Mini-projet système : développement d'un noyau de système d'exploitation

Responsable : Christophe Rippert
Christophe.Rippert@Grenoble-INP.fr

Gestion de l'horloge calendaire

Introduction

L'horloge calendaire (RTC pour *Real-Time Clock*) est un circuit capable de générer des interruptions à intervalle périodique, comme l'horloge matérielle gérée précédemment. C'est ce circuit qui a également la charge de conserver et mettre à jour la date et l'heure courante dans la machine. A la fin, le système devra afficher en haut à gauche de l'écran une chaîne contenant la date et l'heure courante, mise à jour automatiquement chaque seconde.
Attention : vous ne devez pas désactiver la gestion du temps (PIT) pour réaliser cette partie. En effet, il existe une notion de priorité entre les IRQ, et masquer l'IRQ0 aurait pour conséquence de masquer également l'IRQ8.

Lecture de la date et de l'heure

Les informations concernant la date et l'heure sont stockées dans une mémoire de très petite taille couramment appelée le CMOS (nom historique provenant de la technologie utilisée initialement pour préserver les données) et qui est alimentée par une batterie autonome fonctionnant même lorsque la machine est arrêtée. De nos jours, on parle de façon plus précise de la mémoire non-volatile du BIOS.
Le CMOS peut être accédé grâce aux ports 0x70 (commande) et 0x71 (données). Les informations (heure, jour, mois, etc.) sont stockés dans des registres internes au CMOS qu'on sélectionne en envoyant le numéro du registre via le port de commande.
Le schéma général pour récupérer les données est donc :

1. envoyer un octet calculé par `0x80 | num_reg` sur le port de commande ;
2. lire un octet résultat sur le port de données.

Les numéros de registres intéressants sont :
- 0 : secondes
- 2 : minutes
- 4 : heures
- 6 : numéro du jour dans la semaine
- 7 : numéro du jour dans le mois
- 8 : mois
- 9 : année
- 0x32 : centenaire

Les données sont en pratique stockées dans un format appelé Décimal Codé en Binaire Compact (*packed BCD* en anglais). Le principe de ce format est assez simple : chaque chiffre (0-9) de la valeur est codée sur un demi-octet (4 bits, parfois appelé *nibble* en anglais). Par exemple, si la valeur est 39, elle sera composée de deux demi-octets valant respectivement 0011 (pour le 3) et 1001 (pour le 9) en binaire, ce qui veut donc dire que 39 sera représenté par l'octet 00111001 (alors qu'en utilisant un codage binaire classique, 39 s'écrirait 00100111).
En pratique, toutes les valeurs à lire depuis le CMOS tiennent sur un octet : vous devez donc simplement décoder l'octet lu pour le convertir en décimal (ce qui se fait très facilement quand on se rappelle que $39 = 3 \times 10 + 9$ par exemple). Vous aurez besoin de la multiplication qu'on a déjà utilisé en assembleur, et toutes les autres opérations peuvent être réalisées grâce à des masques et décalages.
Le jour de la semaine renvoyé est un entier compris entre 1 et 7 inclus : 1 représente le dimanche, 2 le lundi, et ainsi de suite. De même, le numéro du mois est un entier compris entre 1 et 12 inclus : 1 représente janvier, etc.

La lecture du CMOS est particulièrement lente : il est en théorie possible qu'après avoir lu les minutes par exemple, elles changent avant qu'on ait eu le temps de lire le reste des informations. Par exemple, s'il est actuellement 13h59, on va lire le 59, et le temps de lire le 13, il sera en fait 14h00 : on lira donc 14 et l'heure affichée finalement sera 14h59. On ignorera ce problème qui en pratique se manifestera peu fréquemment sur un émulateur comme QEmu.
Note : le CMOS est une technologie préhistorique datant des tous premiers PC, qui a subit de nombreuses extensions depuis cette époque. Les informations données sur cette page devraient fonctionner sur QEmu, mais il est loin d'être garanti qu'elles fonctionnent nativement sur des machines récentes, ou même d'un modèle de machine à un autre.

Utilisation de l'interruption RTC

En plus de mémoriser la date et l'heure courantes, l'horloge calendaire est capable de générer des interruptions, de façon assez similaire au PIT vu précédemment.

Réglage de la fréquence

Pour régler la fréquence d'émission des signaux d'interruption en provenance de la RTC, on doit utiliser le port de commande 0x70 et le port de données 0x71 comme pour lire les valeurs du CMOS.
La RTC comprend un quartz capable d'émettre un signal à la fréquence maximale de 32 KHz, ce qui est trop rapide pour notre besoin : on fixera la fréquence à 2 Hz. La RTC est moins flexible que le PIT en ce qui concerne les fréquences supportées : on ne peut en effet fixer que des fréquences obtenues en divisant 32 KHz par une puissance de 2. La valeur à passer à la RTC est donc un diviseur de la fréquence maximale de 32 KHz.
Si on appelle cette valeur d, on peut la définir par : $f = \frac{2^{15}}{2^{(d-1)}}$ où f est la fréquence finale souhaitée, en notant que d est codée sur 4 bits. On peut ainsi déduire très facilement la bonne valeur du diviseur en notant que 2 Hz est la fréquence minimale supporté par le quartz de la RTC.
Pour fixer la fréquence d'émission des signaux, on doit donc :

1. envoyer l'octet de commande 0x8A sur le port de commandes de la RTC ;
2. lire sur le port de données un octet qui correspond à la valeur actuelle (qu'on nommera v) stockée dans le CMOS ;
3. envoyer de nouveau l'octet 0x8A sur le port de commandes ;
4. envoyer enfin sur le port de données un octet dont les 4 bits de poids faibles correspondent à la valeur du diviseur de fréquence d calculé, et dont les 4 bits de poids forts sont ceux de la valeur v lue précédemment.

Choix du type de signal

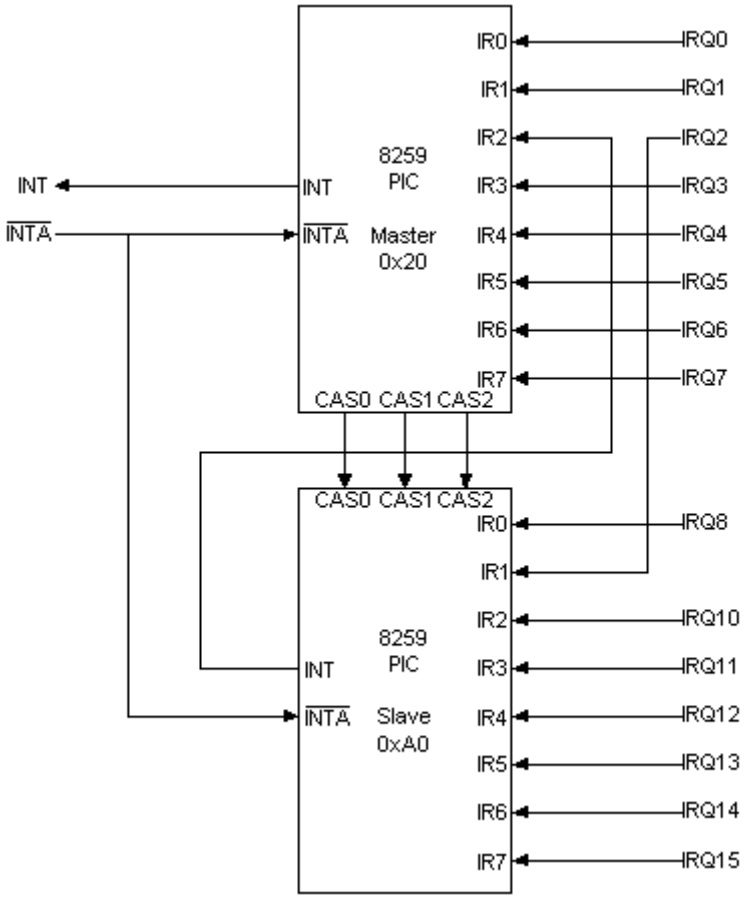
La RTC est capable d'émettre différents types de signaux : celui qui nous intéresse est un signal périodique émis à la fréquence réglée précédemment.
Pour régler le type de signal, on doit procéder de façon très similaire que pour le réglage de la fréquence :

1. envoyer l'octet de commande 0x8B sur le port de commandes de la RTC ;
2. lire sur le port de données un octet qui correspond à la valeur actuelle (qu'on nommera v) stockée dans le CMOS ;
3. envoyer de nouveau l'octet 0x8B sur le port de commandes ;
4. envoyer enfin sur le port de données la valeur calculée en forçant le 7e bit de v à 1 (le 7e bit est le bit numéro 6 puisqu'ils sont numérotés à partir de 0).

Une fois la RTC réglée, on doit procéder de façon très similaire au travail réalisé sur le PIT pour amener le signal émis jusqu'au processeur.

Réglage des IRQ

Le signal périodique émis par la RTC passe par l'IRQ 8 (alors qu'on rappelle que le PIT était connecté à l'IRQ 0). Les IRQ de numéro compris entre 0 et 7 sont gérées par le contrôleur d'interruption principal (ou « maître ») qu'on a déjà utilisé. Par contre les IRQ 8 à 15 sont gérées par un contrôleur secondaire (ou « esclave ») qui est connecté au premier via le canal d'IRQ 2, comme illustré dans le schéma ci-dessous :



En pratique cela implique simplement qu'en plus de démasquer l'IRQ 8 pour laisser passer les signaux en provenant de la RTC, on doit aussi démasquer l'IRQ 2.

On rappelle que pour démasquer l'IRQ numéro N on doit :

- lire un octet sur le port de données du contrôleur d'interruption gérant l'IRQ qu'on veut démasquer : cet octet est le masque actuel, qu'on appellera m ;
- forcer à 0 le bit de m correspondant à l'IRQ à démasquer : si N est < 8, il s'agit du bit N, si N est $>= 8$, il s'agit du bit N - 8 ;
- envoyer m sur le port de données du contrôleur gérant l'IRQ à démasquer.

Le port de données est le 0x21 pour le contrôleur d'interruption maître et le 0xA1 pour le contrôleur esclave.

Initialisation de la table des vecteurs d'interruption

L'arrivé d'un signal via l'IRQ 8 provoque l'émission de l'interruption 40 qui est transmise au processeur. On doit donc initialiser l'entrée numéro 40 de la table des vecteurs d'interruption avec l'adresse du traitant associé (notez bien que l'entrée numéro 40 est la 41e case de la table, vu que les indices commencent à 0).

Traitant de l'interruption 40

On doit finalement écrire le traitant de l'interruption 40 qui va réagir aux signaux en provenance de la RTC. On rappelle qu'un traitant d'interruption doit :

1. sauvegarder le contexte d'exécution courant (i.e. les registres modifiés par le traitant) ;
2. éventuellement effectuer certaines opérations supplémentaires nécessaires selon l'interruption (cela sera le cas ici) ;
3. appeler la fonction C réalisant le traitement voulu ;
4. restaurer le contexte d'exécution sauvegardé et rendre la main en utilisant `iret`.

Comme pour le PIT, le traitant d'une interruption sera en pratique composée :

— d'une partie en assembleur : c'est ce code qui sera appelé lorsqu'une interruption arrivera ;
— d'une partie en C : qui effectuera le travail à proprement parler, c'est à dire lire les informations du CMOS et les afficher à l'écran, c'est la fonction `affiche_date` décrite plus bas.

La partie en assembleur devra, en plus de sauvegarder et restaurer le contexte :

1. acquitter l'interruption 40 comme expliqué ci-dessous ;
2. réaliser une lecture sur le port de données de la RTC, pour lui signaler qu'on a bien reçu son signal : cela se fait très simplement en :
   (a) envoyant l'octet de commande 0x8C sur le port de commande de la RTC ;
   (b) lisant un octet sur le port de données de la RTC : la valeur lue n'a pas d'importance et ne sera pas utilisée, il faut juste réaliser une opération de lecture ;
3. appeler la fonction C.

L'acquittement de l'interruption 40 est un peu plus complexe que dans le cas du PIT. Il faut réaliser les deux opérations suivantes :

1. acquitter l'interruption auprès du contrôleur maître en envoyant l'octet de commande 0x20 sur le port 0x20 ;
2. acquitter aussi cette interruption auprès du contrôleur esclave, vu qu'elle provient d'une IRQ de numéro supérieur à 7 : cela se fait sur le même principe, mais en envoyant cette fois-ci l'octet de commande 0x20 sur le port 0xA0.

Résumé du travail demandé

On conseille de commencer par implanter les fonctions de lecture des informations du CMOS, c'est à dire :

1. une fonction `uint8_t lit_CMOS(uint8_t reg)` qui renvoie en décimal la valeur lue dans le registre de numéro reg : **cette fonction doit être traduite en assembleur** (vous implanterez le décodage BCD vers décimal dans cette fonction) ;
2. une fonction `void affiche_date(void)` qui lit les différentes valeurs intéressantes du CMOS, les mets en forme dans une chaine de caractères, et appelle une fonction `ecrit_date` pour réaliser l'affichage ;
3. une fonction `void ecrit_date(char *chaine)` à ajouter dans votre module de gestion de l'écran et qui écrit la chaine en haut à gauche de l'écran, de façon très similaire à la fonction `ecrit_temps` écrite précédemment pour gérer l'affichage de l'*uptime*.

Ensuite, il faudra gérer l'interruption 40, ce qui peut être décomposé en écrivant :

1. une fonction `void regle_frequence_RTC(void)` qui règle la fréquence et le type de signal de la RTC, **cette fonction doit être traduite en assembleur** ;
2. modifier ou étendre les fonctions que vous avez déjà écrites pour la gestion de l'interruption 32, pour qu'elles puissent aussi gérer l'interruption 40 ;
3. écrire la partie assembleur du traitant de l'interruption 40, comme expliqué plus haut (remarque : vous devez écrire un traitant séparé de celui de l'interruption 32, il n'est pas recommandé de chercher à écrire un traitant générique, mais vous pouvez fortement vous inspirer du traitant existant).

Si vous avez fini en avance

Vous pouvez ajouter la gestion du clavier à votre noyau. Pour cela, il suffit d'écrire un traitant pour l'interruption 33, qui doit :

— acquitter l'IRQ1 ;

— faire une lecture d'un octet sur le port 0x60.

Et bien sûr également démasquer l'IRQ1 et enregistrer l'adresse du traitant de l'interruption 33 dans la table des vecteurs d'interruption).

La valeur lue correspond au code de la touche (*scancode*) : décoder ce code pour obtenir le code ASCII de la touche est un processus plus compliqué car il dépend du modèle de clavier utilisé.