Outline

- Introduction
- Architecture
- Implementation

Architecture

- Software visible registers
- Memory Addressing
- The instruction set
- The exception / reset mechanism

Exception / Reset

Exception / Interrupt / Reset mechanism :

- Reset mechanism
- Interrupt mechanism
- Exception mechanism

Exception / Reset

Reset

Re-initialize the system (the processor and all the other components of the system)

Abort the execution of the current program
All the data are lost
Re-initialize the software (including the OS)

Exception / Reset

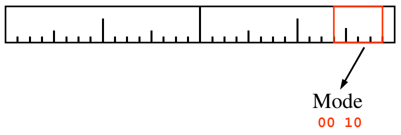
Reset

Abort the current program
Jump to the Reset Handler

- Initialize the address of the next instruction : `0xBFC0 0000`
- Initialize the Status Register :

Exception / Reset

- Status Register



`10 00` : user
`00 xx` : kernel
`xx 1x` : kernel
`xx x1` : kernel

Exception / Reset

Reset

Abort the current program
Jump to the Reset Handler

- Initialize the address of the next instruction : `0xBFC0 0000`
- Initialize the Status Register : `0x0000 0004`

Exception / Reset

Exception / Interrupt / Reset mechanism :

- Reset mechanism
- Interrupt mechanism
- Exception mechanism

Exception / Reset

Exception vs. Interrupt

Interrupts are events that require the processor to perform some operation

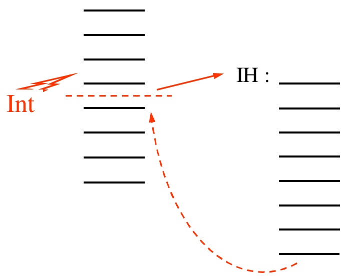
Interrupts are normal events during the life of a program

Exceptions are events that denote a malfunction in the program

Exceptions are abnormal events during the life of a program

Exception / Reset

Interrupt



Exception / Reset

Interrupt

Stop executing the current program
Execute the Interrupt Handler
Resume the interrupted program

- Initialize the address of the next instruction :
  `Ebase (31 downto 12) & X"180" : EXH-ADR`

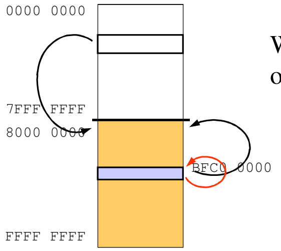
Exception / Reset

Reset

Abort the current program
Jump to the Reset Handler

- Initialize the address of the next instruction : `0xBFC0 0000`
- Initialize the Status Register : `0x0000 0004`
- Initialize the Exception Base Register : `0x8000 0000`

Exception / Reset



What happens if an interrupt occurs during the boot ?

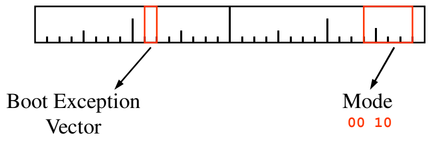
→ Go to **EXH-ADR**

If the Os is not yet loaded ?

→ Go to `0xBFC0 0380`

Exception / Reset

- Status Register



`10 00`	user
`00 xx`	kernel
`xx 1x`	kernel
`xx x1`	kernel

Exception / Reset

Reset Abort the current program
Jump to the Reset Handler

- Initialize the address of the next instruction : `0xBFC0 0000`
- Initialize the Status Register : `0x0040 0004`
- Initialize the Exception Base Register : `0x8000 0000`

Exception / Reset

Interrupt Stop executing the current program
Execute the Interrupt Handler
Resume the interrupted program

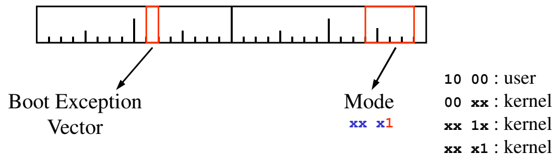
- Initialize the address of the next instruction :

if *BootExcVect* = 0	**EXH-ADR**
if *BootExcVect* = 1	`0xBFC0 0380`

- Set the SR : Mode (Kernel)

Exception / Reset

- Status Register



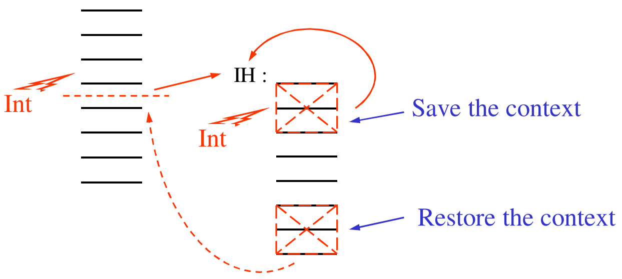
Boot Exception Vector

Mode

xx x1

`10 00` : user
`00 xx` : kernel
`xx 1x` : kernel
`xx x1` : kernel

Exception / Reset



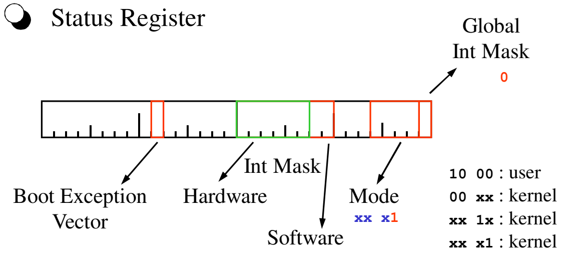
Save the context

Restore the context

Do not accept a new interrupt
until the context is saved

Exception / Reset

- Status Register



Global Int Mask

0

Boot Exception Vector

Int Mask

Hardware

Software

Mode

xx x1

`10 00` : user
`00 xx` : kernel
`xx 1x` : kernel
`xx x1` : kernel

Exception / Reset

Interrupt
Stop executing the current program
Execute the Interrupt Handler
Resume the interrupted program

- Initialize the address of the next instruction :

if *BootExcVect* = 0	**`EXH-ADR`**
if *BootExcVect* = 1	**`0xBFC0 0380`**

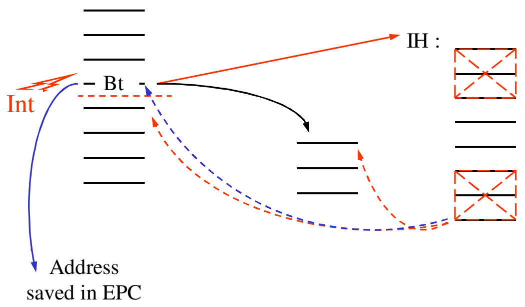
- Set the SR : Mode (Kernel) , set the Global Int Mask

Exception / Reset

Interrupt
Stop executing the current program
Execute the Interrupt Handler
Resume the interrupted program

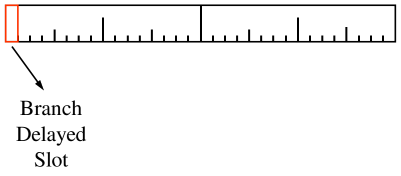
- Initialize the address of the next instruction :
  if *BootExcVect* = 0 `EXH-ADR`
  if *BootExcVect* = 1 `0xBFC0 0380`
- Save the return address in EPC
- Set the SR : Mode (Kernel) , set the Global Int Mask

Exception / Reset



Exception / Reset

- Cause Register



Exception / Reset

Interrupt
Stop executing the current program
Execute the Interrupt Handler
Resume the interrupted program

- Initialize the address of the next instruction :
  if *BootExcVect* = 0 `EXH-ADR`
  if *BootExcVect* = 1 `0xBFC0 0380`
- Save the return address in EPC
- Set the SR : Mode (Kernel) , set the Global Int Mask
- Set the Cause

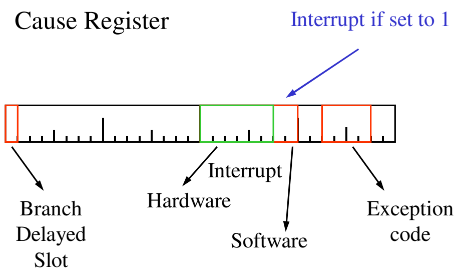
Exception

- ➢ Interrupt

Exception causes

Exception / Reset

- ❍ Cause Register



Exception / Reset

Reset

Abort the current program
Jump to the Reset Handler

- ❍ Initialize the address of the next instruction : `0xBFC0 0000`
- ❍ Initialize the Status Register : `0x0040 0004`
- ❍ Initialize the Cause Register : `0x0000 0000`
- ❍ Save the return address into Eepc
- ❍ Initialize the Exception Base Register : `0x8000 0000`

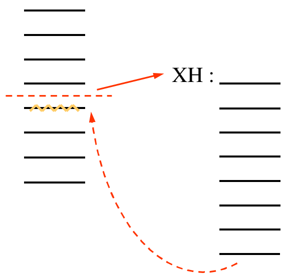
Exception / Reset

Exception / Interrupt / Reset mechanism :

- ➢ Reset mechanism
- ➢ Interrupt mechanism
- ➢ Exception mechanism

Exception / Reset

Exception



To preserve the integrity of the system, the faulty instruction should not be executed

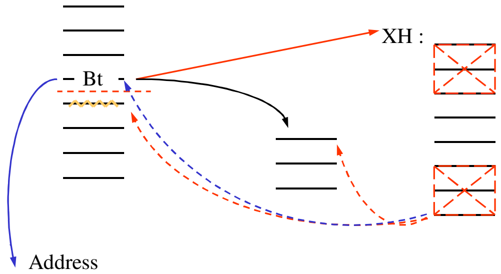
In most of the cases a faulty program is not resumed, but killed

Exception / Reset

Exception
Stop executing the current program
Execute the Exception Handler
Resume the interrupted program

- Initialize the address of the next instruction :
  `EXH-ADR` or `0xBFC0 0380`
- Save the faulty instruction’s address in EPC
- Set the SR : Mode (Kernel) , set the Global Int Mask
- Set the Cause
- Set the BadVAddr

Exception / Reset



Address saved in EPC

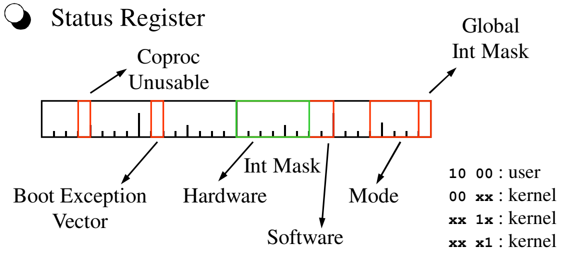
Exception

Exception causes

- Overflow
- Illegal read address
- Illegal write address
- Coprocessor unusable

Exception / Reset

- Status Register



`10 00` : user
`00 xx` : kernel
`xx 1x` : kernel
`xx x1` : kernel

Exception

Exception causes

- Overflow
- Illegal read address
- Illegal write address
- Coprocessor unusable
- Unknown instruction
- Syscall
- Break
- Trap
- Data bus error
- Instruction bus error
- Machine check

Exception / Reset

Exception vs. Interrupt

Interrupts are events that require the processor to perform some operation

Interrupts are asynchronous
⇨ no emergency

Exceptions are events that denote a malfunction in the program

An exception is an error
⇨ the faulty instruction should NOT be executed