Exception Handling

Involves close interaction between hardware and software.

Exception handling is similar to a procedure call with important differences:

- processor prepares exception handling: save* part of the current processor state before execution of the software exception handler
- assigned to each exception is an exception number, the exception handler's code is accessible via some exception table that is configurable by software
- exception handlers run in a different processor mode with unrestricted access to the system resources.

* in special registers or on the stack – we will go into the details for some architectures
Recall: ARM Processor Modes

ARM from v5 has (at least) seven basic operating modes

- Each mode has access to own stack and a different subset of registers
- Some operations can only be carried out in a privileged mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description / Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>Reset / Software Interrupt</td>
</tr>
<tr>
<td>FIQ</td>
<td>Fast Interrupt</td>
</tr>
<tr>
<td>IRQ</td>
<td>Normal Interrupt</td>
</tr>
<tr>
<td>Abort</td>
<td>Memory Access Violation</td>
</tr>
<tr>
<td>Undef</td>
<td>Undefined Instruction</td>
</tr>
<tr>
<td>System</td>
<td>Privileged Mode with same registers as in User Mode</td>
</tr>
<tr>
<td>User</td>
<td>Regular Application Mode</td>
</tr>
</tbody>
</table>
Recall: ARM Register Set

ARM has 37 registers, all 32-bits long
A subset is accessible in each mode
Register 13 is the Stack Pointer (by convention)
Register 14 is the Link Register**
Register 15 is the Program Counter (settable)
CPSR* is not immediately accessible

* current / saved processor status register, accessible via MSR / MRS instructions
** more than a convention: link register set as side effect of some instructions
## Exception handling on ARM

### Hardware action at entry (invoked by exception)
- R14(exception_mode):= return link
- SPSR(exception_mode) := CPSR
- CPSR[4:0] := exception_mode number
- CPSR[5] := 0 (* execute in ARM state *)
  - If exception_mode = Reset or FIQ then CPSR[6]=1 (* disable fast IRQ *)
- CPSR[7]=1 (* disable normal interrupts *)
- PC=exception vector address

### Software
- STMDB SP!, {R0 .. R11, FP, LR} (* store all non-banked registers on stack *)
- ... (* exception handler *)
- LDMIA SP! {R0..R11,FP,LR} (* read back all non-banked registers from stack*)
- SUBS PC,LR, #ofs (* return from interrupt instruction *)

### Hardware action at exit (invoked by MOVS or SUBS instruction)
- CPSR := SPSR(exception mode) (* includes a reset of the irq/fiq flag *)
- PC := LR – ofs
Initialization of Exceptions

InstallHandler(SWITrap, Platform.SWI);
InstallHandler(....);
....

FOR i := 0 TO 7 DO
    SYSTEM.PUT32(ExceptionVectorBase + 4*i, 0E59FF018H);
END;
Enable/Disable IRQs

PROCEDURE EnableIRQs*;
VAR cpsr: SET32;
BEGIN
    SYSTEM.STPSR(0, cpsr);
cpsr := cpsr - {7};
SYSTEM.LDPSR(0, cpsr)
END EnableIRQs;

PROCEDURE DisableIRQs*;
VAR cpsr: SET32;
BEGIN
    SYSTEM.STPSR(0, cpsr);
cpsr := cpsr + {7, 8};
SYSTEM.LDPSR(0, cpsr)
END DisableIRQs;

* reverse cmp/sub meaning compared with x86
Install Timer

Platform.WriteWord(Platform.STC1, 
    Platform.ReadWord(Platform.STCLO)+Platform.TimerInterval);

Platform.WriteBits (Platform.STCS, {1});

nextTimerInterrupt := Platform.ReadWord(Platform.STC1);

EnableIRQ(Platform.SystemTimerIRQ, TRUE);

Sets bit in IRQEnable registers
cf. BCM2835 ARM Peripherals document, Chapter 7, p. 109ff
The System Timer on RPI

- The system timer on RPI is described in BCM2835 document, chapter 12.
- It provides 4 timer match registers, where two of them (1 and 3) are available to the ARM Core.
- The system timer is either driven by the APB (peripheral bus), running with 1/2 cpu frequency or from the crystal (19.2 MZh).
- Default source on the RPI is the APB with a timer divider register (described in Quad A7 control document) initialized with 0x06aaaab corresponding to a divider of about 19.2 => 1 MHz timer frequency.
- The System timer IRQs are GPU#1 1 and 3.
- The ARM Timer (chapter 14) is driven from the (variable) GPU frequency.
Install IRQ (Uart)

Kernel.EnableIRQ( Platform.UartInstallIrq , TRUE );

Kernel.InstallIrqHandler(Platform.UartEffectiveIrq , UartHandler0);

cf. BCM2835 ARM Peripherals document, Chapter 7, basic pending register (p. 110 + 113)
IRQ Trap Handler

PROCEDURE {INTERRUPT, PCOFFSET=4} IRQTrap;
VAR i, j, spsr: SIZE; basicPending, pending1, pending2: SET32;
BEGIN
SYSTEM.STPSR( 1, spsr ); (* store SPSR *)
(* read pending bits *)
...
(* disable corresponding device interrupts *)
...
(* cf BCM2835 Manual, Section 7.5 *)
(* process pending bits and call irq handler*)
...
SYSTEM.LDPSR( 1, spsr ); (* SPSR := old *)
END IRQTrap;
(*page fault*)
PROCEDURE {INTERRUPT, PCOFFSET=8} DataAbort;
VAR lnk, fp: LONGINT;
BEGIN
  (* The location that trapped was lnk – 8 *)
  lnk := SYSTEM.LNK – 8;
  fp := SYSTEM.FP;
  IF trapHandler # NIL THEN
    trapHandler(Platform.DataAbort, lnk, fp)
  ELSE
    (* diagnostics output and halt *)
  END
END DataAbort;
SWITrap handler

PROCEDURE {INTERRUPT, PCOFFSET=0} SWITrap;
(* software interrupt (e.g. failed ASSERT) *)
  VAR lnk, fp: LONGINT;
BEGIN
  (* The location that trapped was lnk - 4 *)
  lnk := SYSTEM.LNK - 4;
  fp := SYSTEM.FP;
  IF trapHandler # NIL THEN
    trapHandler(Platform.SWI, lnk, fp) (* stack trace *)
  END
END SWITrap;
SWITrap handler

PROCEDURE {INTERRUPT, PCOFFSET=0} SWITrap;
(* software interrupt (e.g. failed ASSERT) *)
  VAR lnk, fp: LONGINT;
BEGIN
  (* The location that trapped was lnk - 4 *)
  lnk := SYSTEM.LNK - 4;
  fp := SYSTEM.FP;
  IF trapHandler # NIL THEN
    trapHandler(Platform.SWI, lnk, fp) (* stack trace *)
  END
END SWITrap;

All registers are saved during entering the trap. Get the original FP (reg12 – not banked) from the local stack and traverse the stack.
1.4. TASK SCHEDULING
Scheduling Strategy

- **Task types**
  - High priority synchronous tasks (scheduled each 5 ms)
  - Low priority synchronous tasks (scheduled each 20 ms)
  - Background tasks

- **Rules of preemption**
  - High priority tasks preempt all others
  - Low priority tasks preempt background tasks
  - Background tasks don’t preempt
Scheduling Example

Priority

High

Low

BG

time % interval = 0

Input

Navigation

Logger

200 Hz Rate

50 Hz Rate

Time [ms]
A Stack for each Process?
One Stack for All?

When, How?
Run to completion!
Preemption
Stack Organisation

Stack

BG

PAF Schedule
Low priority

PAF Schedule
High priority

vars

vars

vars

Where is the process context?
**Tasks**

- Descriptors for *asynchronous* (background) tasks

  ```
  Task* = POINTER TO TaskDesc;
  TaskDesc* = RECORD
    next: Task;
    proc: TaskCode;
    name: ARRAY 32 OF CHAR;
  END;
  ```

- Descriptors for *synchronous* (periodic) tasks

  ```
  PeriodicTaskDesc* = RECORD (TaskDesc)
    interval: LONGINT;
    subPriority: LONGINT;
    nextTime: LONGINT;
  END;
  ```

PROCEDURE (me: Task)
Scheduler

- Recursive interrupt procedure

<table>
<thead>
<tr>
<th>Prolog (Interrupts masked)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduling (Interrupts allowed)</td>
</tr>
<tr>
<td>Epilog(Interrupts masked)</td>
</tr>
</tbody>
</table>

- Must be reentrant
  - Register values on stack
  - Private variables

- Assume that \(\text{Interval} (\text{low})\) is a multiple of \(\text{Interval} (\text{high})\)
Context change, schematic

Before the interrupt

In the interrupt handler

*Processor Status Word
Process Context

SVC Mode
- SP
- LR

SVC Stack
- BG
- Scheduler
- Low
- Scheduler
- High

IRQ Stack
- R12
- ...
- R1
- R0

Process Context
- SVC LR
- Handler

IRQ Mode
- SP
- LR
Some tricks required ...

Kernel.TimerIrqHandler

VAR lr: INTEGER;
BEGIN
    INC( timer, Platform.UNIT );
    IF timerHandler # NIL THEN
        SYSTEM.LDPSR( 0, SVCMode + IRQDisabled );
        globalLR := SYSTEM.LNK();
        SYSTEM.LDPSR( 0, IRQMode + IRQDisabled );
        lr := globalLR;
        SYSTEM.LDPSR( 0, Platform.SVCMode );
        timerHandler;
        SYSTEM.LDPSR( 0, IRQMode + IRQDisabled );
        globalLR := lr;
        SYSTEM.LDPSR( 0, SVCMode + IRQDisabled );
        SYSTEM.SETLNK(globalLR);
        SYSTEM.LDPSR( 0, IRQMode + IRQDisabled );
    END;

Switch to SVC mode, no IRQs
Scheduler Code

Assumptions:

- linked list stores tasks sorted by period / priority
- tasks run to completion within given period
Rate Monotonic Scheduling

Minos.Scheduler

currentTime := Kernel.GetTime();
current := periodicTasks;
WHILE current # NIL DO
    IF currentTime MOD current.interval = 0 THEN
        current.proc( current )
    END;
    current := current.next(PeriodicTask);
END;