ECE 531/598 – Advanced Operating Systems
Lecture 7

Vince Weaver

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

vincent.weaver@maine.edu

19 September 2023

Announcements

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

HW#2 Review – Blinking

- Still grading this part, swapping SD card 32 times not fun while grading, got side-tracked trying to see if I could set up a bootloader menu or netboot to make this easier

HW#2 Review – Filesize

- Size: C about 200 bytes, assembly 68 bytes?
- Can look at .dis files for disassembly
- Init: C has 60 bytes to set things up, assembly has none
- Delay: C 64 bytes due to pessimization from volatile (has to load/store load/store i over and over) asm 12 bytes
- C also saves/restores LR and registers to maintain calling convention.

HW#2 Review – Other questions

- 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
  Another good final project
  Will we have a full OS? We'll have a minimal OS that runs, does some I/O, multi-task, run small C programs you write. Feel free to add more, in projects
  I have my own version vmwOS based on OS from this class, got stuck for a bit.

Include File Notes

- Including with “ ” versus <>

Writing printk

```
int printk(char *string,...) {

        va_list ap;
        va_start(ap, string);
    int x;

        while(1) {
                if (*string==0) break;

                if (*string=='%') {
                        string++;
                        if (*string=='d') {
                                string++;
                                x=va_arg(ap, int);
```

Integer to String Conversion

This it the algorithm I use, there are other ways to do it that don't involve the backwards step (starting off by dividing by 1 billion and dividing the divisor by 10 each time).

- Repeatedly divide by 10.
- Digit is the remainder. Repeat until quotient 0.
- Make sure handle 0 case.
- Convert each digit to ASCII by adding 48 ('0')
- Why does the number end up backwards?

Division by 10

- ARM1176 in Pi has no divide routine, why isn't this a problem?
  We are on ARMv7/v8 which does, but for backwards compatibility we are compiling to ARMv6.
- Generic x=y/z division is not possible without fancy work (iterative subtraction? Newton approximation?)
- Dividing by a constant is easier
- C compiler cheats, for /10 it effectively multiplies by 1/10.

- Look at generated assembly, you'll see it multiply by `0x66666667`
- Why is it not a problem when dividing by 16?
- What does the C compiler do if you do divide by a non-constant? Makes a call to C-library or gcc-library divide routine, which we don't link in.
- If on ARM 1176 you try to use division, C compiler will try to call something like `__aeabi_uidiv()` which you have to provide.
  Can be fancy assembly, or just iterative subtraction

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.
- Linux will avoid network interrupts when busy, or timer interrupts if trying to sleep.

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 (often need to do this two ways)
  - 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.
- Note ARM32 on Cortex-A (this class) can be different than Cortex-M (like the STM32 boards in 271)
- Possibly also different in ARM64

How to find out?

- ARM ARM for ARMv7 (2700+ pages)
- Look at Linux source code
- Look at Raspberry Pi Forums
- Note Pi4 has extra gic-400 interrupt controller you need to enable

ARM has various Modes

- Modes:
- States
  - ISA: ARM (normal), Thumb, Jazelle, ThumbEE
  - Execution state (?)
  - Security: Secure and Non-secure
- Privilege 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
UND	PL1	undefined instruction
SYS	PL1	system
HYP	PL2	hypervisor (only if virtual extensions)

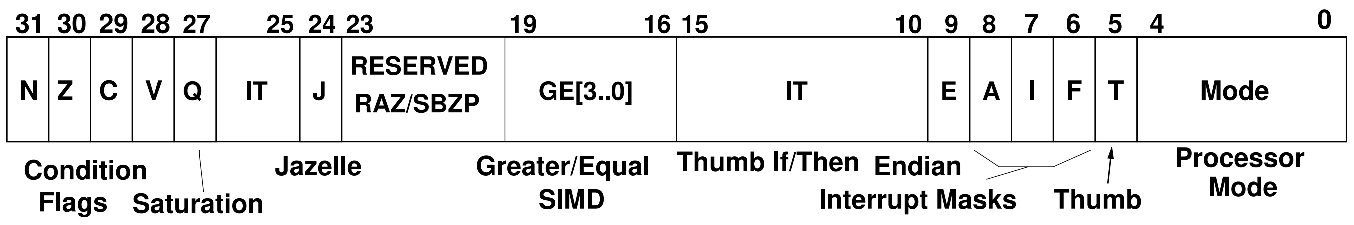
ARM Modes – continued

- User mode – unprivileged, 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



- 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