MIPS Exceptions

Dinesh Sharma

EE Department
IIT Bombay, Mumbai

April 19, 2021

What are exceptions?

Exceptions are conditions which cause unanticipated changes in program flow.

- Exceptions may be caused by external devices. (Also called Interrupts)
- Exceptions may result from errors during operations. For example, an attempt to divide by zero – or an overflow condition for signed operations.
- Exceptions may result due to violations of hardware assumptions. For example, an attempt to fetch an instruction form an address which is not divisible by 4. (Unaligned access).
- An unrecognized instruction can also cause an exception. Some times, this is used for extending the instruction set of a processor. An unrecognised instruction causes an exception, whose handler then implements the function associated with that instruction in software.

Exception Types

- Exceptions caused by external devices through a hardware signal on a pin (interrupts) do not indicate a problem with the running program. These cause the current instruction to be completed and the program counter to be advanced to the next address. Only after this, interrupt handling begins.
- The *updated* PC value is stored as the return address by the hardware managing the exception.
- In case of errors (like divide by zero), the instruction cannot be completed. Advancing of PC is part of the instruction execution. In this case, PC is stored as is, pointing to the instruction which caused the error.
- The handler should advance the PC value in this case, so that the instruction causing error is not executed on resumption.

Exception Handling

- Apart from external interrupts and internal errors which cause exceptions, software may cause an exception on purpose – for example through a trap instruction to single step through a program or through syscall to request a particular service from the operating system.
- Resumption from an exception must completely restore the processor state as if nothing had happened.
- MIPS can be operating in user mode or in kernel mode. Restoration must make sure that this state is correctly restored before control is handed over to the resumed program.
- In case of MIPS, exceptions are handled through a co-processor. Architecture of MIPS permits upto 4 co-processors which are, in general, optional. However this co-processor (co-processor 0) is always implemented.

Co-Processors in MIPS

- MIPS co-processors operate independently from the main processor.
- Each co-processor has its own registers. Data can be transferred between MIPS registers and co-processor registers through special instructions.
- These are the only R type instructions for which the top 6 bits are not all zero.

The instructions are similar to the ones used for moving data to and from Hi and Lo registers of the multiplier.

```
mfc0 k0,13 # CPU register $k0 is loaded with contents of
           # the “Cause” register : register 13 of coprocessor 0

mtc0 0,12 # CPU register $0 stored in the Status register
          # which is thus cleared to zero.
```

Co-Processor 0

- Co-Processor 0 is used for multiple purposes.
- It is used for implementing virtual memory as well as for exception handling.
- Of its 32 registers, 4 are used for exception handling.
- Other registers of interest are $9 and $11 which are count and compare registers of the internal timer.
- Wires from external devices provide input interrupt signals to coprocessor 0.
- The co-processor sends a signal to the CPU control unit when an exception occurs (internal or external).

Co-Processor 0 registers

Co-processor 0 maintains data related to cause and handling of exceptions.
Exception handling Registers of Co-Processor 0 include:

Reg.	Name	Function
8	BadVAddr	Memory addr. which caused the exception
12	Status	Interrupt mask, enable bits and status
13	Cause	Exception type and pending interrupt bits
14	EPC	Addr. of instr. which caused exception

Bits within the Status register can be manipulated at the assembly language level to selectively enable or disable certain interrupts.
Assemblers refer to Co-processor registers by number only and not by name, so descriptive comments are essential.

Co-Processor 0 registers

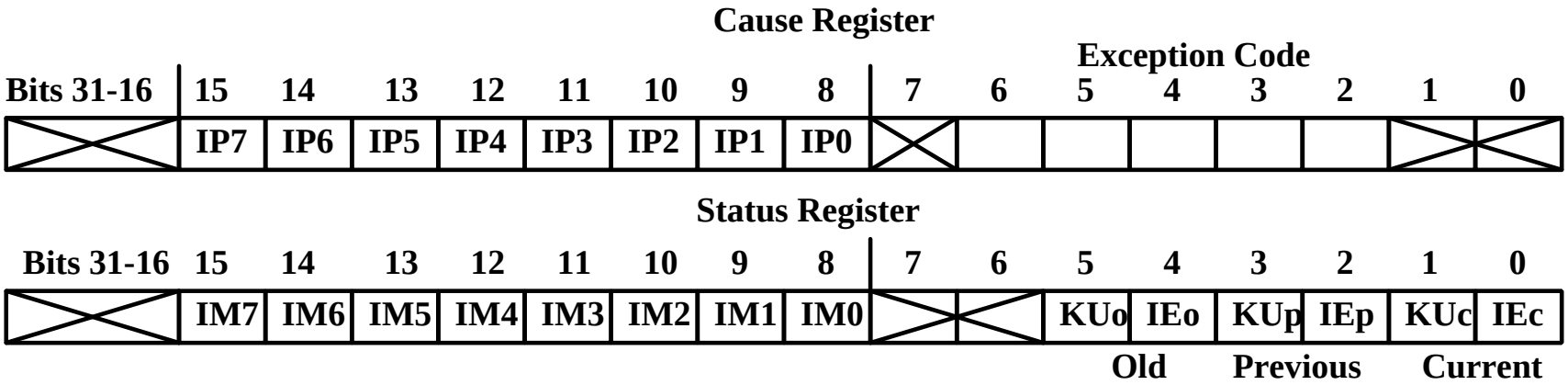
BadVAddr The name stands for Bad Virtual Address. It contains the memory address where the exception occurred. If access to an unaligned address causes an exception, the address will be stored in this register.

Cause Register It provides information about the cause of the interrupt and about pending interrupts.
The top 16 bits of this register are undefined.

The higer byte of the low half-word contains flags (IP7-IP0) to indicate if there are pending interrupts due to 8 possible levels of interrupt.

bits 6 to 2 contain a code which indicates the type of event which resulted in the interrupt.

Co-Processor 0 registers



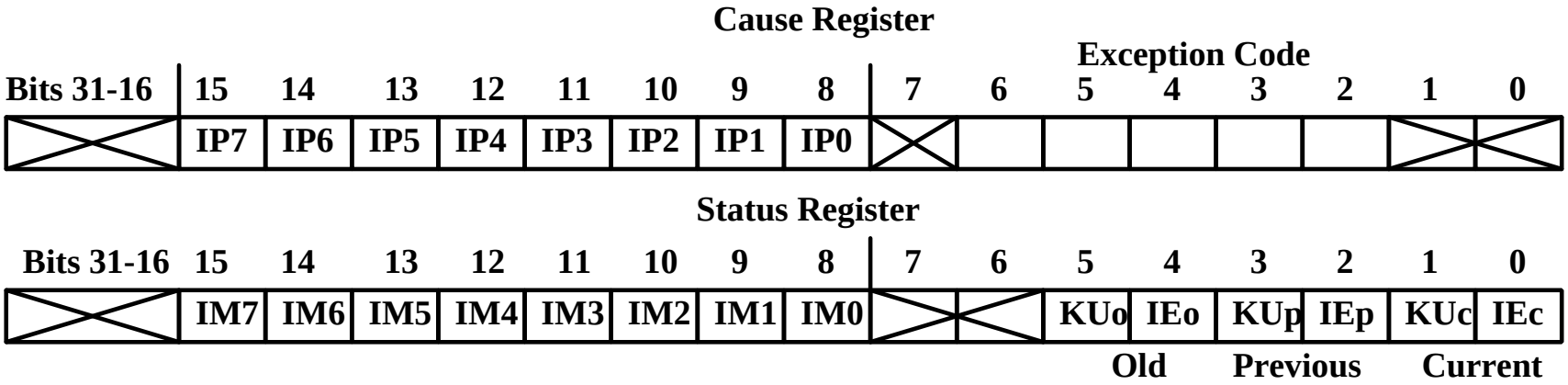
- IP bits in the cause register represents pending interrupts.
- Exception code in the cause register encodes what caused the exception.
- IM bits in the status register are Mask bits. A 0 in the mask bit prevents interrupts from that level.
- The K bits show if the processor is operating in Kernel Mode.
- IE bits show whether interrupts are globally enabled.

Exception Codes in the Cause Register

Code	Name	Description
0	INT	External Interrupt
1-3	Virt Mem	Reserved for Virtual Memory
4	ADDRL	Load from illegal address
5	ADDRS	Store to illegal address
6	IBUS	Bus error on Instr. fetch
7	DBus	Bus error on Data reference
8	SYSCALL	Syscall instruction executed
9	BKPT	Break instruction executed
10	RI	Reserved Instruction
11	CoProc	Missing Co-Processor
12	OVF	Arithmetic Overflow

Codes above 12 are used for floating point exceptions.

Co-Processor 0 registers



- KUc represents whether the processor is currently operating in the Kernel mode or the User mode. (0 = Kernel, 1 = User).
- IEc represents if currently interrupts are enabled.
- When an exception occurs, Previous values KUp and IEp are copied to old values KEo and IEo. Current values KUc and IEc are copied to previous values KEp and IEp.
- On return from exception handler, KUp and IEp are restored to KUc and IEc, KUo and IEp are copied to KUp and IEp, while the values of KUo and IEo remain unchanged.

Co-Processor 0 registers

The EPC register (register $14 of co-processor)

- Register $14 of the co-processor (The EPC register) is used for enabling return to the running program after exception handling.
- $ra is not used because that would clobber the return address of a function when an interrupt occurs.
- In case of an interrupt, EPC contains the updated PC.
- In case of a trap due to error or program initiated instruction, the instruction is not completed and hence EPC points to the instruction which caused the trap.
  In this case, the return code should explicitly increment the value in EPC.

Exception Processing

- When an exception occurs and the MIPS processor reaches the state where the next instruction would be fetched, the CPU controller goes to a special state for processing the exception.
- The Cause register is updated with a code to identify the source of the interrupt.
- If the instruction that was executing involved a memory access that caused the error, then the memory address is stored in the BadVaddress register.
- KU and IE bits are copied to previous and old versions.
- Current values KUc and IEc are cleared, representing kernel mode and disabled interrupts.
- Return address is copied to the register EPC.
- PC is loaded with the address of the handler code. (In case of QtSpim, this is at 0x80000180).

Exception Processing

The return sequence for an external interrupt uses the code:

```
mfc0 $k0, $14 # EPC to $K0 register of the main processor
rfe           # Prepare to Return from exception
jr $k0        # Resume the original program
```

However, if we are returning due to a trap (includeing syscall), EPC contains unincremented PC. So the return sequence is:

```
mfc0 $k0, $14 # EPC to $K0 register of the main processor
addiu $k0, 4  # Update return address to next instruction
rfe           # Prepare to Return from exception
jr $k0        # Resume the original program
```

Exception Emulation in QtSpim

- QtSpim emulates exceptions, including internal exceptions such as overflow and address errors.
- If you “single step” through a program in QtSpim, it will not respond to interrupts generated.
- This is because single stepping is enabled through trap.
- However, using breakpoints, it is possible to run through exception and handler code.
- The exception handler code is loaded at address 0x80000180. This region of memory is called Kernel segment 0.
- QtSpim also includes a timer which can generate an interrupt after a specified amount of time has elapsed.