Advanced Micro-controllers

Agenda

- ARM Cortex-A Architecture
  - Modes & Registers
  - CPSR Register
  - Exceptions
  - Exception handling
  - ARM7 VIC
  - IRQ vs FIQ
- ARM assembly instructions
  - ADD, SUB
  - ADDS, SUBS, MOVS
  - NZCV flags in xPSR/CPSR
  - ~~MUL, UDIV~~
  - ~~MVN~~
  - ~~AND, ORR, EOR, BIC, ANDS, TST~~

ARM Cortex-A Architecture

Exceptions

- Due to exception, normal flow of execution is deviated/differs.
- Flow of execution
  - User program is executing
  - Exception arise
  - Current execution is paused
  - Exception handler is executed

- Paused execution is resumed

- ARM Cortex-A Exceptions
    - Reset -- When processor is reset i.e. power off/on
    - Software interrupt -- When SWI/SVC instruction is executed i.e. software interrupt
    - Prefetch Abort -- When invalid address used for instruction fetch (PC).
    - Data Abort -- When invalid address is used for data.
    - Undef -- When instruction is not known to decoder.
    - IRQ -- Normal priority interrupt
    - FIQ -- Fast/High priority interrupt
- When exception occurs, mode change -- CPSR[4:0]
    - Exception rst or swi --> SVC mode (10011)
    - Exception irq --> IRQ mode (10010)
    - Exception fiq --> FIQ mode (10001)
    - Exception und --> UND mode (11011)
    - Exception pabt or dabt --> ABT mode (10111)
    - USR(10000) / SYS(11111) modes are entered programatically (not due to exception).
- Exception priority, vector address
    - Exception rst --> 1 - 0x00 | I=1, F=1
    - Exception und --> 6 - 0x04 | I=1, F=0
    - Exception swi --> 6 - 0x08 | I=1, F=0
    - Exception pabt --> 5 - 0x0C | I=1, F=0
    - Exception dabt --> 2 - 0x10 | I=1, F=0
    - Exception irq --> 4 - 0x18 | I=1, F=0
    - Exception fiq --> 3 - 0x1C | I=1, F=1
- Sloss Book: Chapter 2, Chapter 9 - 9.1, 9.2

PSR registers (Cortex-A)

- cpsr is shared register for all modes.
- Different spsr is available in modes - irq, fiq, undef, abort, svc.
- During exception cpsr is copied into spsr (by CPU core).
- cpsr -- 32 bit register
    - flags section -- ALU flags
        - N -- Negative
        - Z -- Zero
        - C -- Carry (carry from 31st bit - MSB)
        - V -- Overflow (for signed calc -- carry from 30th bit)
        - Q -- Saturated Math
        - de -- If-Then instruction
        - J -- Jazelle core
    - status section
        - GE (4 bits) -- SIMD operation
    - execution section
        - abc -- If-Then instruction
        - E -- Endianness
            - 0 = Data load/store operation is Little endian
            - 1 = Data load/store operation is Big endian
        - A -- Precise data abort
    - control section
        - mode -- 5 bits
        - T bit -- Thumb bit
        - I bit --
            - When I=1, IRQs are masked/disabled.
            - When I=0, IRQs are unmasked.
        - F bit --
            - When F=1, FIQs are masked/disabled.
            - When F=0, FIQs are unmasked.

ARM states

- ARM state => T=0
  - Only ARM instructions (32-bit instruction) are executed
  - PC always incremented by 4 bytes.
  - PC word aligned (i.e. multiple of 4 bytes) i.e. last 2 bits = 0 -- PC [31:2]
- Thumb state => T=1
  - Only Thumb instructions are executed.
  - PC incremented by 2 bytes.
  - PC half-word aligned (i.e. multiple of 2 bytes) i.e. last 1 bits = 0 -- PC [31:1]
- Jazzelle state => T=0 and J=1
  - PC incremented by 4 bytes.
  - Multiple bytecode instructions are fetched at once.

Interrupt handling

```
IRQ_Handler: @ARM7
    @- Adjust and save LR_irq in IRQ stack
    sub lr, lr, #4      @ this is done in ARM core for Cortex-A
    stmfd sp!, {lr}
    @- Save SPSR need to be saved for nested interrupt
    mrs r14, SPSR
    stmfd sp!, {r14}
    @- Save and r0 in IRQ stack
    stmfd sp!, {r0}
    @- Load the ISR-Address from VICVectAddr
    ldr r14, =LPC_BASE_VIC
    ldr r0 , [r14, #VIC_VectAddr]
    @- Enable Interrupt and Switch in Supervisor Mode (for nested intrs)
    msr CPSR_c, #Mode_SVC
    @- Save scratch/used registers and LR in User Stack
    stmfd sp!, { r1-r3, r12, r14 }
    @- Branch to the routine pointed by the VIC_VectAddr -- CALL ISR
    mov r14, pc
    bx r0
    @- Restore scratch/used registers and LR from User Stack
    ldmfd sp!, { r1-r3, r12, r14 }
    @- Disable Interrupt and switch back in IRQ mode (for nested intrs)
    msr CPSR_c, #I_Bit | Mode_IRQ
    @- Restore R0
    ldmfd sp!, {r0}
    @- Restore SPSR_irq from IRQ stack
    ldmfd sp!, {r14}
    msr SPSR_cxsf, r14
    @- Restore adjusted LR_irq from IRQ stack directly in the PC, ^ sign copies spsr into cpsr.
    ldmfd sp!, {pc}^
```

- LPC2148 (ARM7) Interrupt Programming -- VIC

```
// timer0 peripheral number is 4
VICIntSelect &= ~BV(4); // IRQ=0
VICVectCntl5 = BV(EN) | 4;
VICVectAddr5 = (uint32_t)timer0_isr;
VICIntEnable |= BV(4);  // Enable interrupt
```

```
void timer0_isr(void) {
    // handle the peripheral interrupt
}
```

ARM Cortex-M Assembly Programming

- basic.s (assembly code) --> Assembler (arm-none-eabi-as) --> basic.o (object code)

```
> arm-none-eabi-as -mcpu=cortex-m3 -march=armv7 -mthumb -gwarf2 -o basic.o basic.s
```

- basic.o (object code) --> Linker (arm-none-eabi-ld) --> basic.elf (executable code)

```
> arm-none-eabi-ld -T./lm3.ld -o basic.elf basic.o
```

- To inspect the elf file

```
> arm-none-eabi-readelf -a basic.elf

> arm-none-eabi-objdump -h basic.elf

> arm-none-eabi-objdump -S basic.elf

> arm-none-eabi-objdump -t basic.elf
```

- basic.elf (executable code) --> Objcopy (arm-none-eabi-objcopy) --> basic.bin (binary image)

```
> arm-none-eabi-objcopy -O binary basic.elf basic.bin
```

- basic.bin --> ARM Cortex-M3 or ARM emulator (QEmu)
- To burn basic.bin into qemu

```
> qemu-system-arm -machine lm3s811evb -cpu cortex-m3 -S -d "in_asm,int,exec,cpu,guest_errors,unimp" -gdb
tcp::1234 -nographic -kernel basic.bin
```

- To debug basic program on qemu

```
> arm-none-eabi-gdb basic.elf -ex "target remote localhost:1234"
```

- Debugging commands

```
> break main

> cont

> step

> next

> info registers r0 r1 cpsr

> quit
```

- To stop qemu

```
> pkill qemu-system-arm
```