TD-TP 5 : interruptions et périphériques

Le but de cette séance est de comprendre le mécanisme d'interruption d'un processeur en réalisant des traitants d'interruption en langage d'assemblage.

Commencez par récupérer les sources pour cette séance sur Chamilo.

Les exercices sont organisés comme dans les TP précédents : un fichier `exo.c` qui contient le programme principal, un fichier `fct_exo.s` à remplir, et une règle de génération dans le Makefile (`make exo`) pour générer l'exécutable. Dans tous les exercices, il convient de vérifier l'exécution pas à pas du programme avec GDB et le simulateur.

Ex. 1 : Traitant d'interruption minimaliste : it

Le but de cet exercice est d'explorer le mécanisme d'interruption à partir d'un système très simple. Le code vous est donné, et permet de faire clignoter les LEDS à chaque appui sur un bouton poussoir. Il s'agit du programme `it`. Le traitant d'interruption est directement le code décrit dans `mon_traitant`

Concrètement, un clic sur les boutons poussoirs déclenche une interruption. Le processeur interrompt alors son travail en cours pour sauter à l'adresse du traitant d'interruption (`0x80000180` pour le MIPS).

Dans cet exercice, le processeur est ensuite redirigé vers le symbole `mon_traitant`.

Question 1 Testez l'exécutable `it` sur le simulateur QEMU en enlevant l'option `-nographic`.

Question 2 Ouvrez le fichier `fct_it.s`, et répondez aux questions suivantes : Quels sont les registres sauvegardés et restaurés lors d'une interruption dans le cas présent, et pourquoi ? Pourquoi la routine se termine par `eret` et non `jr $ra` comme habituellement pour une fonction ? A quoi sert `lw $t2,4($t1)`, que se passe-t-il si on l'enlève (tester) ? Comment écrit-t-on sur les LEDS ?

Question 3 A présent on va s'intéresser au mécanisme d'interruption. Placez vous en mode debug (en ajoutant les options -s et -S sur QEMU et en lançant en parallèle GDB). Testez l'exécutable `it` sur le simulateur. Placez un point d'arrêt à l'adresse `0x80000180`, et continuez l'exécution. Appuyez avec la souris sur un bouton poussoir de la plateforme dessinée dans le simulateur et exécutez en pas à pas le programme d'interruption et notez les étapes nécessaires à la gestion de l'interruption.

Question 4 Modifiez le programme présent dans `fct_it.s` pour qu'à chaque appui sur un bouton poussoir, `blink` soit incrémenté de 1 et non complémenté comme c'est le cas dans ce qui vous est fourni. Tester.

Question 5 Modifiez le programme présent dans `fct_it.s` pour qu'à chaque appui sur le bouton poussoir BTN0, `blink` soit incrémenté de 1 et l'appui sur le bouton poussoir BTN1, `blink` soit décrémenté. Tester.

Ex. 2 : Timer : timer

Dans cet exercice, nous allons exploiter plus de périphériques de la plateforme, et également utiliser un gestionnaire d'interruption beaucoup plus complet permettant de gérer plusieurs sources d'interruptions (`cep_exp.S`)

Le programme utilisera l'horloge intégrée dans le Coprocesseur 0 (`CP0`) du processeur MIPS pour afficher un compteur sur l'écran et/ou sur la sortie 7-segments de la plateforme.

Le fichier `timer.c` fourni le code permettant de gérer le fonctionnement de ce compteur.

Question 1 Dans le fichier `fct_timer.s`, écrivez la fonction `set_alarm`, qui remet à zéro le registre compteur (`Count`, `$9` de `CP0`) et fixe le registre de comparaison (`Compare`, `$11` de `CP0`) à la valeur founie en argument. Les informations utiles pour répondre à cette question sont fournies en annexes [1]. On utilisera pour ce faire l'instruction `mtc0`. [2]

Question 2 Compilez, puis exécutez votre code avec la commande :

```
qemu-system-mips -M mipscep -nographic -kernel timer
```

De l'observation du résultat de l'exécution sur la sortie standard et du fichier `timer.c`, que pouvez-vous dire de l'invocation de la fonction `timer_irq_hdl` ?

Question 3 Pour s'en assurer, relancez l'exécution en mode debogage (`-s -S`). Dans GDB, posez un point d'arrêt à l'adresse d'entrée des interruptions (`b *0x80000180` ou `b excp_hdl`), puis continuez (`c`) jusqu'au premier point d'arrêt. Observez et notez alors ce qu'il se passe jusqu'à la fin de ce traitant d'interruption (achevé par l'instruction `eret`). La commande GDB `list` peut vous aider à observer le code parcouru et la commande `si` vous permet d'avancer pas à pas. La documentation du MIPS doit vous aider à comprendre les instructions et leur utilisation (p.53 et 58 du volume III). Quelles sont les numéros des lignes d'interruptions associées au timer et au bouton poussoir (pushbtn) ?

Ex. 3 : Compteur commandé par les boutons poussoirs

L'exercice précédent vous a permis de voir que le traitant fourni appelle la fonction pushbtn_irq_hdl en cas d'interruption sur les boutons poussoirs.

Question 1 Dans le fichier `fct_timer.s`, implantez la fonction `pushbtn_irq_hdl` de sorte que l'appui sur un bouton provoque la remise à zéro du compteur (champ count de la structure pointée par param) et un rafraîchissement de l'affichage (appel à la fonction display). Puis, vérifiez le résultat dans le simulateur sans l'option `-nographic`.

Pour aller plus loin... **Question 2** Modifiez votre fonction de sorte que seul le premier bouton fasse la remise à zéro (voir memory map en annexe).

Pour aller plus loin... **Question 3** Modifiez-le encore pour que l'appui sur le deuxième bouton change le sens de comptage (utiliser le champ way de la structure définie dans le fichier C). La pseudo-instruction `la` (*load address*) pourra vous servir pour récupérer dans un registre l'adresse d'un symbole.

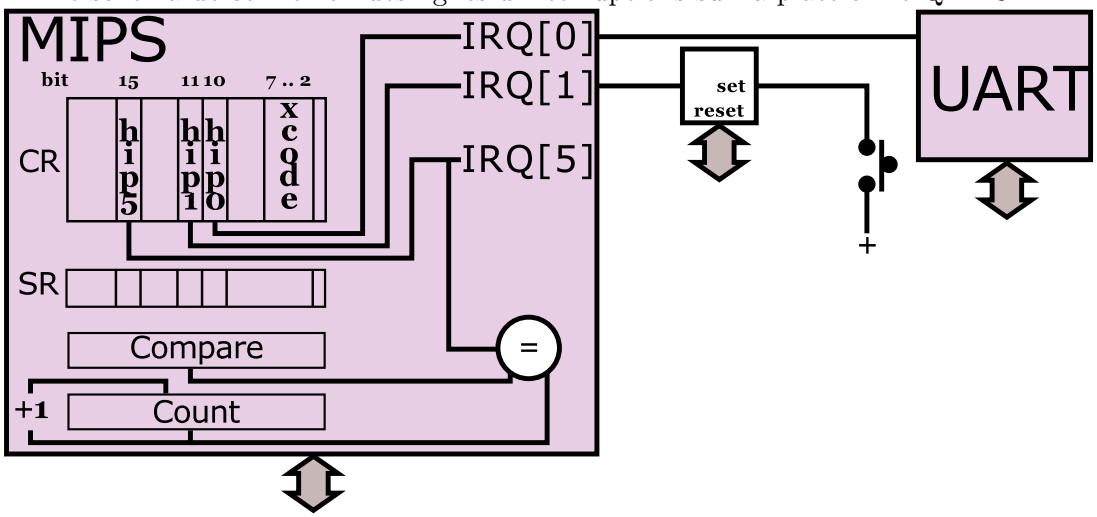
Pour aller plus loin... **Question 4** Pour finir, modifier le programme tel que le bouton 3 active ou non l'affichage VGA et que le bouton 4 active ou inactive l'affichage 7-segments. Regardez le champ mode de la structure fournie.

---

1. Elles sont directement extraites des documents "MIPS32 Architecture For Programmers Volume II" et du document "MIPS32 Architecture For Programmers Volume III", tous les deux disponibles sur l'Ensiwiki.
2. Par exemple si on souhaite mettre la valeur du registre `$t0` dans le registre 3 du coprocesseur 0, on fera `mtc0 $t0,$3`

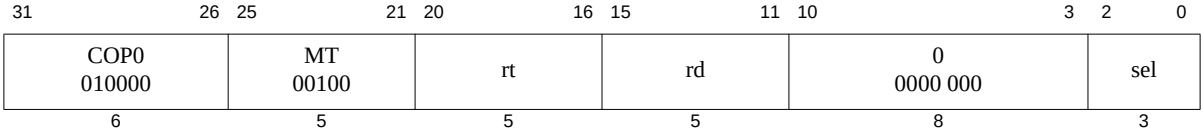
Annexes

Le schéma de connexion des lignes d'interruptions sur la plateforme QEMU :



Memory map de la plateforme QEMU

Périphériques	adresse	définition C	fonctionnement
LED	0x80050000	PERIPH_LED	Accès en écriture. Les 8 bits de poids faibles écrivent sur les LED
Boutons poussoirs et interrupteurs	0x80050004	PERIPH_SWITCHES	Accès en lecture. Interrupteurs sur les bits 0 à 7, boutons poussoirs sur les bits 8 à 11.
Contrôle du périphérique boutons poussoirs	0x80050008	PERIPH_PUSHBTN_CTL	Accès en lecture et écriture. 0 = mode scrutation. 1 = mode interruption
Afficheur 7 segments	0x8005000C	PERIPH_7SEG	Accès écriture. Selon le mode de fonctionnement, on affichera les bits 0-15 sur les 4 afficheurs en hexadécimales, les bits 16-31 en hexa, ou les 32 bits (1 bit par segments)
Contrôle du 7 segments	0x80050010	PERIPH_7SEG_CTL	Accès lecture/écriture, 0 = affichage 0-15, 1= affichage 16-31, 2= affichage brut



Format: `MTC0 rt, rd` **MIPS32**
`MTC0 rt, rd, sel` **MIPS32**

Purpose: Move to Coprocessor 0

To move the contents of a general register to a coprocessor 0 register.

Description: `CPR[0, rd, sel] ← GPR[rt]`

The contents of general register rt are loaded into the coprocessor 0 register specified by the combination of *rd* and *sel*. Not all coprocessor 0 registers support the the *sel* field. In those instances, the *sel* field must be set to zero.

Restrictions:

The results are **UNDEFINED** if coprocessor 0 does not contain a register as specified by *rd* and *sel*.

Operation:

```
data ← GPR[rt]
reg ← rd
CPR[0,reg,sel] ← data
```

Exceptions:

Coprocessor Unusable

Reserved Instruction

Coprocessor 0 Registers

The Coprocessor 0 (CP0) registers provide the interface between the ISA and the PRA. Each register is discussed below, with the registers presented in numerical order, first by register number, then by select field number.

6.1 Coprocessor 0 Register Summary

Table 6-1 lists the CP0 registers in numerical order. The individual registers are described later in this document. If the compliance level is qualified (e.g., "*Required* (TLB MMU)"), it applies only if the qualifying condition is true. The Sel column indicates the value to be used in the field of the same name in the MFC0 and MTC0 instructions.

Table 6-1 Coprocessor 0 Registers in Numerical Order

Register Number	Sel	Register Name	Function	Reference	Compliance Level
0	0	Index	Index into the TLB array	Section 6.3 on page 41	Required (TLB MMU); Optional (others)
1	0	Random	Randomly generated index into the TLB array	Section 6.4 on page 42	Required (TLB MMU); Optional (others)
2	0	EntryLo0	Low-order portion of the TLB entry for even-numbered virtual pages	Section 6.5 on page 43	Required (TLB MMU); Optional (others)
3	0	EntryLo1	Low-order portion of the TLB entry for odd-numbered virtual pages	Section 6.5 on page 43	Required (TLB MMU); Optional (others)
4	0	Context	Pointer to page table entry in memory	Section 6.6 on page 45	Required (TLB MMU); Optional (others)
5	0	PageMask	Control for variable page size in TLB entries	Section 6.7 on page 46	Required (TLB MMU); Optional (others)
6	0	Wired	Controls the number of fixed ("wired") TLB entries	Section 6.8 on page 47	Required (TLB MMU); Optional (others)
7	all		Reserved for future extensions		Reserved
8	0	BadVAddr	Reports the address for the most recent address-related exception	Section 6.9 on page 48	Required
9	0	Count	Processor cycle count	Section 6.10 on page 49	Required
9	6-7		Available for implementation dependent user	Section 6.11 on page 49	Implementation Dependent

Table 6-1 Coprocessor 0 Registers in Numerical Order

Register Number	Sel	Register Name	Function	Reference	Compliance Level
10	0	EntryHi	High-order portion of the TLB entry	Section 6.12 on page 51	Required (TLB MMU); Optional (others)
11	0	Compare	Timer interrupt control	Section 6.13 on page 52	Required
11	6-7		Available for implementation dependent user	Section 6.14 on page 52	Implementation Dependent
12	0	Status	Processor status and control	Section 6.15 on page 53	Required
13	0	Cause	Cause of last general exception	Section 6.16 on page 58	Required
14	0	EPC	Program counter at last exception	Section 6.17 on page 61	Required
15	0	PRId	Processor identification and revision	Section 6.18 on page 62	Required
16	0	Config	Configuration register	Section 6.19 on page 63	Required
16	1	Config1	Configuration register 1	Section 6.20 on page 65	Required
16	2	Config2	Configuration register 2	Section 6.21 on page 69	Optional
16	3	Config3	Configuration register 3	Section 6.22 on page 70	Optional
16	6-7		Available for implementation dependent user	Section 6.23 on page 71	Implementation Dependent
17	0	LLAddr	Load linked address	Section 6.24 on page 72	Optional
18	0-n	WatchLo	Watchpoint address	Section 6.25 on page 73	Optional
19	0-n	WatchHi	Watchpoint control	Section 6.26 on page 74	Optional
20	0		XContext in 64-bit implementations		Reserved
21	all		Reserved for future extensions		Reserved
22	all		Available for implementation dependent use	Section 6.27 on page 76	Implementation Dependent
23	0	Debug	EJTAG Debug register	EJTAG Specification	Optional
24	0	DEPC	Program counter at last EJTAG debug exception	EJTAG Specification	Optional
25	0-n	PerfCnt	Performance counter interface	Section 6.30 on page 79	Recommended
26	0	ErrCtl	Parity/ECC error control and status	Section 6.31 on page 82	Optional