Topic V19

A Non-Reentrant

Exception Handler Example

No Readings associated with this lecture

An Example

Write an exception handler that prints the address of the instruction that caused the exception and the cause of the exception

Assume that there is a data area reserved for usage by the exception handler.

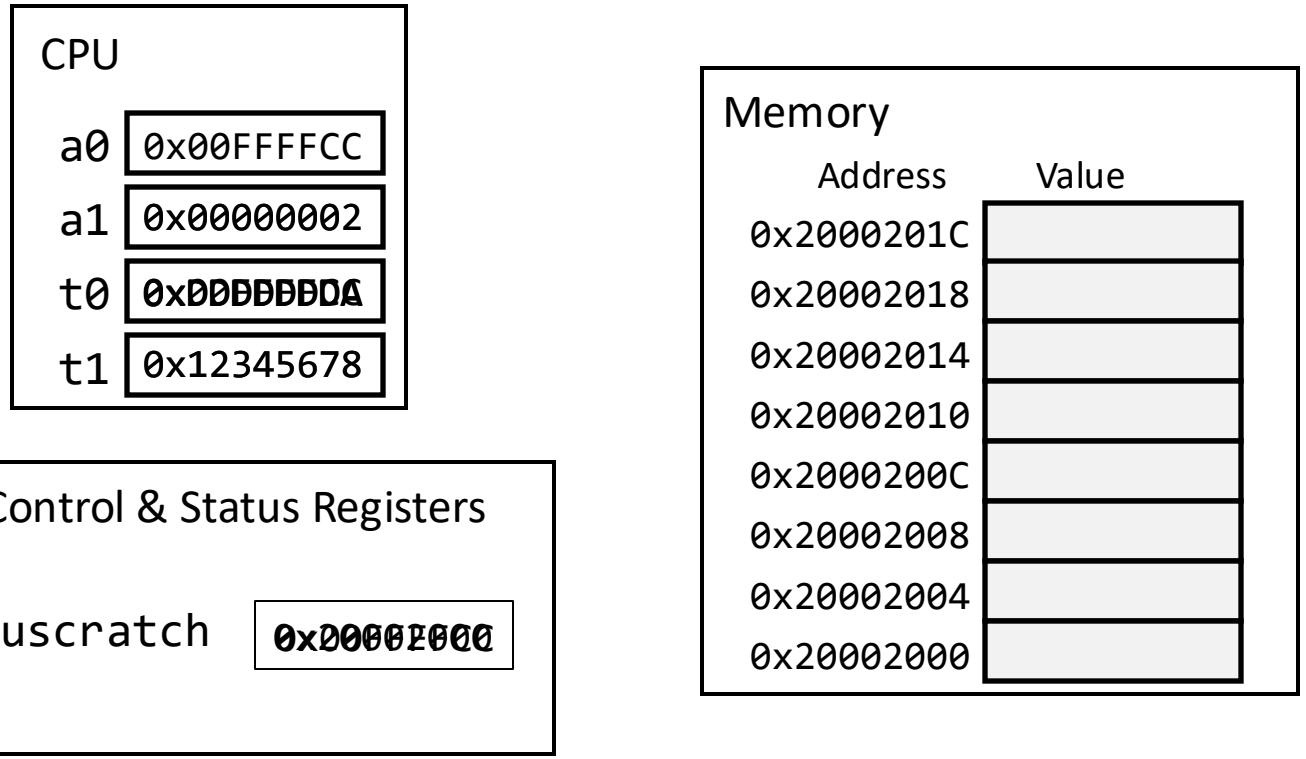
The address of this area has been placed in the uscratch register (CSR 0x040) at initialization.

Non-Reentrant Handler

Registers can be saved anywhere in the data area reserved for the handler.

There is no need to maintain a kernel stack or a ksp

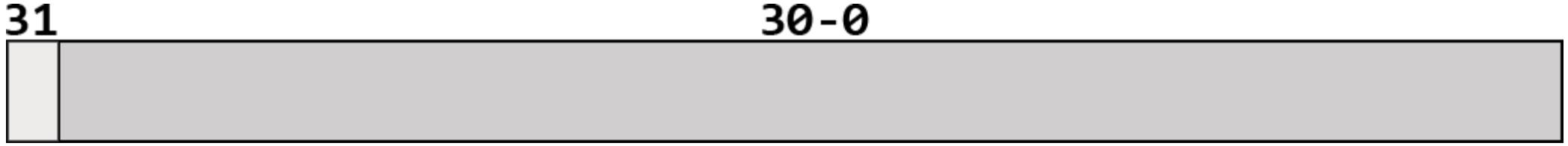
Saving registers used in the handle




```
# swap uscratch and a0
csrrw a0, 0x040, a0 # a0 <- Addr[iTrapData], uscratch <- USERa0
# save registers used in the handler EXCEPT a0
sw t0, 0(a0)
sw t1, 4(a0)
sw a1, 8(a0)
# store USERa0
csrr t0, 0x040 # t0 <- USERa0
sw t0, 12(a0) # save USERa0
```

Exception Handler (Example)

Cause register(ucause):




```
Handle:
# has to save any registers that the handle uses
li     t1, 0
csrrw  t0, 0x42 , t1        #  Move Cause to t0 and clear it
li     t1, 0x7FFFFFFF
and    a0, t0, t1           #  Extract Exception Code field
beqz   a0, done             #  Branch if ExcCode is zero
mov    a0, t0               #  Move Cause into a0
csrrsi a1, 0x41 , 0         #  Move EPC into a1
Jal    print_excp           # Print exception error message
```

Returning from exception

Before returning the handler must:

- Clear the Cause register
- Restore the values of any registers that it has used
- Execute uret (Exception return instruction)
  - uret copies uepc

```
done:
csrrsi  t0, 0x41 , 0   # copy EPC into t0
addi    t0, t0, 4      # incr. EPC to not re-execute faulting instructio
csrrw    t1, 0x41 , t0 # write incremented value to EPC
# load USERa0
la      a0, iTrapData # a0 <- Addr[iTrapData]
lw      t0, 12(a0)    # t0 <- USERa0
csrw    t0, 0x040     # uscratch <- usera0
# load registers used in the handler EXCEPT a0
lw      t0, 0(a0)
lw      t1, 4(a0)
lw      a1, 8(a0)
# swap uscratch and a0
csrrw   a0, 0x040, a0  # a0 <- USERa0, uscratch <- Addr[iTrapData]
uret                   # Return to EPC
```

Restore PC

Restore t0, t1, a0, a1

What happens if a timer interrupt happens while the handler is disabled to handle a keyboard interrupt?

James Prior, Senior Director of Product Marketing Communications
– October 24, 2019

Incredibly Scalable High-Performance RISC-V Core IP

—

Introducing the new SiFive U8-Series Core IP

SiFive is pleased to introduce the SiFive U8-Series Core IP, an incredibly scalable high-performance microarchitecture for modern SoC designs. The SiFive U8-Series is the highest performance RISC-V ISA based Core IP available today, based on a superscalar out-of-order pipeline with configurable pipeline depth and issue queue width. SiFive U8-Series Core IP is designed for use in performance- and latency-sensitive markets, such as automotive, datacenter attach, and edge or end point deep learning SoCs.

https://www.sifive.com/blog/incredibly-scalable-high-performance-risc-v-core-ip

A quad-core SiFive U84 CPU, including 2MB of L2 cache, requires only 2.63mm2 in 7nm process technology while capable of operating at up to 2.6GHz clock speed. A single SiFive U8-Series CPU core, without L2 cache, can be laid out in as little as 0.28mm2. This massive area reduction without performance impact reduces overall solution cost, or allows the use of silicon area for new compute functions.

$$\text{clock cycle time} = \frac{1}{\text{frequency}} = \frac{1}{2.6 \times 10^{9}} = 0.38 \times 10^{-9}$$

How many exception handlers can execute in between two timer interrupts?

Assume 200 instructions executed in a handler.

Assume 2 cycles/instruction.

⇒ A handler takes 400 cycles to execute.

A quad-core SiFive U84 CPU, including 2MB of L2 cache, requires only 2.63mm2 in 7nm process technology while capable of operating at up to 2.6GHz clock speed. A single SiFive U8-Series CPU core, without L2 cache, can be laid out in as little as 0.28mm2. This massive area reduction without performance impact reduces overall solution cost, or allows the use of silicon area for new compute functions.

Number handlers per second = $\frac{2.6 \times 10^9}{400}$ = 3.25×10^6 = 3,250,000