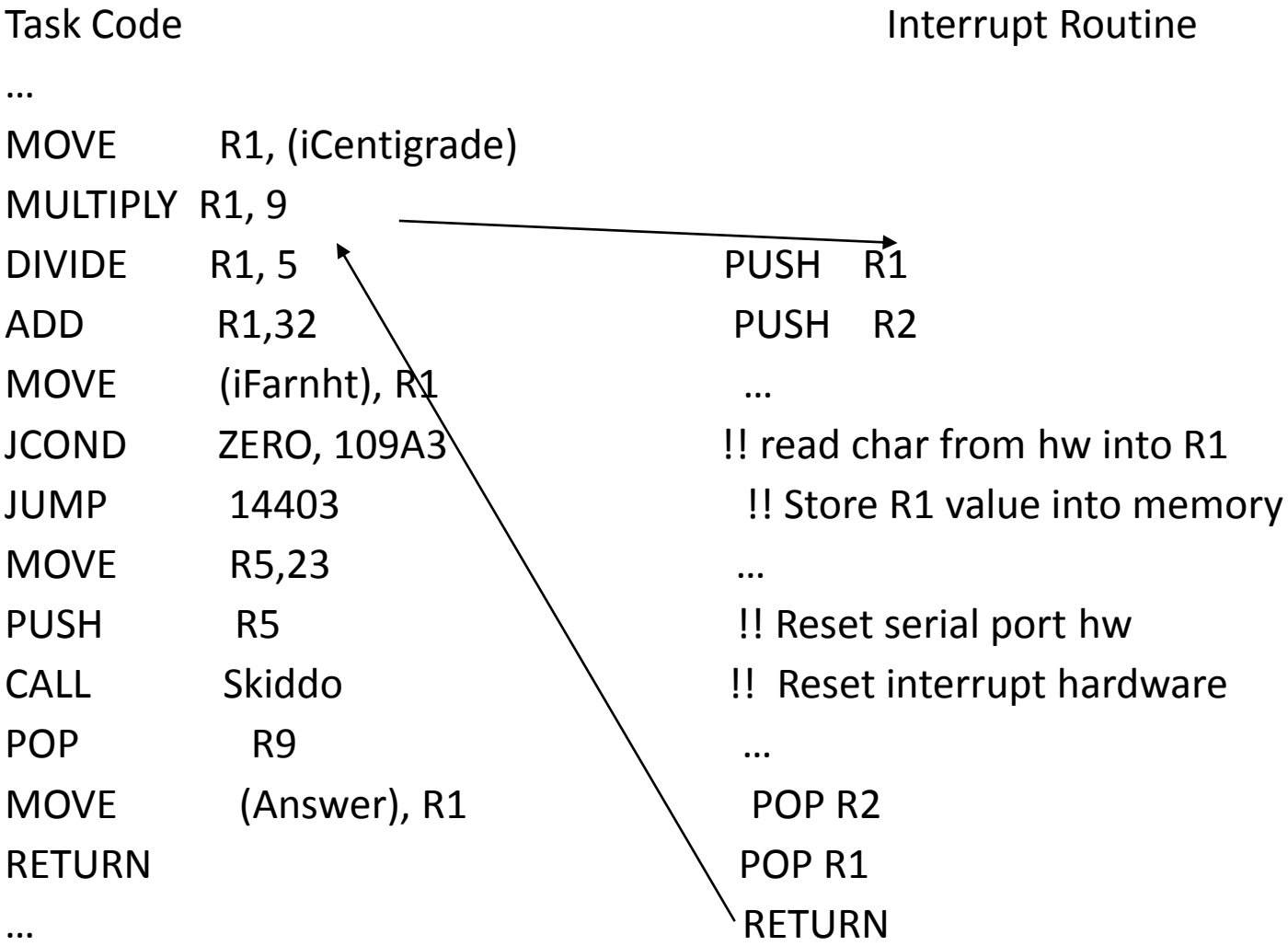
CS466 Chapter 4

Interrupt Basics

Interrupt Basics…

- Interrupts start with an input from the hardware, usually an interrupt request is asserted by some hardware which needs service. This device normally asserts the interrupt request line or IRQ on the microprocessor.

Figure 4.3 ISR

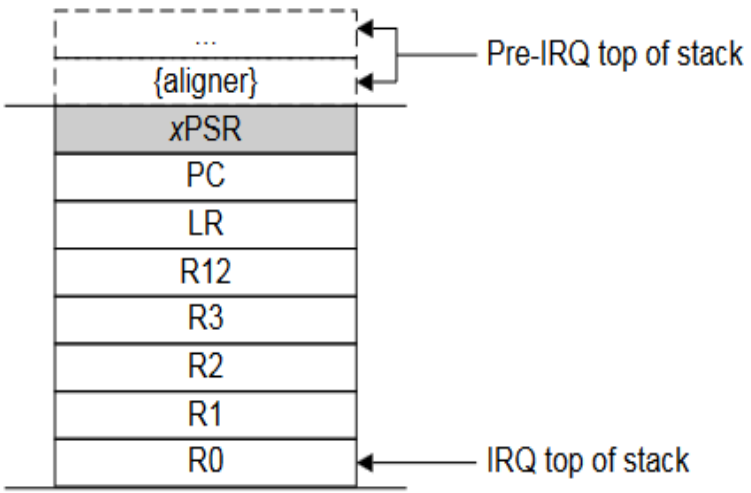


Saving and Restoring Context

- Pushing all the registers at the beginning of an isr and popping at the end is known as saving and restoring the context.
- This is CRITICAL. Failing to do the above can cause troublesome bugs.

ARM Cortex M4 Exception Entry

1. When the processor takes an exception the processor pushes information onto the current stack. This operation is referred to as *stacking* and the structure of eight data words is referred as the *stack frame*.
2. Immediately after stacking, the stack pointer indicates the lowest address in the stack frame.
3. The stack frame includes the return address. This is the address of the next instruction in the interrupted program. This value is restored to the PC at exception return so that the interrupted program resumes.



Exception frame without floating-point storage

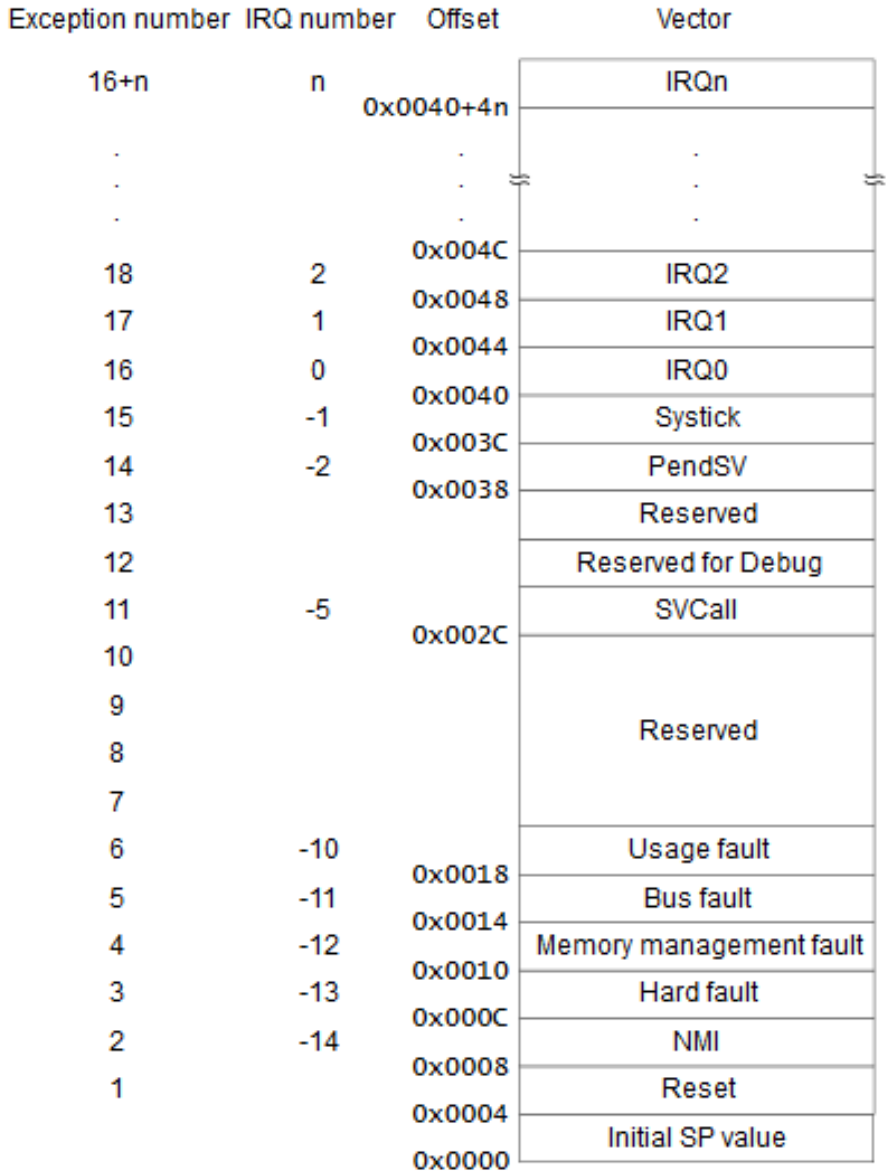
ARM Cortex M4 Exception (cont)

4. In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time
5. If no higher priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.
6. If another higher priority exception occurs during exception entry, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception. This is the late arrival case.
7. The ISR, executes and if it uses registers beyond those in the stack frame, It needs to push and eventually restore the registers before performing the interrupt return.
8. Generally the ISR will perform the handling for the interrupt and clear the interrupt in hardware.

ARM Cortex M4 Exception Return(cont)

9. Exception return occurs when the processor is in Handler mode and executes one of the following instructions to load the EXC_RETURN value into the PC:
   - an LDM or POP instruction that loads the PC
   - an LDR instruction with PC as the destination
   - a BX instruction using any register.

Figure 2.2. Vector table



Disabling Interrupts

- Interrupt Priority Levels
- Non-Maskable Interrrupts
- Critical Regions of Code
- Atomicity
- Etc etc

4.3 The Shared-Data Problem

- As soon as we use interrupts, a shared data problem arises.
- Guidelines of perhaps 10% maximum of cpu cycles in isr's are sometimes given. This is to increase responsiveness.
- Therefore, we are driven to perform as much work as possible at the task level and we generally use isrs to signal a task that there is work to do.
- Therefore, isr's and tasks must share common data.

Fig 4.4 Shared-Data (cont)

```
static int iTemperatures[2];                    // global to task and isr
void interrupt vReadTemperatures(void)
{
        iTemperatures[0] = !! read in value from hw
        iTemperatures[1] = !! read in value from hw
}
void main(void)
{
        int iTemp0, iTemp1;
        while(TRUE)
        {
                iTemp0 = iTemperatures[0];
                iTemp1 = iTemperatures[1];
                if(iTemp0 != iTemp1)
                        !! Set off howling alarm;
        }
 }
```

Fig 4.5 Harder Shared Data Problem

```
static int iTemperatures[2];

void interrupt vReadTemperatures(void)
{
    iTemperatures[0] = !! read in value from hardware
    iTemperatures[1] = !! read in value from hardware
}

void main(void)
{
    while(TRUE)
    {
        if(iTemperatures[0] != iTemperatures[1])
            !! Set off howling alarm
    }
}
```

Assembly language equivalent

```
.
.
MOVE      R1, (iTemperatures[0])
MOVE      R2, (iTemperatures[1])
SUBTRACT      R1,R2
JCOND     ZERO, TEMPERATURES_OK
.
.             Code goes here to set off the alarm…
.
TEMPERATURES_OK:
```

Characteristics of the shared data bug

- The iTemperatures[] array is shared between the interrupt routine and the task code.
- These are fiendish bugs…all works ok unless the interrupt just happens to occur in a small window of vulnerability.
  - 5:00 on Friday afternoon
  - Any time you are not working on it
  - Whenever no debugging tools/equip attached
  - After landing on Mars
  - During Customer Demos

Solving the Shared Data Prob

- Disable Interrupts whenever the task code uses the shared data…
- No C compilers or assemblers are smart enough to figure out when to disable or enable the interrupts!

Atomic vs. Critical

- A part of a program is said to be **atomic** if it cannot be interrupted.
- The shared data problem arises in part because the task code uses the data in a way that is not atomic.
- A set of instructions that must be atomic for the system to perform correctly is called a **critical section**.

Another potential solution…

Figure 4.11 illustrates another potential solution…

Does

Return(lSecondsToday) work correctly? If

it is a long (say 32 bit int) and we have an 8 bit microprocessor?

The volatile keyword

- When we allow the compiler to optimize, it makes the assumption that values stay in memory until the program changes it. This can and does cause problems.
- The compiler can optimize out certain assignments logically…talk about this.
- Declare keywords volatile to warn the compiler that an identifier may be changed with it not knowing…

Interrupt Latency

- Interrupts are a tool for getting better response from our system
- The speed of response of an embedded system is always of interest
- The term **interrupt latency** refers to the amount of time that it takes a system to respond to an interrupt.

Interrupt latency (cont)

- The following are contributing factors:
  - The longest period of time during which that interrupt is (or all interrupts are) disabled.
  - The period of time that it takes to execute any interrupt routines for interrupts that are of higher priority than the one in question.
  - How long it takes the microprocessor to stop what it is doing, do the necessary bookkeeping, and start executing instructions in the interrupt routine.
  - How long it takes the interrupt routine to save the context and then do enough work to count as a response.