# Lab 9, coding assembler part

*By* ADMIN | *Published:* JANUARY 27, 2013

Time to have assembler part of Inferno kernel to be implemented. Let's start with routines that allows to make labels and later jump to them, they are used in kernel sources to have scheduler to switch control and do context switching between processes.
The **Label** structure in **dat.h**:

```
struct Label {
    ulong   sp;
    ulong   pc;
};
```

Simple idea to remember the PC(program counter) and SP(stack pointer), to init the structure and use later, we should have two functions implemented:

```
TEXT setlabel(SB), $-4
    MOVW    R13, 0(R0)
    MOVW    R14, 4(R0)
    MOVW    $0, R0
    RET

TEXT gotolabel(SB), $-4
    MOVW    0(R0), R13
    MOVW    4(R0), R14
    MOVW    $1, R0
    RET
```

It works in simple way, **setlabel()** has pointer to Label as argument (R0 registry), so initialization 0(R0) puts value of R13/SP into Label.sp, and 4(R0) puts value of R14/PC into Label.pc; **gotolabel()** does opposite, taking pointer to Label and initialize SP and PC and as result pass control of execution.

**getcallerpc()** also is very simple, when the function is called, the PC is put in stack for return address, so we can extract it as 0(SP):

```
TEXT getcallerpc(SB), $-4
    MOVW    0(SP), R0
    RET
```

**splhi()** is function that saves PC into Mach structure, disables all maskable interrupts and returns the previous interrupt enable state, see the http://www.vitanuova.com/inferno/man/10/splhi.html, so we modify Mach structure to have *ulong splpc* at very beginning so address is same as address of Mach.

```
struct Mach {
    ulong   splpc;      /* pc of last caller to splhi */
    int     machno;     /* physical id of processor */
    ulong   ticks;      /* of the clock since boot time */
    Proc*   proc;       /* current process on this processor */
    Label   sched;      /* scheduler wakeup */
};
```

So the code of **int splhi()**:

```
TEXT splhi(SB), $-4
    MOVW    $(MACHADDR), R6
    MOVW    R14, (R6)    /* m->splpc */
```

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

```
    MOVW    R14, (R6)   /* m->splpc */
    MOVW    CPSR, R0
    ORR     $(PsrDirq), R0, R1
    MOVW    R1, CPSR
    RET
```

**int spllo()** enables interrupts and returns a flag representing the previous interrupt enable state:

```
TEXT spllo(SB), $-4
    MOVW    CPSR, R0
    BIC     $(PsrDirq|PsrDfiq), R0, R1
    MOVW    R1, CPSR
    RET
```

**splx(int x)** saves *PC* into *m->splpc*, restores the interrupt enable state state to x, which must be a value returned by a previous call to splhi or spllo
**splxpc(int x)** does same but without saving *PC*

```
TEXT splx(SB), $-4
    MOVW    $(MACHADDR), R6
    MOVW    R14, (R6)   /* m->splpc */
TEXT splxpc(SB), $-4
    MOVW    R0, R1
    MOVW    CPSR, R0
    MOVW    R1, CPSR
    RET
```

**int islo()** returns true (non-zero) if interrupts are currently enabled, and 0 otherwise:

```
TEXT islo(SB), $-4
    MOVW    CPSR, R0
    AND     $(PsrDirq), R0
    EOR     $(PsrDirq), R0
    RET
```

**FILES:**
dat.h
armv6.h
armv6.s
main.c
splhi.pdf

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

« *Lab 8, memory model*

*Lab 10, Bss, memory pools, malloc* »