PIC, PIT et clavier

2017 – 2018

Florent Gluck – Florent.Gluck@hesge.ch

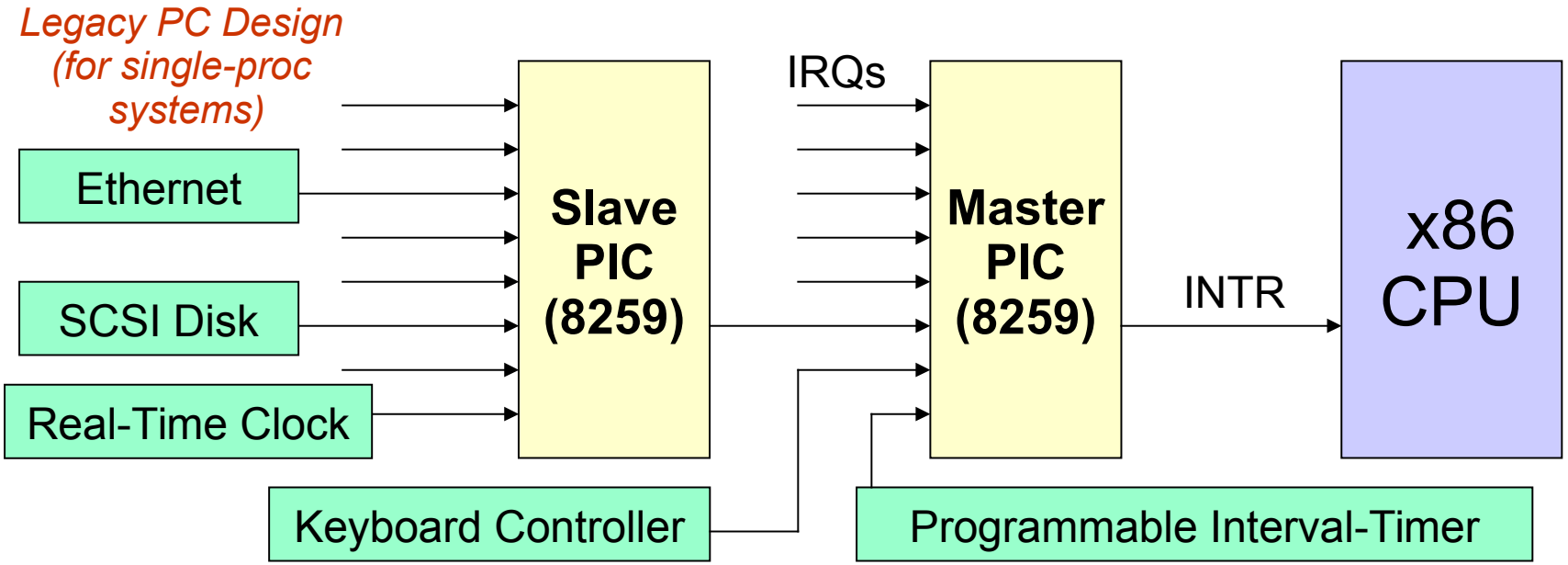
Version 0.2

h e p i a
Haute école du paysage, d'ingénierie et d'architecture de Genève

Hes·so // GENÈVE
Haute École Spécialisée de Suisse occidentale

Programmable Interrupt Controller (PIC)

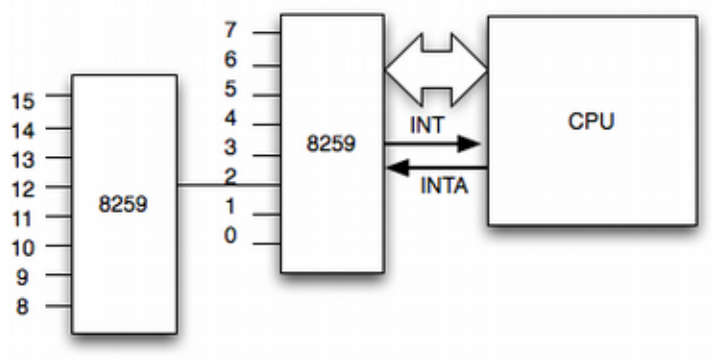
IRQ : matériel

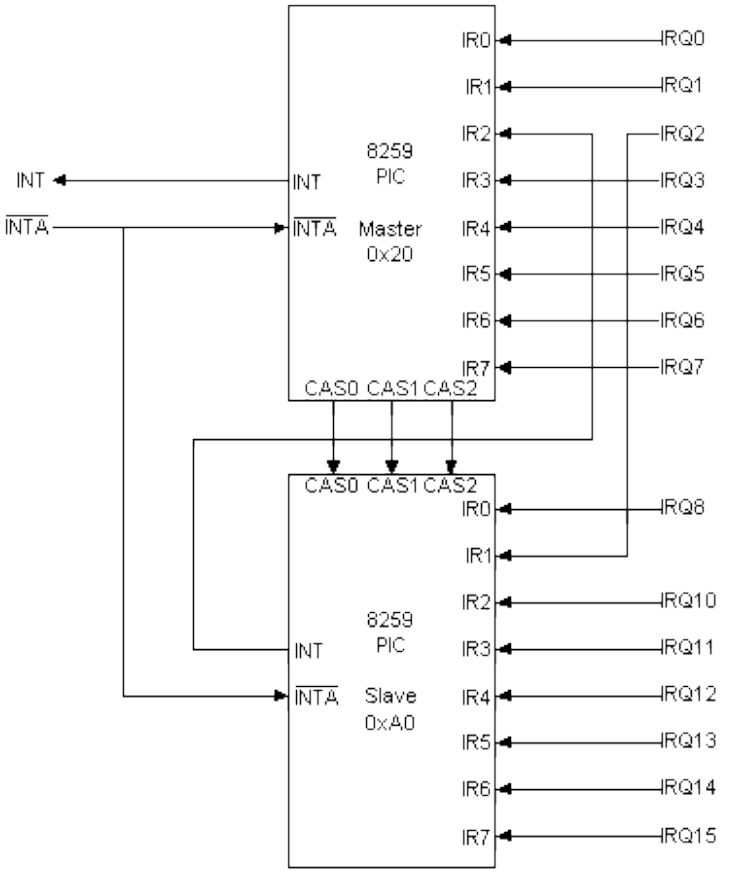


- Les périphériques possèdent des lignes générant les IRQ.
- Les IRQ sont mappées sur les vecteurs d'interruption par un chip dédié, le Programmable Interrupt Controller (PIC).

Contrôleurs d'interruptions

- Les interruptions matérielles sont gérées par deux contrôleurs appelés PIC (Programmable Interrupt Controller).
- Le CPU n'ayant qu'une ligne d'interruption, le 2ème PIC doit donc être connecté en cascade au 1er en une configuration maître/esclave (*master/slave*).
- Chaque PIC peut gérer huit IRQ maximum. Le PIC maître gère IRQ0 à IRQ7 et le PIC esclave IRQ8 à IRQ15.
- Chaque PIC possède un registre de commande, accessible par les ports :
  - Maître : `0x20`
  - Esclave : `0xA0`





End Of Interrupt

- Lorsqu'un périphérique signale une IRQ, une interruption matérielle est générée et le CPU appelle le gestionnaire d'interruption correspondant à l'IRQ signalée.
- Le CPU doit signaler au PIC qu'il a terminé le traitement de l'interruption en lui envoyant un signal End Of Interrupt (EOI), sinon l'interruption ne sera plus déclenchée.
  - Ceci est réalisé au début ou à la fin du code de l'ISR.
- Pour envoyer le signal EOI au PIC maître :
  - Ecrire la commande `0x20` au registre de commande du maître
- Pour envoyer le signal EOI au PIC esclave :
  - Ecrire la commande `0x20` au registre de commande du esclave

Remapping des interruptions matérielles (1)

- Lorsque le CPU a terminé le traitement d'une interruption (matérielle) liée au PIC maître, un EOI doit être envoyé à celui-ci.
- Lorsque le CPU a terminé le traitement d'une interruption liée au PIC esclave, un EOI doit être envoyé à l'esclave, **puis** au maître.
- La configuration par défaut du PIC mappe IRQ0 à 7 aux interruptions 0x8 à 0xF et IRQ8 à 15 aux interruptions 0x70 à 0x77.
- **Problème ?**

Remapping des interruptions matérielles (1)

- Lorsque le CPU a terminé le traitement d'une interruption (matérielle) liée au PIC maître, un EOI doit être envoyé à celui-ci.
- Lorsque le CPU a terminé le traitement d'une interruption liée au PIC esclave, un EOI doit être envoyé à l'esclave, **puis** au maître.
- La configuration par défaut du PIC mappe IRQ0 à 7 aux interruptions 0x8 à 0xF et IRQ8 à 15 aux interruptions 0x70 à 0x77.
- **Problème** : les 8 premières IRQ entrent en conflit avec les exceptions processeur (interruptions 0 à 31) !
- **Solution ?**

Remapping des interruptions matérielles (1)

- Lorsque le CPU a terminé le traitement d'une interruption (matérielle) liée au PIC maître, un EOI doit être envoyé à celui-ci.
- Lorsque le CPU a terminé le traitement d'une interruption liée au PIC esclave, un EOI doit être envoyé à l'esclave, **puis** au maître.
- La configuration par défaut du PIC mappe IRQ0 à 7 aux interruptions 0x8 à 0xF et IRQ8 à 15 aux interruptions 0x70 à 0x77.
- **Problème** : les 8 premières IRQ entrent en conflit avec les exceptions processeur (interruptions 0 à 31) !
- **Solution** : reprogrammer le PIC pour remapper IRQ0 à IRQ15 après l'interruption 31.

Remapping des interruptions matérielles (2)

Le code ci-dessous remappe IRQ0 à 7 du PIC maître aux interruptions 32 à 39 et IRQ8 à 15 du PIC esclave aux interruptions 40 à 47 :

```
#define PIC1_CMD        0x20
#define PIC1_DATA       0x21
#define PIC2_CMD        0xA0
#define PIC2_DATA       0xA1

// Restart both PICs
outb(PIC1_CMD, 0x11);
outb(PIC2_CMD, 0x11);

// Remap IRQ [0..7] to interrupts [32..39]
outb(PIC1_DATA, 32);

// Remap IRQ [8..15] to interrupts [40..47]
outb(PIC2_DATA, 40);

// Setup PIC cascading
outb(PIC1_DATA, 0x04);
outb(PIC2_DATA, 0x02);
outb(PIC1_DATA, 0x01);
outb(PIC2_DATA, 0x01);
```

Programmable Interval Timer (PIT)

Introduction

- Dans chaque PC se trouve un chip (Intel 8253) indispensable pour mesurer le temps et implémenter des timers : le PIT (Programmable Interval Timer).
- Le PIT génère une interruption matérielle à une fréquence sélectionnable entre 18.2065 Hz et 1.19318 MHz, via la ligne IRQ0.
- Au boot, le BIOS programme le PIT pour qu'il génère une IRQ0 à 18.2065 Hz.
- Le PIT possède une horloge interne oscillant à 1.19318 Mhz. Le signal de cette horloge est programmé via un diviseur de fréquence, ce qui permet de moduler la fréquence de sortie.

Configuration

- Pour fixer la fréquence du PIT, il suffit de lui envoyer le diviseur souhaité, càd la valeur pour laquelle sa fréquence d'entrée de 1.19318 Mhz doit être divisée.
- Le diviseur est une valeur de 16 bits comprise entre 1 et 65535
  - Un diviseur de 1 correspond à une fréquence de 1'193'180 Hz
  - Un diviseur de 65535 correspond à une fréquence de 18.2 Hz
- Le PIT possède 3 canaux, chacun avec un diviseur propre :
  - Canal 0 : le canal utilisé ici ; sa sortie est connectée à l'IRQ0.
  - Canal 1 : absent des PC modernes, mais utilisé historiquement pour rafraîchir la DRAM.
  - Canal 2 : contrôle le haut parleur interne du PC (avant la démocratisation des cartes son).

Programmation du PIT

- Le PIT se programme à travers les ports suivants :
  - Port 0x43 : registre de commande du PIT.
  - Port 0x40 à 0x42 : canaux 0 à 2 du PIT.

- Mettre à jour le diviseur du PIT se fait de la manière suivante :
  - Ecrire 0x36 dans le registre de commande du PIT indique la sélection du diviseur et le mode “répetition” (réinitialisation du compteur une fois celui-ci arrivé à 0).
  - Ecrire dans le canal 0 les 8 bits de poids faible (LSB) du diviseur.
  - Ecrire dans le canal 0 les 8 bits de poids fort (MSB) du diviseur.

Programmation du clavier

Clavier et scan codes

- A chaque pression ou relâchement d'une touche du clavier, le micro-contrôleur du clavier déclenche une interruption matérielle sur la ligne IRQ1.
- A chaque pression d'une touche, le **make code** de la touche pressée est stocké dans le buffer interne du clavier.
- A chaque relâchement d'une touche, le **break code** de la touche relâchée est stocké dans le buffer interne du clavier.
- Make codes et break codes forment ce qu'on appelle les **scan codes** du clavier ; chaque code est stocké sur 8 bits.
- **Attention** : les scan codes d'une touche n'ont rien à voir avec le code ASCII de celle-ci !

Registres du clavier

- Le micro-contrôleur du clavier est relativement complexe ; nous n'en voyons ici qu'un sous-ensemble volontairement très simple.
- Le clavier comporte un registre de données et un registre d'état :
  - Le registre de données est accessible au port 0x60
  - Le registre d'état est accessible au port 0x64

Lecture du clavier

- Lors de la pression ou le relâchement d'une touche, le scan code de la touche pressée ou relâchée peut être lu (8 bits) depuis le registre de données.

- A noter qu'avant de pouvoir lire une donnée en provenance du clavier, il est nécessaire de s'assurer que le buffer de sortie du clavier est plein (cela devrait toujours être le cas lorsqu'une IRQ est déclenchée) :
  - Une donnée est prête à être lue si le bit 0 du registre d'état est à 1.

Make code et break code

- Une fois le scan code lu depuis le registre de données, le bit 7 permet de déterminer si la touche a été pressée our relâchée :
  - Si le bit 7 est à 0 → pression de touche ; cette valeur est appelée le **make code**.
  - Si le bit 7 du scan code est à 1 → relâchement d'une touche ; cette valeur est appelée le **break code**.