٤ #define I386_PIT_COMMAND_REG       0x43
٥
٦ #define I386_PIT_OCW_MASK_BINCOUNT 0x1
٧ #define I386_PIT_OCW_MASK_MODE      0xe
٨ #define I386_PIT_OCW_MASK_RL        0x30
٩ #define I386_PIT_OCW_MASK_COUNTER   0xc0
١٠
١١
١٢ #define I386_PIT_OCW_BINCOUNT_BINARY 0x0
١٣ #define I386_PIT_OCW_BINCOUNT_BCD    0x1
١٤
١٥ #define I386_PIT_OCW_MODE_TERMINALCOUNT 0x0
١٦ #define I386_PIT_OCW_MODE_ONESHOT      0x2
١٧ #define I386_PIT_OCW_MODE_RATEGEN     0x4
١٨ #define I386_PIT_OCW_MODE_SQUAREWAVEGEN 0x6
١٩ #define I386_PIT_OCW_MODE_SOFTWARETRIG 0x8
٢٠ #define I386_PIT_OCW_MODE_HARDWARETRIG 0xa
٢١
```

```
٢٢ #define I386_PIT_OCW_RL_LATCH          0x0
٢٣ #define I386_PIT_OCW_RL_LSBONLY       0x10
٢٤ #define I386_PIT_OCW_RL_MSBONLY       0x20
٢٥ #define I386_PIT_OCW_RL_DATA          0x30
٢٦
٢٧ #define I386_PIT_OCW_COUNTER_0         0x0
٢٨ #define I386_PIT_OCW_COUNTER_1         0x40
٢٩ #define I386_PIT_OCW_COUNTER_2         0x80
٣٠
٣١ extern void i386_pit_send_command(uint8_t cmd);
٣٢ extern void i386_pit_send_data(uint16_t data, uint8_t counter);
٣٣ extern uint8_t i386_pit_read_data(uint16_t counter);
٣٤ extern uint32_t i386_pit_set_tick_count(uint32_t i);
٣٥ extern uint32_t i386_pit_get_tick_count();
٣٦ extern void i386_pit_start_counter(uint32_t freq, uint8_t counter,
    uint8_t mode);
٣٧ extern void _cdecl i386_pit_init();
٣٨ extern bool _cdecl i386_pit_is_initialized();
```

Example ١.١: hal/pit.cpp: PIT Implementation

```
١ static volatile uint32_t _pit_ticks = 0;
٢ static bool _pit_is_init = false;
٣
٤ void _cdecl i386_pit_irq();
٥
٦ void i386_pit_send_command(uint8_t cmd) {
٧     outportb(I386_PIT_COMMAND_REG, cmd);
٨ }
٩
١٠ void i386_pit_send_data(uint16_t data, uint8_t counter) {
١١     uint8_t port;
١٢
١٣     if (counter == I386_PIT_OCW_COUNTER_0)
١٤         port = I386_PIT_COUNTER0_REG;
١٥     else if (counter == I386_PIT_OCW_COUNTER_1)
١٦         port = I386_PIT_COUNTER1_REG;
١٧     else
١٨         port = I386_PIT_COUNTER2_REG;
١٩
```

```
٢٠ outportb(port,uint8_t(data));
٢١ }
٢٢
٢٣ uint8_t i386_pit_read_data(uint16_t counter) {
٢٤     uint8_t port;
٢٥
٢٦     if (counter == I386_PIT_OCW_COUNTER_0)
٢٧         port = I386_PIT_COUNTER0_REG;
٢٨     else if ( counter == I386_PIT_OCW_COUNTER_1)
٢٩         port = I386_PIT_COUNTER1_REG;
٣٠     else
٣١         port = I386_PIT_COUNTER2_REG;
٣٢
٣٣     return inportb(port);
٣٤ }
٣٥
٣٦ uint32_t i386_pit_set_tick_count(uint32_t i) {
٣٧     uint32_t prev = _pit_ticks;
٣٨     _pit_ticks = i;
٣٩     return prev;
٤٠ }
٤١
٤٢ uint32_t i386_pit_get_tick_count() {
٤٣     return _pit_ticks;
٤٤ }
٤٥
٤٦ void i386_pit_start_counter(uint32_t freq,uint8_t counter,uint8_t
    mode) {
٤٧     if (freq == 0)
٤٨         return;
٤٩
٥٠     uint16_t divisor = uint16_t(1193181/uint16_t(freq));
٥١
٥٢     /* send operation command */
٥٣     uint8_t ocw = 0;
٥٤
٥٥     ocw = (ocw & ~I386_PIT_OCW_MASK_MODE) | mode;
٥٦     ocw = (ocw & ~I386_PIT_OCW_MASK_RL) | I386_PIT_OCW_RL_DATA;
٥٧     ocw = (ocw & ~I386_PIT_OCW_MASK_COUNTER) | counter;
٥٨
٥٩     i386_pit_send_command(ocw);
```

```
٦٠
٦١  /* set frequency rate */
٦٢  i386_pit_send_data(divisor & 0xff,0);
٦٣  i386_pit_send_data((divisor >> 8) & 0xff,0);
٦٤
٦٥  /* reset ticks count */
٦٦  _pit_ticks = 0;
٦٧ }
٦٨
٦٩ void _cdecl i386_pit_init() {
٧٠     set_vector(32,i386_pit_irq);
٧١     _pit_is_init = true;
٧٢ }
٧٣
٧٤ bool _cdecl i386_pit_is_initialized() {
٧٥     return _pit_is_init;
٧٦ }
٧٧
٧٨ void _cdecl i386_pit_irq() {
٧٩
٨٠     _asm {
٨١         add esp,12
٨٢         pushad
٨٣     }
٨٤
٨٥     _pit_ticks++;
٨٦
٨٧     int_done(0);
٨٨
٨٩     _asm {
٩٠         popad
٩١         iretd
٩٢     }
٩٣ }
```

٣.٤.١ واجهة HAL الجديدة

المثال ١.١٥ يوضح الواجهة العامة لطبقة HAL

Example ١.١٥: New HAL Interface

```

١ extern int _cdecl hal_init();
٢ extern int _cdecl hal_close();
٣ extern void _cdecl gen_interrupt(int);
٤ extern void _cdecl int_done(unsigned int n);
٥ extern void _cdecl sound(unsigned int f);
٦ extern unsigned char _cdecl inportb(unsigned short port_num);
٧ extern void _cdecl outportb(unsigned short port_num, unsigned char
    value);
٨ extern void _cdecl enable_irq();
٩ extern void _cdecl disable_irq();
١٠ extern void _cdecl set_vector(unsigned int int_num, void (_cdecl far &
    vect)());
١١ extern void (_cdecl far * _cdecl get_vector(unsigned int int_num))();
١٢ extern const char* _cdecl get_cpu_vendor();
١٣ extern int _cdecl get_tick_count();

```

Example ١.١٦: New HAL Impelmentation

```

١ int _cdecl hal_init() {
٢     i386_cpu_init();
٣     i386_pic_init(0x20, 0x28);
٤     i386_pit_init();
٥     i386_pit_start_counter(100, I386_PIT_OCW_COUNTER_0,
        I386_PIT_OCW_MODE_SQUAREWAVEGEN);
٦
٧     /* enable irq */
٨     enable_irq();
٩
١٠    return 0;
١١ }
١٢
١٣ int _cdecl hal_close() {
١٤     i386_cpu_close();
١٥     return 0;
١٦ }
١٧
١٨ void _cdecl gen_interrupt(int n) {
١٩     #ifdef MSC_VER
٢٠     __asm {
٢١         mov al, byte ptr [n]

```

```
٢٢     mov byte ptr [address+1], al
٢٣     jmp address
٢٤
٢٥     address:
٢٦         int 0    // will execute int n.
٢٧     }
٢٨ #endif
٢٩ }
٣٠
٣١
٣٢ void _cdecl int_done(unsigned int n) {
٣٣     if (n > 16)
٣٤         return;
٣٥
٣٦     if (n > 7)
٣٧         /* send EOI to pic2 */
٣٨         i386_pic_send_command(I386_PIC_OCW2_MASK_EOI, 1);
٣٩
٤٠     /* also send to the primary pic */
٤١     i386_pic_send_command(I386_PIC_OCW2_MASK_EOI, 0);
٤٢ }
٤٣
٤٤ void _cdecl sound(unsigned int f) {
٤٥     outportb(0x61, 3 | unsigned char(f << 2));
٤٦ }
٤٧
٤٨ unsigned char _cdecl inportb(unsigned short port_num) {
٤٩ #ifdef _MSC_VER
٥٠     _asm {
٥١         mov dx, word ptr [port_num]
٥٢         in al, dx
٥٣         mov byte ptr [port_num], al
٥٤     }
٥٥ #endif
٥٦
٥٧     return unsigned char(port_num);
٥٨ }
٥٩
٦٠
٦١ void _cdecl outportb(unsigned short port_num, unsigned char value) {
٦٢ #ifdef _MSC_VER
```



```

٦٣  _asm {
٦٤      mov al,byte ptr[value]
٦٥      mov dx,word ptr[port.num]
٦٦      out dx,al
٦٧  }
٦٨  #endif
٦٩  }
٧٠
٧١  void _cdecl enable_irq() {
٧٢  #ifdef MSC_VER
٧٣      _asm sti
٧٤  #endif
٧٥  }
٧٦
٧٧  void _cdecl disable_irq() {
٧٨  #ifdef MSC_VER
٧٩      _asm cli
٨٠  #endif
٨١  }
٨٢
٨٣  void _cdecl set_vector(unsigned int int_num,void (_cdecl far & vect)
        ()) {
٨٤      i386_idt_install_ir(int_num,I386_IDT_32BIT|I386_IDT_PRESENT /*
        10001110*/,0x8 /*code desc*/,vect);
٨٥  }
٨٦
٨٧  void (_cdecl far * _cdecl get_vector(unsigned int int_num))() {
٨٨      idt_desc* desc = i386_get_idt_ir(int_num);
٨٩
٩٠      if (desc == 0)
٩١          return 0;
٩٢
٩٣      uint32_t address = desc->base_low | (desc->base_high << 16);
٩٤
٩٥      I386_IRQ_HANDLER irq = (I386_IRQ_HANDLER) address;
٩٦      return irq;
٩٧  }
٩٨
٩٩  const char* _cdecl get_cpu_vendor() {
١٠٠      return i386_cpu_vendor();
١٠١  }

```

```

١٠٢
١٠٣ int _cdecl get_tick_count() {
١٠٤     return i386_pit_get_tick_count();
١٠٥ }

```

Example \.١٧: kernel/main.cpp

```

١ int _cdecl main()
٢ {
٣     hal_init();
٤     enable_irq();
٥
٦     set_vector(0, (void (_cdecl &)(void)) divide_by_zero_fault);
٧     set_vector(1, (void (_cdecl &)(void)) single_step_trap);
٨     set_vector(2, (void (_cdecl &)(void)) nmi_trap);
٩     set_vector(3, (void (_cdecl &)(void)) breakpoint_trap);
١٠    set_vector(4, (void (_cdecl &)(void)) overflow_trap);
١١    set_vector(5, (void (_cdecl &)(void)) bounds_check_fault);
١٢    set_vector(6, (void (_cdecl &)(void)) invalid_opcode_fault);
١٣    set_vector(7, (void (_cdecl &)(void)) no_device_fault);
١٤    set_vector(8, (void (_cdecl &)(void)) double_fault_abort);
١٥    set_vector(10, (void (_cdecl &)(void)) invalid_tss_fault);
١٦    set_vector(11, (void (_cdecl &)(void)) no_segment_fault);
١٧    set_vector(12, (void (_cdecl &)(void)) stack_fault);
١٨    set_vector(13, (void (_cdecl &)(void)) general_protection_fault);
١٩    set_vector(14, (void (_cdecl &)(void)) page_fault);
٢٠    set_vector(16, (void (_cdecl &)(void)) fpu_fault);
٢١    set_vector(17, (void (_cdecl &)(void)) alignment_check_fault);
٢٢    set_vector(18, (void (_cdecl &)(void)) machine_check_abort);
٢٣    set_vector(19, (void (_cdecl &)(void)) simd_fpu_fault);
٢٤
٢٥    ...

```

Example \.١٨: kernel/exception.h

```

١ /* Exception Handler */
٢
٣ /* Divide by zero */
٤ extern void _cdecl divide_by_zero_fault(uint32_t cs, uint32_t eip,
    uint32_t eflags);

```

```

٥
٦ /* Single step */
٧ extern void _cdecl single_step_trap(uint32_t cs,uint32_t eip,uint32_t
    eflags);
٨
٩ /* No Maskable interrupt trap */
١٠ extern void _cdecl nmi_trap(uint32_t cs,uint32_t eip,uint32_t eflags)
    ;
١١
١٢ /* Breakpoint hit */
١٣ extern void _cdecl breakpoint_trap(uint32_t cs,uint32_t eip,uint32_t
    eflags);
١٤
١٥ /* Overflow trap */
١٦ extern void _cdecl overflow_trap(uint32_t cs,uint32_t eip,uint32_t
    eflags);
١٧
١٨ /* Bounds check */
١٩ extern void _cdecl bounds_check_fault(uint32_t cs,uint32_t eip,
    uint32_t eflags);
٢٠
٢١ /* invalid opcode instruction */
٢٢ extern void _cdecl invalid_opcode_fault(uint32_t cs,uint32_t eip,
    uint32_t eflags);
٢٣
٢٤ /* Device not available */
٢٥ extern void _cdecl no_device_fault(uint32_t cs,uint32_t eip,uint32_t
    eflags);
٢٦
٢٧ /* Double Fault */
٢٨ extern void _cdecl double_fault_abort(uint32_t cs,uint32_t err,
    uint32_t eip,uint32_t eflags);
٢٩
٣٠ /* Invalid TSS */
٣١ extern void _cdecl invalid_tss_fault(uint32_t cs,uint32_t err,
    uint32_t eip,uint32_t eflags);
٣٢
٣٣ /* Segment not present */
٣٤ extern void _cdecl no_segment_fault(uint32_t cs,uint32_t err,uint32_t
    eip,uint32_t eflags);
٣٥

```

```
٣٦ /* Stack fault */
٣٧ extern void _cdecl stack_fault(uint32_t cs,uint32_t err,uint32_t eip,
    uint32_t eflags);
٣٨
٣٩ /* General Protection Fault */
٤٠ extern void _cdecl general_protection_fault(uint32_t cs,uint32_t err,
    uint32_t eip,uint32_t eflags);
٤١
٤٢ /* Page Fault */
٤٣ extern void _cdecl page_fault(uint32_t cs,uint32_t err,uint32_t eip,
    uint32_t eflags);
٤٤
٤٥ /* FPU error */
٤٦ extern void _cdecl fpu_fault(uint32_t cs,uint32_t eip,uint32_t eflags
    );
٤٧
٤٨ /* Alignment Check */
٤٩ extern void _cdecl alignment_check_fault(uint32_t cs,uint32_t err,
    uint32_t eip,uint32_t eflags);
٥٠
٥١ /* Machine Check */
٥٢ extern void _cdecl machine_check_abort(uint32_t cs,uint32_t eip,
    uint32_t eflags);
٥٣
٥٤ /* FPU Single Instruction Multiple Data (SIMD) error */
٥٥ extern void _cdecl simd_fpu_fault(uint32_t cs,uint32_t eip,uint32_t
    eflags);
```

Example ١.١٩: kernel/exception.cpp

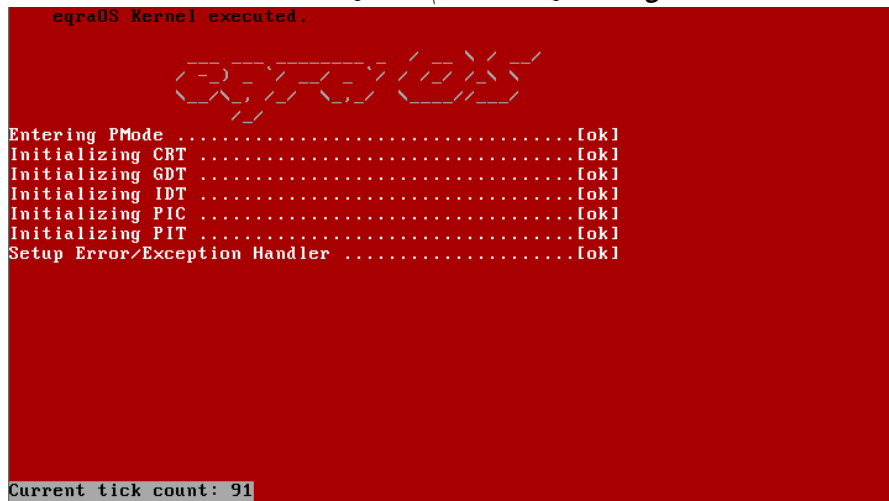
```
١ /* Divide by zero */
٢ void _cdecl divide_by_zero_fault(uint32_t cs,uint32_t eip,uint32_t
    eflags) {
٣     kernel_panic("Divide by 0");
٤     for (;;)
٥ }
```

Example ١.٢٠: kernel/panic.cpp

```
١ void _cdecl kernel_panic(const char* msg,...) {
```

```
٢
٣  disable_irq();
٤
٥  va_list args;
٦  va_start(args,msg);
٧  /* missing */
٨  va_end(args);
٩
١٠ char* panic = "\nSorry, eqraOS has encountered a problem and has
    been shutdown.\n\n";
١١
١٢  kclear(0x1f);
١٣  kgoto_xy(0,0);
١٤  kset_color(0x1f);
١٥  kputs(panic);
١٦  kprintf("*** STOP: %s",msg);
١٧
١٨  /* hang */
١٩  for (;;) ;
٢٠ }
```

شكل ٥.١: واجهة النظام بعد توسعة طبقة HAL



شكل ٦.١: دالة تستخدم المقاطعات الافتراضية

```
**** [I386 HAL] i386_default_handler: Unhandled Exception
```

