PKaya and uMPS2

Legend:

Status related
Exceptions related
Virtual Memory related
Kernel related

Questions

- What is "branch delay" and why it has to be treated as an exception cause?

System Info

- Length of a word: 32bit
- Number of registers for each CPU: 32
- Max number of CPUs: 16 (0-15), only CPU[0] is activated at bootstrap
- Endianess: depends on the host system
- No floating point operations! A Trap is generated if you attempt a floating point op or try to access che floating point dedicated coprocessor (not implemented yet) CP1
- At the **bootstrap** CPU[0] and CP0 are enabled, Virtual Memory is off (Status.VM = 0), Status.BEV = 1, the CPU is in kernel mode (Status.KU = 1), the PC register equals *0x1FC0.0000* (the beginning of the boot ROM)

System Structure

CPU	**Registers (32)** **$0** : hard-wired to 0 (not usable) **$at** : reserved for the assembler (temp values) **$v0-$v1** : exp. Evaluations, static link **$a0-$a3** : first four int args in C function calls **$t0-$t9** : temp values for exp. Evaluations **$s0-$s7** : preserved values (between procedures) **$k0** : reserved to kernel use **$k1** : reserved to kernel use **$gp** : global pointer (see memory layout) **$sp** : stack pointer **$s8**\|**$fp** : a saved register or used as frame point. **$ra** : procedure call return address (or temp values)
CP0	- Switches from user to kernel mode and vv. - Exceptions handling - Virtual Memory Addressing **Registers (8)** Status Cause EPC Index Random EntryHi EntryLo

	BadVAddr
RAM	**Physical Memory Access** - Physical Frame Number + Offset **Virtual Memory Access** - Segment Number + Virtual Memory Address + Offset **Address Space Identifier (ASI)** A number between 0 and 63 (0 is reserved for the kernel).
ROM	
Devices	

CP0 Registers Structure

Status	The Status register contains fields regarding the status of the main CPU features. The VM, KU and IE registers are split into 3 "slots" each of which has an associated character (in {o,p,c}) that respectively mean oLD, pREVIOUS and cURRENT. They're used during the "pushing" and "pulling" operations (see below).	IE – Interrupt Enabler/Disabler KU – Kernel(0)/User(1) mode IM – Interrupt Mask VM – Virtual Memory Enabler/Disabler BEV – Bootstrap Exception Vector CU – Coprocessor Usable (Enabler/Disabler)
Cause	The Cause register contains che cause of the latest exception detected. The ExcCode contains the ID of the exception. Each ID is linked to a specific type of exception cause (Syscall, Breakpoint, etc.). The BD field indicates if the exception was caused by an instruction (=0) (the one pointed by EPC, previously by PC) or an instruction in the branch delay (=1).	IP – Interrupt Pending BD – Branch Delay CE – CoProcessor Error ExcCode – The ID of the exception (0..14)
EPC	Is the Exceptions Program Counter. Whenever an exception happens it is automatically reset, to execute the branch again, iif Cause.BD = 1.	

Index		
Random		
EntryHi		
EntryLo		
BadVAddr		

<u>Exceptions Types and Handling</u>
Whenever there's a double cell it means it has to be distinguished between *Load (or fetch instruction)* and *Store* instructions.

4 5	Address Error (AdEL & AdES)	
6 7	Bus Error (IBE & DBE)	
10	Reserved Instruction (RI)	
11	CoProcessor Unusable (CpU)	
12	Arithmetic Overflow (Ov)	
8	SYSCALL Instruction (Sys)	
9	BREAK Instruction (Bp)	
1	TLB-Modification (Mod)	
2 3	TLB-Invalid (TLBL & TLBS)	
13	Bad-PgTbl (BdPT)	
14	PTE-Miss (PTMs)	
0	External Device Interrupt (and Interrupts in general) (Int)	Influenced by Status.IM and Status.IE(c) bits.

<u>What happens when an Exception occurs?</u>
Notice: All the following operations MUST be executed atomically!
As soon as the exception happens some basic operations are executed:

1) The current PC must be copied into the EPC register.
2) If required the Cause.BD bit must be set (to correct the EPC).
3) Cause.ExcCode is set to the cause of the exception.
4) Whenever an exception occurs the CPU sets the VM(c), KU(c) and IE(c) to 0 as a "safe response". Before that the previous value (p) is stored into the old value (o) and the current value (c) is stored in (p), then it becomes 0. This happens for each of the three fields above and this action is called "pushing".

Then some specific operations (based on the type of the exception) are executed:

- AdEL/AdES: set BadVAddr (its value must be != 0)
- CpU: set Cause.CE
- Int: set Cause.IP
- TLBL/TLBS: set BadVAddr

At the end the PC register must be set to a fixed address (depending on the value of Status.BEV):

- if 1: PC = 0x1FC0.0180

- if 0: PC = 0x0000.0080

Finally the associated handler will be executed and, in case of a TLB exception, either BdPT or PTMs is set.

Now a handler is called to manage the exception rightfully! Its first task is to save the current processor state (see below) and load a new one. Just before it terminates the old state is restored and the VM, KU and IE, both (o) and (p), are **automatically** "pulled" (o->p, p->c).

Processor State Structure (Memory)

libumps provides a nice way of accessing every single CPU feature as a struct field. But how is it mapped into the Main Memory? It is stored at the bottom of the ROM Reserved Frame (first 4096B of the RAM beginning at 0x2000.0000) and it contains: 1 word for the EntryHi CP0 register (for the current ASID, EntryHi.ASID), 1 word for the cause CP0 register, 1 word for the PC (new) or EPC (old), 29 words for the general purpose registers ($0, $k0 and $k1 are excluded).
Notice: whenever a process in User mode (KU=1) tries to access addresses below 0x2000.0000 an Address Error Exception has to be thrown!