Lab 12, interrupts, part 1

By ADMIN | *Published:* APRIL 13, 2013

Time to create content of **io.h** with references to control registers, irq nums etc:

```
#define IOBASE          0x20000000      /* base of io regs */
#define INTREGS         (IOBASE+0x00B200)
#define POWERREGS       (IOBASE+0x100000)
#define PL011REGS       (IOBASE+0x201000)

#define UART_PL01x_FR_TXFF  0x20

typedef struct Intregs Intregs;

/* interrupt control registers */
struct Intregs {
    u32int  ARMpending;
    u32int  GPUpending[2];
    u32int  FIQctl;
    u32int  GPUenable[2];
    u32int  ARMenable;
    u32int  GPUdisable[2];
    u32int  ARMdisable;
};

enum {
    IRQtimer0   = 0,
    IRQtimer1   = 1,
    IRQtimer2   = 2,
    IRQtimer3   = 3,
    IRQclock    = IRQtimer3,
    IRQdma0     = 16,
#define IRQDMA(chan)    (IRQdma0+(chan))
    IRQaux      = 29,
    IRQmmc      = 62,
    IRQbasic    = 64,
    IRQtimerArm = IRQbasic + 0,
    DmaD2M      = 0,        /* device to memory */
    DmaM2D      = 1,        /* memory to device */
    DmaM2M      = 2,        /* memory to memory */
    DmaChanEmmc = 4,        /* can only use 2-5, 11-12 */
    DmaDevEmmc  = 11
};
```

Then in Mach struct we need to add stacks – they will be used as short memory blocks for stack

```
struct Mach {
        ...
    /* stacks for exceptions */
    ulong   fiqstack[4];
    ulong   irqstack[4];
    ulong   abtstack[4];
    ulong   undstack[4];
    int stack[1];
};
```

Also *Mach.stack* reference at the end of Mach will be very useful for us because it is indication of low limit of kernel stack. If kernel stack address moves below this we detect that it is “kernel panic” situation – that will be performed in interrupts coding.

Categories

- Blog
- Boost
- C++
- Cryptography
- Embedding
- Hybrids
- Inferno OS
- MacAppStore
- Misc
- Models
- Projects
- PyQt
- PySide
- Qt
- QtSpeech
- Raspberry Pi
- Research
- Ru
- TogMeg
- Trac
- TTS
- Tutorial
- Undo
- Web

Then, because we will work with assembler codes and can easily get unpredictable cases, let's create **dump.c** with functions that will help us with dumping all registers, stack, memory fragments, etc:

```
#include "u.h"
#include "../port/lib.h"
#include "mem.h"
#include "dat.h"
#include "ureg.h"
#include "armv6.h"

void
dumplongs(char *msg, ulong *v, int n)
{
    int i, l;

    l = 0;
    iprint("%s at %.8p: ", msg, v);
    for(i=0; i<n; i++){
        if(l >= 4){
            iprint("\n    %.8p: ", v);
            l = 0;
        }
        if(isvalid_va(v)){
            iprint(" %.8lux", *v++);
            l++;
        }else{
            iprint(" invalid");
            break;
        }
    }
    iprint("\n");
}

static void
_dumpstack(Ureg *ureg)
{
    ulong *v, *l;
    ulong inst;
    ulong *estack;
    int i;

    l = (ulong*)(ureg+1);
    if(!isvalid_wa(l)){
        iprint("invalid ureg/stack: %.8p\n", l);
        return;
    }
    print("ktrace /kernel/path %.8ux %.8ux %.8ux\n"
        ,ureg->pc, ureg->sp, ureg->r14);
    if(up != nil && l >= (ulong*)up->kstack
        && l <= (ulong*)(up->kstack+KSTACK-4))
        estack = (ulong*)(up->kstack+KSTACK);
    else if(l >= (ulong*)m->stack && l <= (ulong*)((ulong)m+BY2PG-4))
        estack = (ulong*)((ulong)m+BY2PG-4);
    else{
        iprint("unknown stack\n");
        return;
    }
    i = 0;
    for(; l<estack; l++) {
        if(!isvalid_wa(l)) {
            iprint("invalid(%8.8p)", l);
            break;
        }
        v = (ulong*)*l;
        if(isvalid_wa(v)) {
            inst = v[-1];
            if((inst & 0x0ff0f000) == 0x0280f000 &&
                 (*(v-2) & 0x0fff000) == 0x028fe000    ||
                (inst & 0x0f000000) == 0x0b000000) {
                iprint("%8.8p=%8.8lux ", l, v);
                i++;
            }
        }
        if(i == 4){
            iprint("\n");
            i = 0;
        }
    }
    if(i)
        print("\n");
}

```

```
/*
 * Fill in enough of Ureg to get a stack trace, and call a function.
 * Used by debugging interface rdb.
 */
void
callwithureg(void (*fn)(Ureg*))
{
    Ureg ureg;
    ureg.pc = getcallerpc(&fn);
    ureg.sp = (ulong)&fn;
    ureg.r14 = 0;
    fn(&ureg);
}

void
dumpstack(void)
{
    callwithureg(_dumpstack);
}

void
dumparound(uint addr)
{
    uint addr0 = (addr/16)*16;
    int a_row, a_col;
    uchar ch, *cha;
    uint c;
    /* +-32 bytes to print */
    print("%8.8uX:\n", addr0 +(-2)*16);
    for (a_col = 0; a_col<16; ++a_col) {
        print(" %.2uX", a_col);
    }
    print("\n");

    for (a_row = -2; a_row < 3; ++a_row) {
        for (a_col = 0; a_col<16; ++a_col) {
            cha = (uchar *)(addr0 +a_row*16+a_col);
            ch = *cha;
            c = ch;
            if (cha == (uchar *)addr)
                print(">%2.2uX", c);
            else print(" %2.2uX", c);
        }
        print("\n");
    }
    print("\n");
}

void
dumpregs(Ureg* ureg)
{
    print("TRAP: %s", trapname(ureg->type));
    if((ureg->psr & PsrMask) != PsrMsvc)
        print(" in %s", trapname(ureg->psr));
    print("\n");
    print("PSR %8.8uX type %2.2uX PC %8.8uX LINK %8.8uX\n",
        ureg->psr, ureg->type, ureg->pc, ureg->link);
    print("R14 %8.8uX R13 %8.8uX R12 %8.8uX R11 %8.8uX R10 %8.8uX\n",
        ureg->r14, ureg->r13, ureg->r12, ureg->r11, ureg->r10);
    print("R9  %8.8uX R8  %8.8uX R7  %8.8uX R6  %8.8uX R5  %8.8uX\n",
        ureg->r9, ureg->r8, ureg->r7, ureg->r6, ureg->r5);
    print("R4  %8.8uX R3  %8.8uX R2  %8.8uX R1  %8.8uX R0  %8.8uX\n",
        ureg->r4, ureg->r3, ureg->r2, ureg->r1, ureg->r0);
    print("Stack is at: %8.8luX\n", ureg);
    print("PC %8.8lux LINK %8.8lux\n", (ulong)ureg->pc, (ulong)ureg->link);

    if(up)
        print("Process stack:  %8.8lux-%8.8lux\n",
            up->kstack, up->kstack+KSTACK-4);
    else
        print("System stack: %8.8lux-%8.8lux\n",
            (ulong)(m+1), (ulong)m+BY2PG-4);
    dumplongs("stack", (ulong *)(ureg + 1), 16);
    _dumpstack(ureg);
    dumparound(ureg->pc);
}
```

So, nothing complicated, just *print()* mostly

FILES:

io.h

dat.h

fns.h
main.c
dump.c
trap.c
pl011.c
mkfile

This entry was posted in *Blog*, *Inferno OS*, *Raspberry Pi*, *Research*. Bookmark the *permalink*. *Post a comment* or leave a trackback: *Trackback URL*.

« *Lab 11, _div, testing print*

Lab 13, interrupts, part 2 »