Conception et exploitation des processeurs

Frédéric Pétrot

GRENOBLE INP UGA Ensimag

Équipe pédagogique :
Julie Dumas, Claire Maiza, Olivier Muller, Frédéric Pétrot, Lionel Rieg (resp.),
Manu Selva et Sebastien Viardot

Année universitaire 2021-2022

C1 Présentation du projet CEP et rappels de VHDL
C2 Chaîne de compilation et assembleur RISC-V
C3 Conventions pour les appels de fonctions
C4 Gestion des interruptions par le logiciel

Sommaire

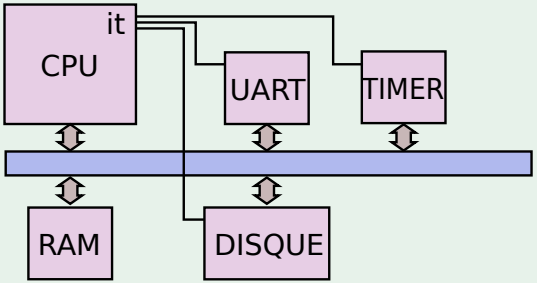
Introduction

Déroutement du cours de l'exécution

- trappes : appels systèmes, point d'arrêts du débugger, ...
- exceptions : dysfonctionnement au cours de l'exécution « normale »
- interruptions : événements externes, cible de ce cours !

Interruptions : événements *externes asynchrones* sur interface processeur

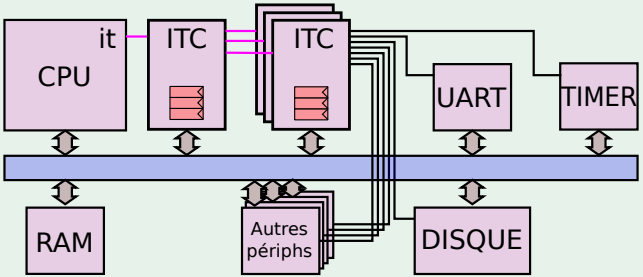
Émis par les périphériques



Introduction

Sources d'interruption

- peuvent être nombreuses : 256 prévues sur PC,
- entrées processeurs : de 1 à 6 ports d'interruption
- priorité logicielle ou matérielle



Structure « arborescente »

- ajout contrôleur(s) d'interruption (ITC, PIC, APIC, PLIC, ...)
- déterminer cause $\Rightarrow$ lecture successive registres ITCs par CPU

Introduction

Principe

Processeur :

- achève instruction en cours et arrête exécution programme courant
- détermine la cause de l'interruption
- agit en fonction de la cause pour la faire disparaitre
- reprend le programme après la dernière instruction exécutée

Sorte d'appel de fonction déclenché par un événement externe

Attention

- comportement du programme en cours d'exécution non altéré
- action processeur doit mener à arrêt demande d'interruption

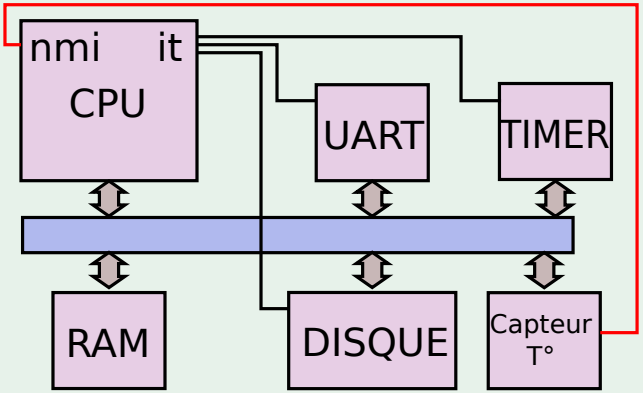
Corolaire

- processeur peut refuser d'être interrompu $\Rightarrow$ masquage
- sauf urgence absolue : `nmi`, *non-masquable interrupt*

Introduction

Non masquable

Par ex. destruction CPU si arrêt ventilateur



Quoi que fasse le CPU, il **faut** arrêter la machine !

Introduction

Pour résumer :

Aspect matériel

- connexion des lignes d'interruption
- présence ou non de contrôleurs
- registres pour configurer (masquage, priorités) ou connaître la cause
- appel du gestionnaire logiciel

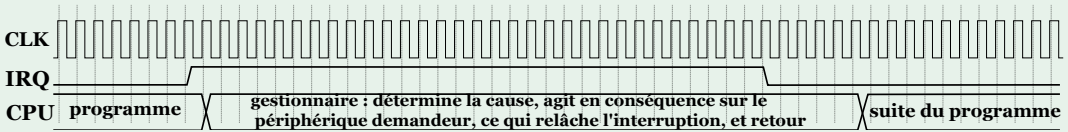
Aspect logiciel

- gestionnaire d'interruption (*interrupt handler, interrupt service routine - isr*)
- repose sur une connaissance exacte de l'aspect matériel
- ne doit pas changer l'état de la machine vu du programme interrompu

Sommaire

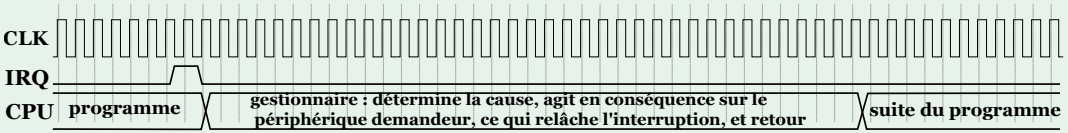
Types d'interruptions

Sur niveau



Périphérique maintient l'interruption, ne la relâche qu'une fois servi
On ne peut pas en perdre, même si masquée, Ex. : bus PCI

Sur événement



Périphérique fait un *pulse*
Doit redemander si masquée, Ex. : bus ISA

Processeur

Départ en interruption

Vecteur d'interruption

- table contenant des adresses : 6502, Cortex M0, ...
- adresse(s) prédéfinie(s) contenant des instructions :
  - une unique adresse « en dur » : MIPS
  - plusieurs adresses « en dur » dépendantes interruption : Microblaze, ARMv5, ...
  - plusieurs adresses dont la base est configurable : Sparc, ARMv6, ARMv7, ...
  - unique adresse ou plusieurs adresses avec base configurable : RISC-V, ...

Masquage global interruptions lors du déroutement

RISC-V : registre `mtvec`

Mise en place du gestionnaire de trappes

```
.text                       .text
boot_vector :               trap_vector :
   la t0, trap_vector           ...
   csrw mtvec, t0               ...
   ...
```

Instructions spécifiques pour registres système

`csrr`, `csrw`, ...

Processeur

Registres liés aux interruptions

Configuration

- masque global des interruptions
- masques « individuels » s'il y en a plusieurs
- priorités parfois

État

- indique la ou les entrées d'interruptions processeurs actives

Pointeur de programme sauvé

- contient l'adresse de l'instruction qui suit la dernière instruction exécutée avant le départ en interruption
- adresse à laquelle sauter à la fin du gestionnaire

RISC-V : registres liés à la gestion des interruptions

Configuration, *Machine Trap-Vector Base-Address Register*, `mtvec`, RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Base-Address																															

Pointeur de programme sauvé, *Machine Exception Program Counter* `mepc`, RW

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Exception PC																															

Configuration, *Machine Status Register*, `mstatus`, RW

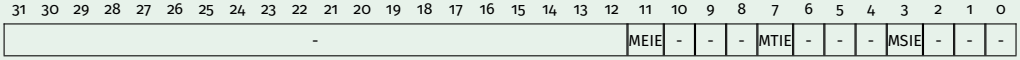
MIE : Machine Interrupt Enable, MPIE : Machine Previous Interrupt Enable

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																								MPIE	-	-	-	MIE	-	-	-

RISC-V : registres liés à la gestion des interruptions

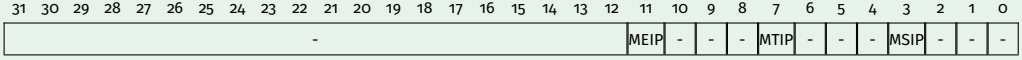
Configuration, *Machine Interrupt Enable Register*, `mie`, RW

MEIE : Machine External Interrupt Enable, MTIE : Machine Timer Interrupt Enable, MSIE : Machine Software Interrupt Enable



État, *Machine Interrupt Pending Register*, `mip`, RO

MEIP : Machine External Interrupt Pending, MTIP : Machine Timer Interrupt Pending, MSIP : Machine Software Interrupt Pending

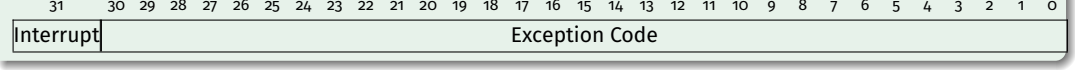


RISC-V : registres liés à la gestion des interruptions

État, *Machine Cause Register*, `mcause`, RO

Interrupt	Exception Code	
0	x	Exception, non traité
1	0-2	Réservé/non considéré
1	3	Machine software interrupt
1	4-6	Réservé/non considéré

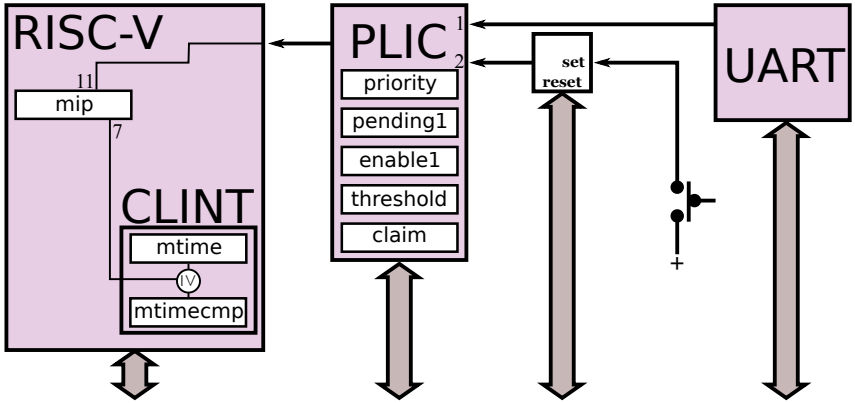
Interrupt	Exception Code	
1	7	Machine timer interrupt
1	8-10	Réservé/non considéré
1	11	Machine external interrupt
1	12–..	Réservé/non considéré



Reprise de l'exécution normale : instruction `mret`

```
mstatus[MIE] ← mstatus[MPIE]
pc ← mepc
```

RISC-V QEMU, machine cep



Organisation des interruptions

- bouton poussoir et UART sur EIP, lecture de `PLIC.claim` acquitte l'interruption
- timer directement sur TIP, positionnement de `mtimercmp` acquitte l'interruption

a. CLINT : Core Local INTerruptor, PLIC : PLatform Interrupt Controller.

Sommaire

Aspect logiciel

Étapes du traitement de l'interruption

- sauve dans pile registres pouvant être modifiés par gestionnaire :
  registres dont les valeurs ne sont pas préservés lors des appels de fonction
- appel d'une fonction (généralement écrite en C) déterminant la cause de l'interruption par analyse registre contenant l'état :
  - si plusieurs interruptions, choisir la plus prioritaire
  - si architecture avec contrôleurs, accès successifs pour identifier la source
  - appel fonction gérant cause interruption
- au retour de cette fonction, restaurer les registres sauvés au début
- retourner à l'instruction suivante dans le programme initial avec instruction spécifique

Aspect logiciel

Difficile d'être général et précis à la fois !

Dépend du processeur

- peut implicitement sauver une partie de ses registres, ou non
- peut empiler l'adresse de « retour », ou non
- peut avoir des priorités matérielles, ou logicielle
- ...

Dépend de l'architecture liés aux interruptions

- capacités du contrôleur, masquage, priorités, nombre de ports d'entrée et de sortie
- accès à ses registres internes
- hiérarchie si grand nombre de sources
- ...

Dépend des périphériques

- condition de déclenchement et de relâchement d'une IT
- ...

Cas du RISC-V : c.f. github.com/michaeljclark/riscv-probe

```
trap_vector:
    addi  sp, sp, -16*4
    sw    ra,  0*4(sp)
    sw    a0,  1*4(sp)
    sw    a1,  2*4(sp)
    sw    a2,  3*4(sp)
    sw    a3,  4*4(sp)
    sw    a4,  5*4(sp)
    sw    a5,  6*4(sp)
    sw    a6,  7*4(sp)
    sw    a7,  8*4(sp)
    sw    t0,  9*4(sp)
    sw    t1, 10*4(sp)
    sw    t2, 11*4(sp)
    sw    t3, 12*4(sp)
    sw    t4, 13*4(sp)
    sw    t5, 14*4(sp)
    sw    t6, 15*4(sp)

    csrr  a0, mcause
    csrr  a1, mie
    csrr  a2, mip
    jal   trap_handler

    lw    ra,  0*4(sp)
    lw    a0,  1*4(sp)
    lw    a1,  2*4(sp)
    lw    a2,  3*4(sp)
    lw    a3,  4*4(sp)
    lw    a4,  5*4(sp)
    lw    a5,  6*4(sp)
    lw    a6,  7*4(sp)
    lw    a7,  8*4(sp)
    lw    t0,  9*4(sp)
    lw    t1, 10*4(sp)
    lw    t2, 11*4(sp)
    lw    t3, 12*4(sp)
    lw    t4, 13*4(sp)
    lw    t5, 14*4(sp)
    lw    t6, 15*4(sp)
    addi sp, sp, 16*4
    mret
```

Sauvegarde et restauration des registres pouvant être modifiés

- registres sauvés sur la pile
- `sp` manipulé, mais non sauvegardé
- ne gère pas interruption plus prioritaire durant traitement interruption moins prioritaire

Appel de `trap_handler` avec :

- premier paramètre `mcause`, pour déterminer la cause de l'appel
- second paramètre et troisième paramètres `mie` et `mip`, pour tester si la source est encore active

RISC-V : gestionnaire d'interruption

```
void trap_handler(uint32_t mcause, uint32_t mie, uint32_t mip)
{
   if (mcause & 0x80000000) { /* Interruption */
      if ((mie & mip) == 0) /* relâchée ! */
         return;                    /* rien à faire */
      /* l'ordre défini les priorités */
      if (mcause & 0x008) {         /* gère l'interruption logicielle */
         software_irq_handler();
      } else if (mcause & 0x080) { /* gère l'interruption l'horloge */
         timer_irq_handler();
      } else if (mcause & 0x800) { /* gère l'interruption externe */
         external_irq_handler();
      }
      /* autres interruptions, non gérées */
      ...;
   else {
      /* Exceptions */
      /* Non gérées => mort du programme */
      ...;
   }
}
```

Sommaire

Pour résumer

Interruptions

Événements *externes et asynchrones* sur interface processeur
Émis par périphériques lorsqu'ils veulent l'attention du processeur

Gestion : activité matérielle et logicielle

- matériel :
  achève l'instruction en cours, sauve l'adresse de la suivante
  inhibe globalement toutes les interruptions, saute à un vecteur
- logiciel :
  sauve registres utilisés par gestionnaire
  détermine cause (éventuellement accès matériel externe)
  agit en fonction (pilote ou *driver* du périphérique)
  restaure registres
  appel instruction de retour d'interruption