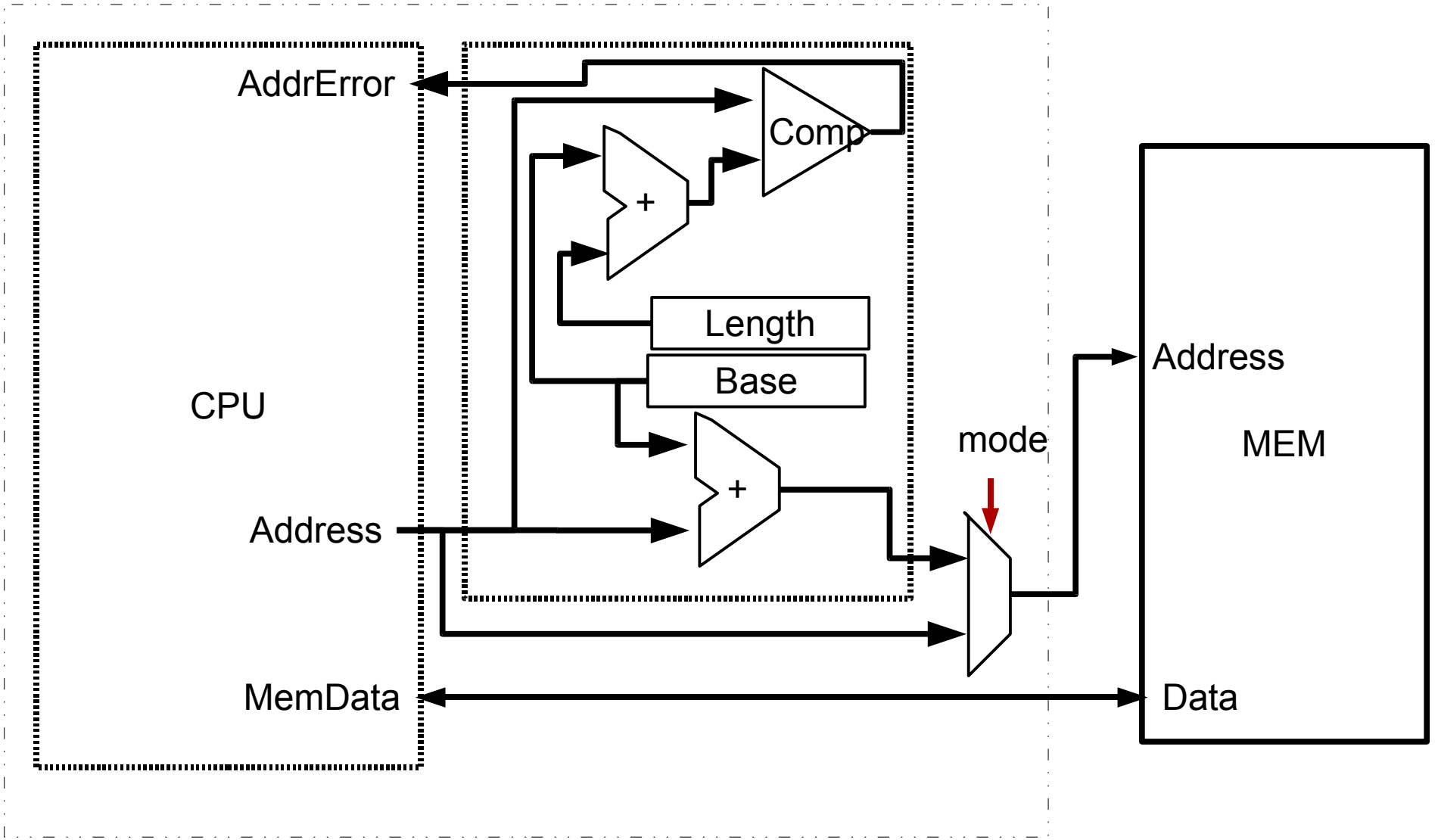
Virtual Memory/MMU

- Processor can be running in real mode or virtual mode
  - In real mode, the MMU does nothing, virtual addresses are identical to real addresses
  - In virtual mode, the MMU translates all memory addresses from the virtual addresses into physical addresses.
- Parts of the OS runs in real mode, user programs can run in virtual mode.

MMU


AddrError
Comp
+
Length
Base
+
CPU
Address
mode
Address
MEM
MemData
Data

Interrupt Styles

- Status Interrupt
  - Cause of interrupt is stored in a register
- Vectored Interrupt
  - Cause of interrupt is implicitly defined by the location jumped to when the interrupt occurs – the interrupt vector

MIPS Interrupt/Exception Implementation

- Access to Interrupt/Exception informations implemented as a co-processor
- All accessible information is stored in co-processor registers
- Special instructions
  - mfc0 $localreg, $cpreg #move from co-processor
  - mtc0 $localreg, $cpreg #move to co-processor
  - eret # return from exception

Coprocessor 0 Registers

Register name	Register number	Usage
BadVAddr	8	memory address at which an offending memory reference occurred
Count	9	timer
Compare	11	value compared against timer that causes interrupt when they match
Status	12	interrupt mask and enable bits
Cause	13	exception type and pending interrupt bits
EPC	14	address of instruction that caused exception
Config	16	configuration of machine

Status and Cause Reg Fields

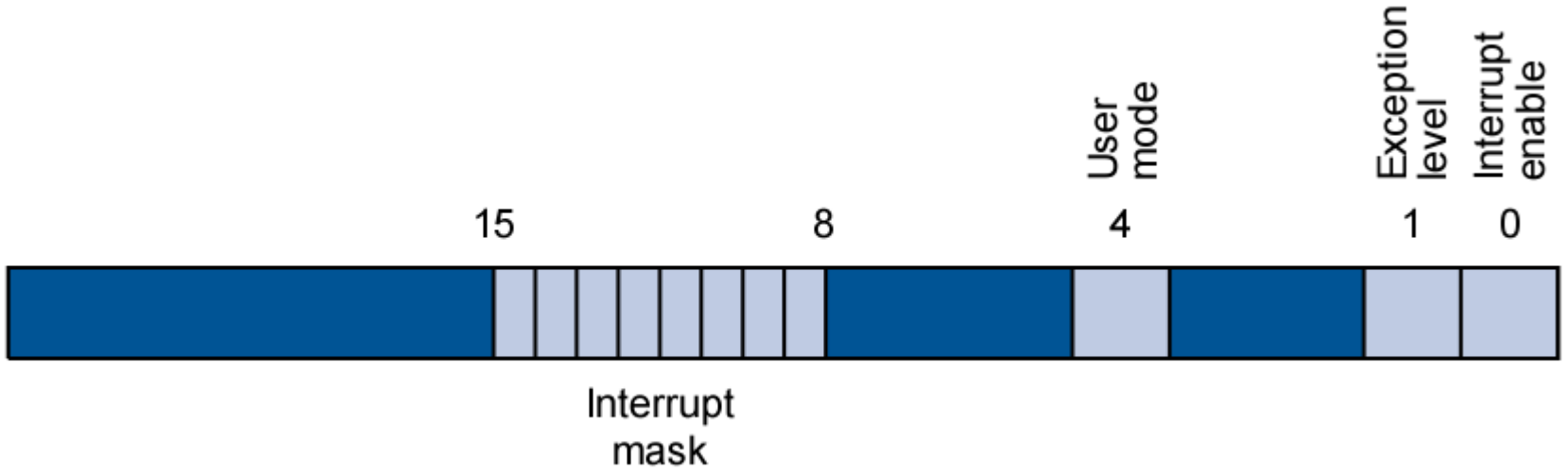


FIGURE A.7.1 The Status register.

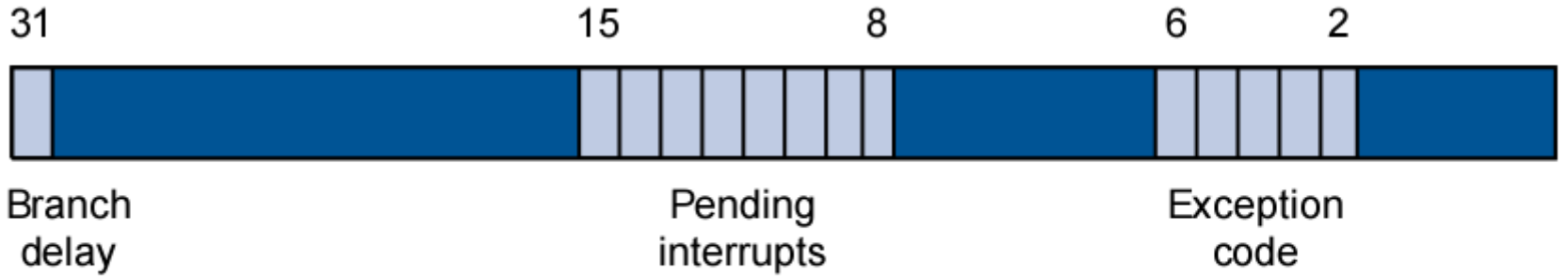


FIGURE A.7.2 The Cause register.

Cause Flags in SPIM

Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point

Simple Exception Handler

```
.ktext 0x80000180
mov  $k1, $at # Save $at register
sw  $a0, save0  # Handler is not re-entrant and can't use
sw  $a1, save1  # stack to save $a0, $a1
# Don't need to save $k0/$k1
mfc0  $k0, $13  # Move Cause into $k0
srl  $a0, $k0, 2 # Extract ExcCode field
andi $a0, $a0, 0xf
bgtz  $a0, done  # Branch if ExcCode is Int (0)
mov  $a0, $k0  # Move Cause into $a0
mfc0 $a1, $14  # Move EPC into $a1
jal  print_excp  # Print exception error message
```

Simple Exception Handler 2

```
done: mfc0  $k0, $14 # Bump EPC
addiu $k0, $k0, 4 # Do not reexecute
# faulting instruction
mtc0  $k0, $14 # EPC
mtc0  $0, $13 # Clear Cause register
mfc0  $k0, $12 # Fix Status register
andi  $k0, 0xfffd # Clear EXL bit
ori $k0, 0x1 # Enable interrupts
mtc0 $k0, $12
lw  $a0, save0 # Restore registers
lw $a1, save1
mov $at, $k1
eret  # Return to EPC
```