ECE 598 – Advanced Operating Systems
Lecture 7

Vince Weaver

http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

13 February 2018 (!)

Announcements

- Homework #3 was assigned, due Thursday
- Be sure you have a serial cable if you need it.

HW#2 Review

- Code: everyone's code blinked fine
  Sorry for the confusion, the C code should have specified to blink GPIO18, not ACT
  Timing of the blinking was not consistent, but possibly that was due to Pi2/Pi3 differences.
- Size: C about 200 bytes, assembly 68 bytes?
  Can look at .dis files for disassembly
  C: 60 bytes of initialization, asm: 12 bytes for delay loop, on C is 56 bytes (due to pessimization from volatile, etc)

also saves/restores LR and registers to maintain calling convention. can't explain some of it
stdint.h shouldn't add anything, just defines. Usually in C you shouldn't include code in header files. Loo for self, /usr/include/stdint.h (though often includes other files and lots of #defines

- volatile – have C compiler not optimize away stores
- C array of 32-bit ints vs actually byte-wise access
- SPI1_CEN_0. Bonus SPI ports

What are interrupts?

- A way to let hardware/software interrupt execution to let the CPU know something important has happened.
- Notified immediately of something happening (as opposed to polling, checking occasionally)
- Without interrupts processes can get stuck/greedy and never let go of what they are doing.
- Do you need precise interrupts?
- Are interrupts good or bad?
  - Can reduce latency... or make it worse (real-time, slow

handler)

- Can add overhead. On OoO need to flush entire pipeline, then enter kernel. Slow slow slow.
- Some HPC or virtual turn off interrupts if possible.

What generates interrupts?

- What types of hardware generate interrupts? Keyboard, timers, Network, Disk I/O, serial etc.
- Some can be critical. Not empty UART FIFO fast enough can drop data on floor.
- What is most frequent interrupt on typical OS? Timer interrupt. regular timer. What is used for?
  - Context switching
  - Timekeeping, time accounting

Typical Interrupts

- Tell pointless 6502/Mockingboard example
- Set up interrupt source (Timer at 50Hz?)
- Install interrupt handler (usually vector at address that jumps to your code to handle things)
  - Handler should be fast, do whatever it needs to do (my case, load up 14 registers with data) or even schedule more work than later
  - Disable interrupts if HW didn't for us. Save/restore any registers we're going to change so when we return

no one notices

  - ○ Handler should ACK the interrupt (let hardware know we handled things so it doesn't retrigger as soon as we exit)

- Enable interrupts on device (often a flag to set)
- Enable (unmask) interrupts on your CPU. Often a processor flag.

Exceptions and Interrupts

- All architectures are different
- ARM does it a little differently from others.

How to find out?

- ARM ARM for ARMv7 (2700+ pages)
- Look at Linux source code
- Look at Raspberry Pi Forums

ARM has various Modes

- Modes:
- States
  - ISA: ARM (normal), Thumb, Jazelle, ThumbEE
  - Execution state (?)
  - Security: Secure and Non-secure
- Privlege Level
  - If secure: PL0 = user, PL1 = kernel
  - If non-secure: PL0 = user, PL1 = kernel, PL2 = hypervisor

ARM Modes

User	PL0	
FIQ	PL1	fast interrupt
IRQ	PL1	interrupt
SVC	PL1	supervisor
MON	PL1	monitor (only if security extensions)
ABT	PL1	abort
HYP	PL2	hypervisor (only if virtual extensions)
UND	PL1	undefined instruction
SYS	PL1	system

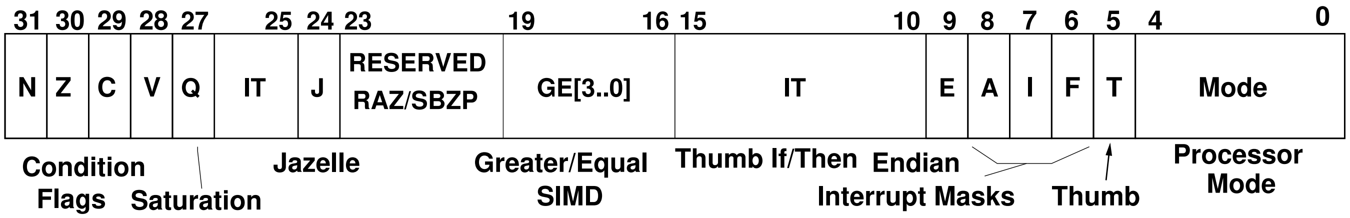
ARM Modes – continued

- User mode – unprivledged, restricted. Can only move to higher level by exception.
- System Mode – like USER, but no restrictions on memory/registers. Sort of like running as root, cannot enter by exception.
- Supervisor – kernel mode. SVC (syscall) instructions take you here. Also at reset (boot).
- Abort – called if a memory or prefetch causes an exception

why is this useful? Virtual memory.

- Undefined – called when undefined instruction happens why is this useful? Emulator?
- FIQ/IRQ – fast or normal interrupt
- HYP – hypervisor, for virtualization. A bit beyond this class.
- Secure – secure mode, can lock things down.

ARM CPSR Register



31	30	29	28	27	25	24	23	19 16	15 10	9	8	7	6	5	4 0
N	Z	C	V	Q	IT	J	RESERVED RAZ/SBZP	GE[3..0]	IT	E	A	I	F	T	Mode

Condition Flags

Saturation

Jazelle

Greater/Equal SIMD

Thumb If/Then

Endian

Interrupt Masks

Thumb

Processor Mode

- Current Program Status Register
- Contains flags in addition to processor mode
- Six privileged modes
- One non-privileged: user (cannot write CPSR), now APSR?
- Interrupts and exceptions automatically switch modes

ARM Interrupt Registers

User/Sys	Hyp	Fast	IRQ	Supervisor	Undefined	Abort	Monitor
r0							
r1							
r2							
r3							
r4							
r5							
r6							
r7							
r8		r8_fiq					
r9		r9_fiq					
r10		r10_fiq					
r11		r11_fiq					
r12		r12_fiq					
r13/sp	SP_hyp	SP_fiq	SP_irq	SP_svc	SP_und	SP_abt	SP_mon
r14/lr		LR_fiq	LR_irq	LR_svc	LR_und	LR_abt	LR_mon
r15/pc							
apsr							
cpsr	spsr_hyp	spsr_fiq	spsr_irq	spsr_svc	spsr_und	spsr_abt	spsr_mon
	ELR_hyp						

Unlike other architectures, when switching modes the ARM hardware will preserve the status register, PC and stack and give you mode-specific versions (register bank switching). Also for Fast Interrupts r8-r12 are saved as well, allowing fast handlers that do not have to save registers to the stack.

ARM Interrupt Handling

- ARM core saves CPSR to the proper SPSR
- ARM core saves PC to the banked LR (possibly with an offset)
- ARM core sets CPSR to exception mode (disables interrupts)
- ARM core jumps to appropriate offset in vector table

Vector Table

Type	Type	Offset	LR	Priority
Reset	SVC	0x0	–	1
Undefined Instruction	UND	0x04	lr-4/2	6
Software Interrupt	SVC	0x08	lr	6
Prefetch Abort	ABT	0x0c	lr-4	5
Data Abort	ABT	0x10	lr-8	2
UNUSED	–	0x14	–	–
IRQ	IRQ	0x18	lr-4	4
FIQ	FIQ	0x1c	lr-4	3

- See ARM ARM ARMv7 documentation for details.
- Defaults to 0x000000, is SCTL.V is 1 “high-vector” 0xffff0000
- If security mode implemented more complex, separate vectors for secure/nonsecure, and on nonsecure the SCTL.V lets you set it anywhere via VBAR
- Interrupts: IRQ = general purpose hardware, FIQ = fast interrupt for really fast response (only 1), SWI = syscalls, talk to OS
- FIQ mode auto-saves r8-r12.

Complications

- What about thumb or endian mode when call into interrupt? Depends on flags in SCTLR register
- Stack pointer changes when handle interrupt (why?) Need to set that up in advance.

Ways to return from IRQ

- `subs pc,r14,#4`
  Sneakily branches and gets the right status register (due to S in SUBS)
- `sub r14,r14,#4`
  ...
  `movs pc,r14` (or `rfe`)
- Another stores lr and other things to stack, then restores
  ```
  sub r14,r14,#4
  stmbd r13!,{r0-r12,r14}
  ```

```
...
ldmfd r13!,{r0-r12,pc}^
```

The caret means to loast cpsr from spsr

Exclamation point means to update r13 after popping.

IRQ Handlers in C

In gcc for ARM, you can specify the interrupt type with an attribute. Automatically restores to right address.

```
void function () __attribute__ ((interrupt ("IRQ")));

/* Can be IRQ, FIQ, SWI, ABORT and UNDEF */

void __attribute__((interrupt("UNDEF"))) undefined_instruction_vector(void) {

    while(1) {
        /* Do Nothing */
    }
}
```