ECE 598 – Advanced Operating Systems
Lecture 7

Vince Weaver

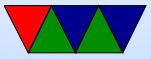
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

5 February 2015

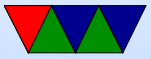
Announcements

- Homework #2 was assigned, due Wednesday
- Don't put it off until the last minute!



UART speed setting Example

- Calculate for 14.4kb/s
- Divider $= \frac{BaseFrequency}{16 \times Desired}$
- Divider $= \frac{3000000}{16 \times 14400} = 13.020$
- Integer part $=$ Integer part $= 13$.
  Fractional part $= (.020 \times 64) + 0.5 = 1.78$ so 1 or 2.
- `mmio_write(UART0_IBRD, 13);`
  `mmio_write(UART0_FBRD, 1);`

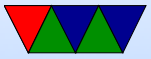


Scanning ATAGS

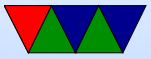
- List of variables passed by bootloader. A standard. See:

  `http://www.simtec.co.uk/products/SWLINUX/files/booting_article.html`

- We mostly care about getting memory size.
- Size, Tag-type, additional
- Located traditionally at 0x100 but you should really check r3 for addr.
- Starts with ATAG_CORE



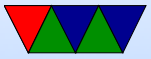
- Ends with ATAG_NONE
- We wants ATAG_MEM and maybe ATAG_CMDLINE on Raspberry Pi.
- Format is SIZE, TYPE, DATA0 ... DATAN. Then repeat.



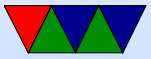
Integer to String Conversion

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 1billion and dividing the divisior by 10 each time).

- Repeatedly divide by 10.
- Digit is the remainder. Repeat until quotient 0.
- Make sure handle 0 case.

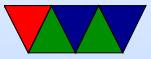


- Convert each digit to ASCII by adding 48 ('0')
- Why does the number end of backwards?

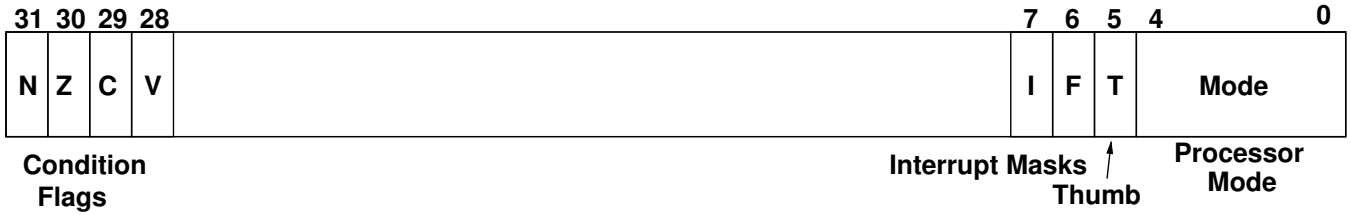


Exceptions and Interrupts

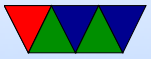
- All architectures are different
- ARM does it a little differently from others.



ARM CPSR Register

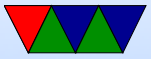


- Current Program Status Register
- There are seven processor modes
  six privledged: abort, fast interrupt, interrupt, supervisor, system, undefined
  one nonprivledged: user
- unprivledged cannot write CPSR



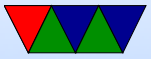
ARM Interrupt Registers

User/Sys	Fast	IRQ	Supervisor	Undefined	Abort
r0 r1 r2 r3 r4 r5 r6 r7					
r8 r9 r10 r11 r12	r8_fiq r9_fiq r10_fiq r11_fiq r12_fiq				
r13/sp r14/lr r15/pc	r13_fiq r14_fiq	r13_irq r13_irq	r13_svc r14_svc	r13_undef r14_undef	r13_abt r14_abt
cpsr	spsr_fiq	spsr_irq	spsr_svc	spsrc_undef	spsr_abt



ARM Interrupt Handling

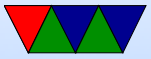
- ARM core saves CPSR to the proper SPSR
- ARM core saves PC to the banked LR
- ARM core sets CPSR to exception mode (disables interrupts)
- ARM core sets PC to the exception handler



Vector Table

Type	Type	Offset	LR
Reset	SVC	0x0	–
Undefined Instruction	UND	0x04	lr
Software Interrupt	SVC	0x08	lr
Prefetch Abort	ABT	0x0c	lr-4
Data Abort	ABT	0x10	lr-8
IRQ	IRQ	0x18	lr-4
FIQ	FIQ	0x1c	lr-4

- See ARM ARM ARMv6 documentation for details.

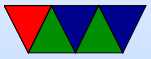


- Defaults to 0x000000. On some ARM you can move to any 32-byte aligned address. Why?

- Return from IRQ `subs pc,r14,#4`
  Sneakily branches and gets the right status register (due to S in SUBS)

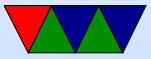
- Interrupts: IRQ = general purpose hardware,
  FIQ = fast interrupt for really fast response (only 1),
  SWI = syscalls, talk to OS

- FIQ mode auto-saves r8-r12.



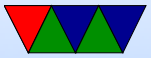
- Different stacks? IRQ mode, SVC mode (boots into), user-mode stack

- Nested vs non-nested.
  Nested = better realtime. As soon as possible re-enable interrupts. Complex though.



Raspberry Pi Interrupts

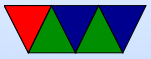
- Two types: GPU and ARM
- ARM: one timer, one mailbox, two doorbells, two GPU halted, two address errors.
- Can pick what gets set to FIQ interrupt.
- No priority. If one is more important, use FIQ.



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 */
    }
}
```

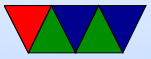


Exceptions and Interrupts

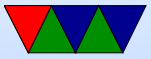
- How do we get the vector to 0x0?
  Copy it there after the fact. Hard part is if we want the routines to be C code.

- Clever, have the reset vector point to start of code, so you can have the reset vector of beginning of code and it will jump to the right location.

```
_start:
    ldr pc, reset_addr
    ldr pc, undefined_addr
    ldr pc, software_interrupt_addr
    ldr pc, prefetch_abort_addr
```

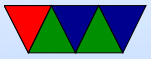


```
    ldr pc, data_abort_addr
    ldr pc, unused_addr
    ldr pc, interrupt_addr
    ldr pc, fast_interrupt_addr
reset_addr:             .word   reset
undefined_addr:         .word   undefined_instruction
software_interrupt_addr:    .word   software_interrupt
prefetch_abort_addr:        .word   prefetch_abort
data_abort_addr:        .word   data_abort
unused_addr:            .word   reset
interrupt_addr:         .word   interrupt
fast_interrupt_addr:        .word   fast_interrupt

_start:
    ...
reset:
    ldr r3, =_start
    mov     r4, #0x0000
    ldmia   r3!,{r5, r6, r7, r8, r9, r10, r11, r12}
    stmia   r4!,{r5, r6, r7, r8, r9, r10, r11, r12}
    ldmia   r3!,{r5, r6, r7, r8, r9, r10, r11, r12}
    stmia   r4!,{r5, r6, r7, r8, r9, r10, r11, r12}
```



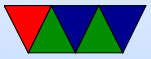
Clearing the Interrupt Status Bit

```
_enable_interrupts:
    mrs     r0, cpsr
    bic     r0, r0, #0x80
    msr     cpsr_c, r0

    mov     pc, lr
```



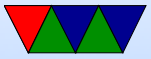
Interrupt Sources

- Section 7 of peripheral manual



Sample Interrupt Handler

```
void __attribute__((interrupt("IRQ"))) interrupt_vector(void) {
    static int lit = 0;

    /* Clear the ARM Timer interrupt */
    /* We assume it's the only interrupt enabled */
    mmio_write(TIMER_IRQ_CLEAR,0x1);

    /* Flip the LED */
    if( lit ) { led_off(); lit=0; }
    else {led_on(); lit=1; }
}
```



Timer Interrupt

- Section 14 of peripheral manual.

```
int timer_init(void) {

        mmio_write(IRQ_ENABLE_BASIC_IRQ,IRQ_ENABLE_BASIC_IRQ_ARM_TIMER);

        /* Timer frequency = Clk/256 * 0x400 */
        mmio_write(TIMER_LOAD,0x400);

        /* Setup the ARM Timer */
        mmio_write(TIMER_CONTROL,
                TIMER_CONTROL_32BIT | /* typo 23 */
                TIMER_CONTROL_ENABLE |
                TIMER_CONTROL_INT_ENABLE |
                TIMER_CONTROL_PRESCALE_256);

    /* Enable interrupts! */
//     _enable_interrupts();
}
```