JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Sparc V8 Traps

Sparc V8 Traps

Traps are the mechanism for responding to events during program execution. Such events include:

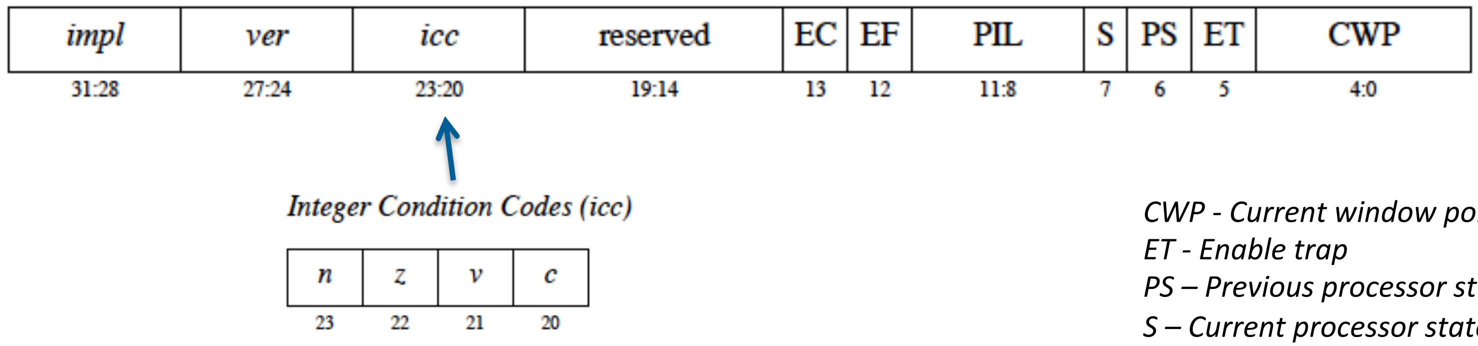
- Attempts to execute privileged or unimplemented instructions
- Exceptions such as divide by zero or overflow
- Attempted access to restricted or reserved memory areas
- Operating system service calls
- External interrupts

To understand trap handling, the processor state register must first be understood.

Sparc V8 Traps

Processor State Register

The processor state register is accessed using privileged read/write instructions (rdpsr & wrpsr).



impl	*ver*	*icc*	reserved	EC	EF	PIL	S	PS	ET	CWP
31:28	27:24	23:20	19:14	13	12	11:8	7	6	5	4:0

Integer Condition Codes (icc)

n	*z*	*v*	*c*
23	22	21	20

CWP - Current window pointer
ET - Enable trap
PS – Previous processor state
S – Current processor state
PIL – Processor interrupt Level
EF - Enable FPU
EC – Enable Coprocessor

JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Sparc V8 Traps

Sparc V8 Traps

A trap is a vectored transfer of control to supervisor code through a 256-entry trap table.

The trap base address (TBA) is held in the trap base register (TBR).

TBA	tt	zero
(31:12)	(11:4)	(3:0)

TBA = the most significant 20 bits of the trap base address

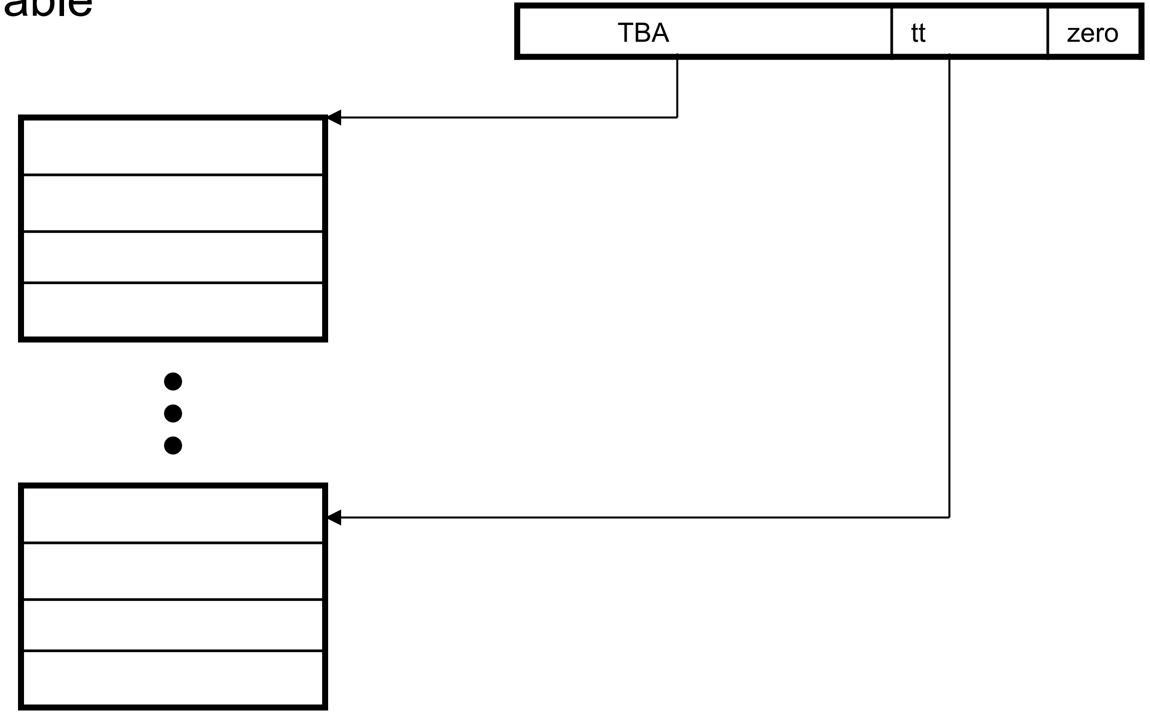
(tt) = an 8-bit number set by hardware when a trap occurs to identify the trap type

(zero) = Bits 3 through 0 are zeroes since each trap table entry contains 16 bytes (the first 4 instructions of the handler)

The TBA field within the TBR can be written by the privileged instruction WRTBR (write TBR).

Sparc V8 Traps

Trap Vector Table



Each entry is 4 words in size and contains the first four instructions of the corresponding trap handler.

Recall that the MIPS employs a single exception vector address through which all exceptions are funneled and examines the cause register to determine the action to take.

Sparc V8 Traps

Traps are like unsolicited or unexpected procedure calls

New register window is obtained when a trap occurs

  pc, npc and psr are saved

  copied into local registers within the new window

For traps other external reset, hardware generates a trap ID

trap ID is written into the tt field within the TBR

ET bit is cleared within PSR (preventing further traps)

Control is transferred into the trap table whose address is in the TBR

Sparc V8 Traps

Trap Table Entries

The tt field (in TBR) can specify 256 distinct types of trap

– half for hardware traps & half for software traps

ticc instruction generates software trap (for system service, etc.)

operand is software_trap#.   Example: ta   12   trap always

opcode	*cond*	*operation*	*icc test*
TA	1000	Trap Always	1
TN	0000	Trap Never	0
TNE	1001	Trap on Not Equal	not Z
TE	0001	Trap on Equal	Z
TG	1010	Trap on Greater	not (Z or (N xor V))
TLE	0010	Trap on Less or Equal	Z or (N xor V)
TGE	1011	Trap on Greater or Equal	not (N xor V)
TL	0011	Trap on Less	N xor V
TGU	1100	Trap on Greater Unsigned	not (C or Z)
TLEU	0100	Trap on Less or Equal Unsigned	(C or Z)
TCC	1101	Trap on Carry Clear (Greater than or Equal, Unsigned)	not C
TCS	0101	Trap on Carry Set (Less Than, Unsigned)	C
TPOS	1110	Trap on Positive	not N
TNEG	0110	Trap on Negative	N
TVC	1111	Trap on Overflow Clear	not V
TVS	0111	Trap on Overflow Set	V

Sparc V8 Traps

Trap Table Entries

Since the low 4 bits of the TBR are zero, table entries are on 16-byte boundaries

Each 4-word (16-byte) entry contains the first 4 of the trap handler's instructions

These 4 instructions can be used to call the corresponding handler

A reset will be performed if a trap occurs while the ET bit is 0.

A reset trap causes a transfer of control to address 0. (boot code)

Sparc V8 Traps

Returning from Traps (rett instruction)

The code sequence below returns to the instruction that caused the trap:

```
jmpl   %l1, %g0       Load PC with saved PC from local register 1
rett   %l2            Load NPC with the saved NPC
```

Alternative return to instruction following the one causing the trap:

```
jmpl   %l2, %g0       Load PC with saved NPC
rett   %l2 + 4        Load NPC with saved NPC+4
```

Memory mapped I/O is used and is based on interrupts or on polling