Exceptions, traps et interruptions: déroutements à l'initiative du matériel

1. Traps

Un **`trap`**, aussi appelé exception, est un genre d'appel automatique de procédure causé, en général, non pas directement par une instruction d'appel mais par un événement lié au programme et détecté par le matériel, comme, par exemple, un `overflow`. Lorsque le trap se produit, le compteur de programme est sauvé et on saute à une adresse dont la valeur est en général notée à une position fixe en mémoire. Cette position est appelée **vecteur de trap**. La procédure qui sera exécutée comme conséquence s'appelle **routine de trap**.

Il y a deux différences importantes entre trap et appel de procédure:

- la décision d'exécuter la routine de trap est prise par le matériel ou par l'interprète de niveau 1 mais pas par le programme de niveau 2;
- l'adresse de la routine de trap n'est pas dans le programme mais dans le vecteur de trap.

Quelques conditions classiques causant des traps sont overflow et underflow entier et virgule flottante, violation des protections de la mémoire ou du processeur, tentative d'accéder à une case de mémoire inexistante, opcode non défini, débordement de la pile, division par zéro, etc.

D'autre part, de nombreux ordinateurs disposent d'instructions qui demandent explicitement l'exécution d'une routine de trap ou cette exécution si une condition particulière est remplie. L'instruction de trap inconditionnelle sert surtout à implémenter le système d'exploitation. Son utilité apparaîtra dans le chapitre consacré à ce dernier. Les traps conditionnels permettent de ne mettre en service que lorsque le programme le demande le test d'une certaine condition par le microprogramme et l'exécution de la routine de trap correspondante si cette conditions est satisfaite.

Les séquences d'appel et de retour des routines de trap sont, en général, identiques à celles des interruptions et seront détaillées ci-dessous.

2. Interruptions

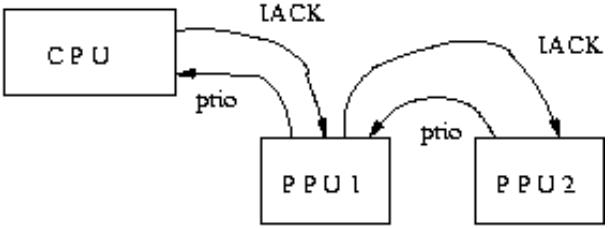
Les **interruptions** sont des événements fort semblables aux traps. Comme eux, ils donnent lieu à l'exécution d'une routine, la **routine d'interruption** dont l'adresse est souvent trouvée à une adresse connue en mémoire. Cependant, alors que les traps sont causés par de événements liés au programme (ils sont donc ***synchrones*** avec le programme), les interruptions sont causées par des événements extérieurs, liés aux processeurs périphériques ou PPUs (ils sont donc ***asynchrones*** avec le programme). Comme le programme en cours d'exécution n'est pas la cause de l'interruption et que celle-ci peut se produire n'importe quand pendant son exécution, la routine d'interruption ne pourra avoir aucun effet sur ce programme. Il faudra, en particulier, que les bits de condition (situés, en général dans le registre d'état du processeur) soient dans le même état qu'avant. Ils doivent donc être sauvés en début de routine d'interruption et restaurés à la fin, tout comme le compteur de programme.

Un ordinateur a souvent plusieurs PPUs actifs simultanément, en plus du CPU. Le CPU peut donc recevoir plusieurs demandes d'interruption, même pendant l'exécution d'autres routines d'interruption. Ceci entraîne plusieurs problèmes:

- quand on reçoit une demande d'interruption, il faut découvrir le PPU qui en est l'origine. Chaque type d'ordinateur utilisera une des techniques suivantes:
    1. Toutes les demandes d'interruption arrivent physiquement par la même ligne au CPU. Lorsqu'un signal arrive par cette ligne, la routine d'interruption dont l'adresse est indiquée dans le vecteur d'interruption unique est exécutée. Le CPU peut donc savoir qu'on lui demande d'exécuter une routine d'interruption mais il ne connaît pas l'origine de cette demande. C'est dans la routine d'interruption elle-même qu'il faudra identifier, par programme, le PPU qui a demandé l'interruption.
    2. Chaque PPU dispose de sa propre ligne d'interruption et un vecteur d'interruption différent est associé à chaque ligne. Ainsi c'est automatiquement la bonne routine d'interruption qui est exécutée.
    3. Toutes les demandes d'interruption arrivent physiquement par la même ligne au CPU. Lorsqu'un signal arrive par cette ligne, le microprogramme envoie un signal `IACK` (Interrupt Acknowledge, parfois appelé `INTA`) aux PPUs. Celui qui avait demandé l'interruption envoie un code identificateur via le bus de données (par exemple l'adresse du vecteur d'interuption). Grâce à ce code, le microprogramme lancera également automatiquement l'exécution de la bonne routine d'interruption. L'avantage de cette technique sur la précédente est de ne pas limiter le nombre de PPUs identifiables automatiquement au nombre de lignes d'interruptions disponibles.

       Ce système amène un nouveau problème: que se passe-t-il si deux PPU demandent en même temps une interruption? Le CPU devra exécuter la routine correspondant au PPU le plus prioritaire et pour cela, le signal `IACK` devra parvenir uniquement à celui-là. Pour résoudre ce problème, on utilise souvent une ***daisy chain***: le PPU le plus prioritaire est placé le plus près du CPU, le suivant, en ordre de priorités décroissantes, juste après, et ainsi de suite. Le signal `IACK` va du CPU au PPU le plus prioritaire, si il a demandé une interruption, il émet son code identificateur sur le bus de données, sinon, il réémet `IACK` vers le suivant, et ainsi de suite.
    4. Même système mais avec plusieurs lignes. Dans ce cas, les différentes lignes indiquent, en général, des priorités différentes et seule la plus prioritaire sera prise en compte.

       Parfois, un code représentant le niveau de priorité du PPU interrompant est émis sur ces lignes, ce qui permet plus de niveaux avec le même nombre de lignes. Dans ce cas, il faudra une *daisy chain* non seulement pour `IACK`, mais aussi pour le niveau de priorité: Ce niveau qui est le signal de demande d'interruption d'un PPU ne va pas au CPU mais au PPU immédiatement plus prioritaire que lui. Si celui-ci de doit pas demander d'interruption, il relaie le code qu'il a reçu, sinon il émet son propre code vers le PPU suivant dans l;a direction du CPU.



- Comme des interruptions peuvent se produire pendant l'exécution d'autres routines d'interruption, l'appel et le retour doivent non seulement être transparents (sauver et restaurer tout l'état du processeur, y compris tous les registres de niveau 1 et 2) mais aussi pouvoir être imbriqués. Ceci se réalise en général au moyen d'une pile.
- Ce n'est pas toujours une bonne idée de laisser n'importe quelle interruption interrompre la routine d'interruption de n'importe quelle autre. Par exemple, il semble peu souhaitable d'interrompre la routine de traitement des interruptions liées au disque pour servir une interruption causée par la frappe d'une touche au clavier d'un terminal. Il faut pouvoir spécifier à certains moments qu'on accepte d'être

interrompu et à d'autres pas.

3.1 Interruptions sur la 370

Quand un canal termine d'exécuter son programme, il peut interrompre le CPU de la manière suivante. Le CPU a un registre interne de 64 bits appelé **program status word (PSW)**. Celui-ci contient le compteur de programme, les codes de condition et d'autres informations. En cas d'interruption, le PSW est sauvé à l'adresse 56 et un nouveau PSW est chargé de l'adresse 120. Dès qu'il a chargé le nouveau PSW, le CPU commence à exécuter la routine générale de traitement des interruptions. Celle-ci devra d'abord identifier la source de la demande d'interruption en examinant l'état des différents canaux; elle peut alors exécuter la prodédure de traitement appropriée. La 370 utilise donc la technique "1." ci-dessus.

Si une nouvelle interruption se produisait pendant l'exécution de la première, le PSW serait à nouveau sauvé en 56, ce qui ferait perdre le précédent. Il faut donc, en début de traitement de l'interruption, sauver ce PSW, p.ex. sur une pile gérée par programme et interdire toute interruption tant que ce n'est pas fait. Dans ce but, la 370 a 7 bits de masque dans son PSW, chacun permettant d'interdire les interruptions en provenance d'un canal. Quand un bit de masque est à 0, on dit que le canal correspondant est désactivé. L'interruption ne passera que quand le bit de masque correspondant sera remis à 1.

3.2 Interruptions sur la PDP-11

Lorsque la PDP-11 est interrompue par un de ses processeurs périphériques, elle sauve sur la pile le **Processor Status Word (PSW**: un registre du processeur contenant les bits de condition et d'autres informations sur l'état de la machine), et le compteur de programme (PC). Ensuite, un nouveau PSW et un nouveau PC sont chargés d'une zone mémoire associée au périphérique. Ces zones sont appelées **vecteurs d'interruption** et, à chaque périphérique, en est associé au moins un. La nouvelle valeur pour le PC, contenue dans le vecteur d'interruption, indique où se trouve la routine de traitement appropriée au périphérique interrompant. *Cette situation est meilleure que celle de l'IBM qui n'a qu'un seul vecteur, donc une seule routine d'interruption qui doit donc d'abord déterminer la source de l'interruption; ici c'est automatique*.

La PDP-11 possède un mécanisme de priorité entre les interruptions. Chaque source d'interruption possède non seulement un vecteur qui lui est propre mais aussi une priorité. Cette priorité, tout comme le vecteur sont fixés dans le matériel. Lorsque le périphérique envoie une demande d'interruption au CPU, il lui fait savoir sa priorité et l'adresse de son vecteur d'interruption par la technique "4.." ci-dessus. Le processeur a aussi une priorité, fixée par programme et inscrite dans lePSW. Si la priorité de la source d'interruption est strictement supérieure à la priorité inscrite à ce moment dans le PSW, l'interruption est acceptée, sinon elle est masquée. Comme le vecteur d'interruption contient un nouveau PC et un nouveau PSW, *la routine d'interruption pourra s'exécuter en donnant n'importe quelle priorité au processeur*, même une valeur différente et de celle de la source d'interruption et du programme interrompu. On peut ainsi interdire d'interrompre certaines routines d'interruption ou certains morceaux critiques de programme.

La PDP-11 avait ainsi 8 niveaux de priorité différents codés sur 3 lignes.

Une particularité des 68K est que le PPU, au lieu de fournir le code identifiant le vecteur d'interruption peut émettre un signal sur une ligne particulière pour indiquer qu'il en est incapable. Le 68K choisit alors un vecteur par défaut associé au niveau de priorité.

3.3 Interruptions sur 8086/8088

Les 8086 et 8088 ont deux lignes d'interruptions, `IRQ` et `NMI`. La seconde ne peut jamais être masquée et elle ne sert que pour détecter les coupures d'alimentation: il faut alors coûte que coûte sauver l'essentiel (p. ex. copier certaines données sur disque pour permettre au système de redémarrer quand le courant reviendra).

Les 8086/8088 sont conçus pour utiliser la ligne `IRQ` selon la technique "3." ci-dessus. Celle-ci ne permet pas de distinguer les demandes d'interruptions sans les accepter et ne permet donc pas de refuser les demandes moins importantes à certains moments. En outre, les contrôleurs de périphériques d'Intel n'étaient pas prévus pour le "daisy chain", on ne pouvait donc utiliser de priorités positionnelles. Pour résoudre ces deux problèmes, les PPUs ne sont pas connectés directement à la ligne d'interruption du CPU, mais connectés, en étoile à un circuit 8259A. Ce circuit introduit une priorité entre les lignes et génère lui-même les codes identifiant le bon vecteur d'interruption pour le CPU. Le 8259A transforme donc la technique "3." en technique "2.". Le 8259A bloque les demandes d'interruption moins prioritaires jusqu'à ce que le CPU lui ordonne explicitement de les accepter à nouveau.

3.4 Interruptions sur PIC 16F87

Les PIC ont une ligne d'interruption externe mais comme ils ont de nombreux contrôleurs de périphériques internes, ce sont ces derniers qui sont les principales sources d'interruptions.

Toutes les sources d'interruptions (au nombre de 14) sont connectées au même signal d'interruption et un seul vecteur est disponible situé dans le quatrième mot de la mémoire de programme (un vecteur de reset est placé à l'adresse 0 de cette mémoire). Chaque source d'interruption a un bit "interrupt enable" et un bit "interrupt flag" individuels. Il existe aussi un bit (`GIE`) permettant de masquer toutes les interruptions. Si une demande d'interruption survient lorsque ce bit global et le bit spécifique de la source sont vrais, l'interruption est acceptée. Dans ce cas, le bit `GIE` devient faux et le compteur de programme est sauvé sur la pile interne. Aucun autre registre, pas même le registre d'état n'est sauvé: il faut le sauver (le copier dans un registre d'usage général) par software, puis examiner les flags de toutes les sources d'interruption afin d'identifier la source. Si on veut permettre des interruptions récursives, on peut alors remettre `GIE` à vrai (attention à ne pas dépasser la profondeur de la pile et à ne pas écraser la valeur sauvée du vecteur d'état: il faut compter le niveau d'interruption où on se trouve). Le plus simple est de ne pas autoriser les interruptions récursives. Dans ce cas, à la fin de la routine d'interruption il faut restaurer les registres sauvés. L'instruction de retour d'interruption remettra `GIE` à vrai et récupérera le compteur de programme sur la pile.

Conception et réalisation: Marc Lobelle