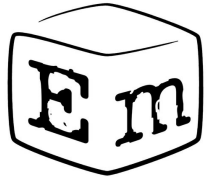


Поддержка архитектуры Эльбрус в ОСРВ Embox

Антон Бондарев

OS Day, 23 июня 2022

Embox

- Свободная операционная система для встроенных систем
- Поддерживает архитектуру Эльбрус
- Соглашение с МЦСТ об открытости Embox

Embox и МЦСТ

- Стороны соглашаются, что исходный код ОСРВ Embox, в том числе код поддержки архитектуры Эльбрус, остается открытым в соответствии с его лицензией, и может быть изучен, модифицирован и запущен на платформах на базе процессоров с архитектурой Эльбрус.
- Стороны стремятся к улучшению поддержки архитектуры Эльбрус в открытом проекте Embox, публикуют собственный исходный код и не препятствуют публикации исходного кода сторонними лицами, если это не противоречит другим соглашениям или договорам.
- Стороны допускают использование кода ОСРВ Embox, в том числе код поддержки архитектуры Эльбрус, как учебного пособия для практического освоения архитектуры Эльбрус, в том числе специфичных для ядра ОС частей.

Embox и МЦСТ

- Стороны способствуют популяризации и распространению процессорной архитектуры Эльбрус путем предоставления в публичный доступ информации о данной архитектуре, а также исходного кода для данной архитектуры, распространяемого под свободными лицензиями.
- Стороны способствуют проведению мероприятий по популяризации архитектуры Эльбрус и стремятся участвовать в подобных мероприятиях.

Агенда

- Доступ к регистрам
- Регистровый файл (плавающие окна)
- Стеки
- Процедуры setjmp/longjmp, context_switch

Доступ к регистрам

- Регистры доступные через специальные команды RWS/RRS
- Регистры доступные через другое адресное пространство

Доступ к регистрам

```
static inline void e2k_upsr_write(uint32_t val) {
    asm volatile ("rws %0, %%upsr" : : "ri"(val));
}

static inline uint32_t e2k_pfpfr_read(void) {
    uint32_t pfpfr;
    asm volatile ("rrs %%pfpfr, %0" : "=r"(pfpfr) :);
    return pfpfr;
}
```

Доступ к регистрам

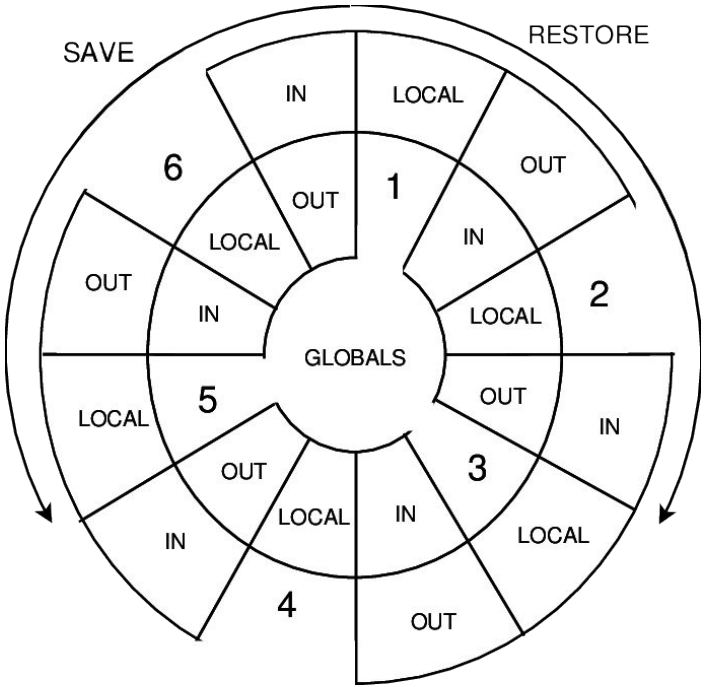
- Регистры доступные через другое адресное пространство
- команды LD/ST с модификаторами доступа ‘b’, ‘h’, ‘w’, ‘d’
- Можно задавать адресное пространство
- Можно задавать канал (не очень понятно, что это:))

Доступ к регистрам

```
#define _E2K_READ_MAS(addr, mas, type, size_letter, chan_letter) \
({ \
    register type res; \
    asm volatile ("ld" #size_letter "," #chan_letter " \t0x0, [%1] %2, %0" \
        : "=r" (res) \
        : "r" ((uintptr_t) (addr)), \
          "i" (mas)); \
    res; \
})

#define _E2K_WRITE_MAS(addr, val, mas, type, size_letter, chan_letter) \
({ \
    asm volatile ("st" #size_letter "," #chan_letter " \t0x0, [%0] %2, %1" \
        : \
        : "r" ((uintptr_t) (addr)), \
          "r" ((type) (val)), \
          "i" (mas) \
        : "memory"); \
})
```

Плавающие окна

- Метод расширения количества доступных регистров
- В SPARC архитектуре есть вращающиеся окна фиксированного размера

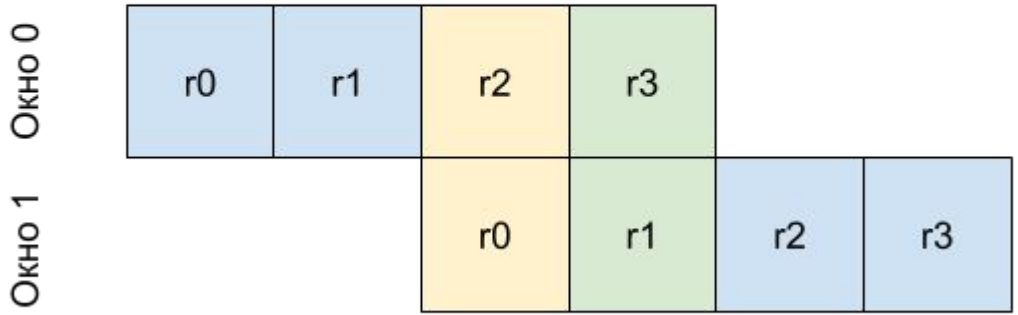


Плавающие окна

- Окно перемещается по регистровому файлу при вызове процедуры или возврате из нее

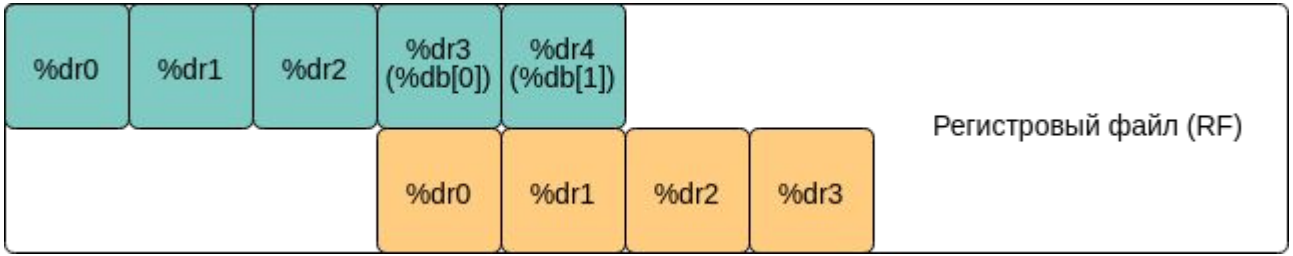
Плавающие окна

- Входные регистры из вышестоящей процедуры видны в текущей как выходные



Плавающие окна

- В Эльбрусе можно задавать размер окна команда
  - setwd wsz=0x10
  - wsz -размер в 128 битных регистрах
- И задать вращающееся окно
  - setbn rbs = 0x3, rsz = 0x8
  - rbs задает размер вращаемого окна
  - rsz — базовый адрес, относительно текущего регистрового окна

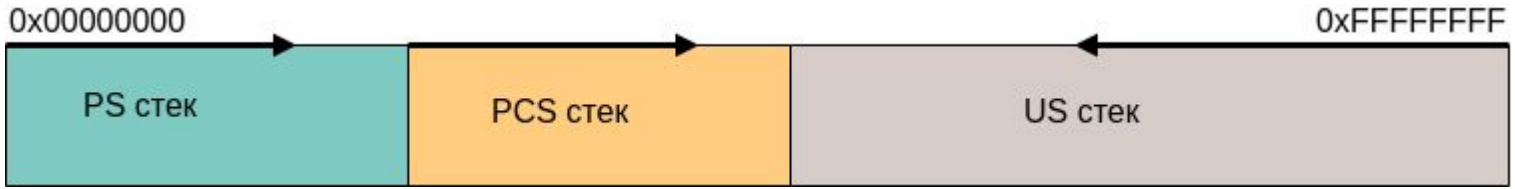


Стеки

- Стек процедур (Procedure Stack — PS)
- Стек связующей информации (Procedure Chain Stack — PCS)
- Стек пользователя (User Stack — US)

Стеки

- Два растут вниз, один вверх



Стеки

- Стек процедур (PS) предназначен для данных, вынесенных на “оперативные” регистры.
- Аргументы и другие регистры общего назначения из регистрового окна

Стеки

- Стек связующей информации (PCS) предназначен для размещения информации о предыдущей (вызвавшей) процедуре и используемой при возврате.
- Свой регистровый файл с откачкой подкачкой

Стеки

- Оба эти стека (PS и PCS) характеризуются базовым адресом, размером и текущим смещением. Эти параметры задаются в регистрах PSP и PCSP, они 128-битные и в ассемблере нужно обращаться к конкретным полям (например high или low).
- Растут вверх

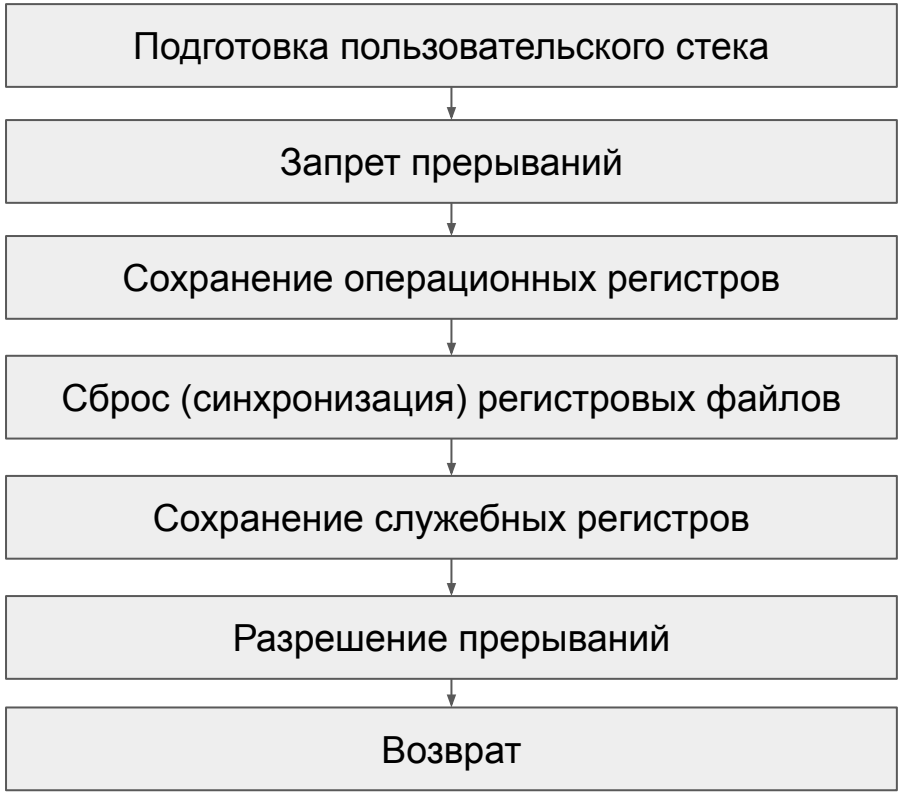
Стеки

- Пользовательский стек обычный стек (локальные переменные), но туда не заносится информация о регистрах и адресах возврата
- Также задается базовым адресом и размером.
- Текущий указатель находится в регистре USD.lo

SETJMP/LONGJMP

- Работа в кернел моде
- Нужно восстановить несколько стеков и учесть подкачки окон
- Регистры CR0, CR1, PSP, PCSP, USD
- При восстановлении стеков нужно запрещать прерывания (подкачка окон должна быть атомарной)

SETJMP


Подготовка пользовательского стека
Запрет прерываний
Сохранение операционных регистров
Сброс (синхронизация) регистровых файлов
Сохранение служебных регистров
Разрешение прерываний
Возврат

Стек связующей информации (PCS)

```
.type update_pcsp_ind,@function

$update_pcsp_ind:

     setwd wsz = 0x4, nfx = 0x0

     /* Here and below, 10 is size of PCSHTP.ind. Here we

     * extend the sign of PCSHTP.ind */

     shld %dr1, (64 - 10), %dr1

     shrd %dr1, (64 - 10), %dr1

     /* Finally, PCSP.ind += PCSHTP.ind */

     addd %dr1, %dr0, %dr0


     E2K_ASM_RETURN
```

Процедурный стек (PSP)

```
/* First arg is PSP, 2nd arg is PSHTP
 * Returns new PSP value with updated PSP.ind
 */
.type update_psp_ind,@function
$update_psp_ind:
     setwd wsz = 0x4, nfx = 0x0

     /* Here and below, 12 is size of PSHTP.ind. Here we
     * extend the sign of PSHTP.ind as stated in documentation */
     shld %dr1, (64 - 12), %dr1
     shrd %dr1, (64 - 12), %dr1
     muld %dr1, 2, %dr1

     /* Finally, PSP.ind += PSHTP.ind */
     addd %dr1, %dr0, %dr0

     E2K_ASM_RETURN
```

SETJMP

```
C_ENTRY(setjmp):
     setwd wsz = 0x14, nfx = 0x0
     /* It's for db[N] registers */
     setbn rsz = 0x3, rbs = 0x10, rcur = 0x0

     /* We must disable interrupts here */
     disp %ctpr1, ipl_save
     ipd  3
     call %ctpr1, wbs = 0x10
     /* Store current IPL to dr9 */
     addd 0, %db[0], %dr9

     /* Store some registers to jmp_buf */
     rrd %cr0.hi, %dr1
     rrd %cr1.lo, %dr2
     rrd %cr1.hi, %dr3
     rrd %usd.lo, %dr4
     rrd %usd.hi, %dr5
```

SETJMP

```
/* Prepare RF stack to flush in longjmp */
rrd %psp.hi, %dr6
rrd %pshtp,  %dr7
addd 0, %dr6, %db[0]
addd 0, %dr7, %db[1]
disp %ctpr1, update_psp_ind
ipd  3
call %ctpr1, wbs = 0x10
addd 0, %db[0], %dr6

/* Prepare CF stack to flush in longjmp */
rrd %pcsp.hi, %dr7
rrd %pcshtp,  %dr8
addd 0, %dr7, %db[0]
addd 0, %dr8, %db[1]
disp %ctpr1, update_pcsp_ind
ipd  3
call %ctpr1, wbs = 0x10
addd 0, %db[0], %dr7
```

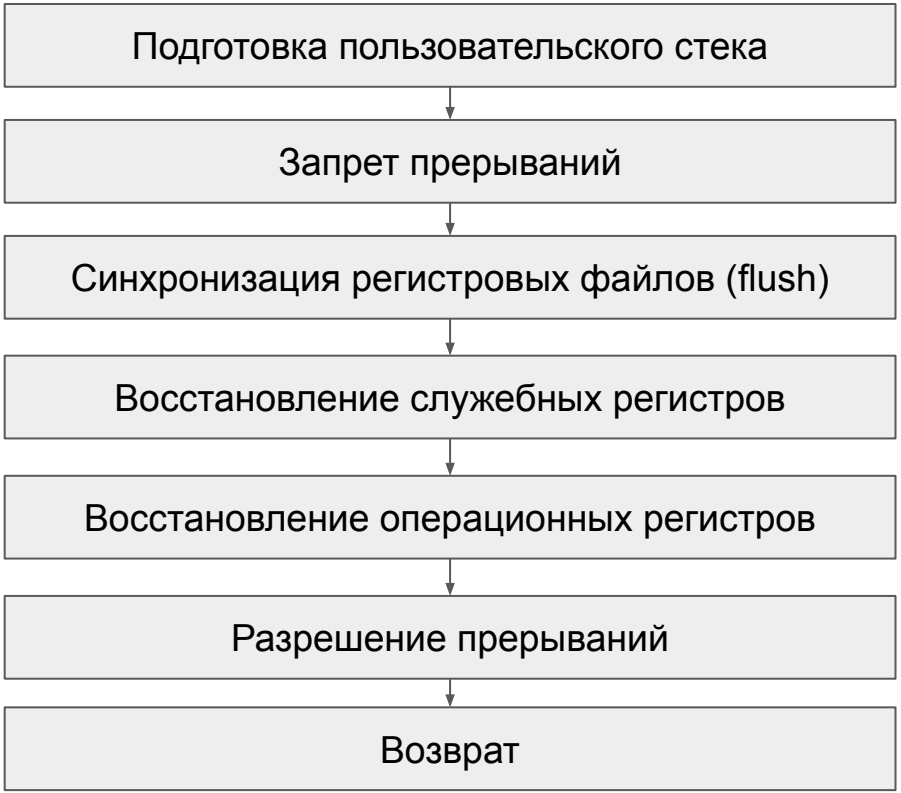
SETJMP

```
/* Prepare RF stack to flush in longjmp */
rrd %psp.hi, %dr6
rrd %pshtp,  %dr7
addd 0, %dr6, %db[0]
addd 0, %dr7, %db[1]
disp %ctpr1, update_psp_ind
ipd  3
call %ctpr1, wbs = 0x10
addd 0, %db[0], %dr6

/* Prepare CF stack to flush in longjmp */
rrd %pcsp.hi, %dr7
rrd %pcshtp,  %dr8
addd 0, %dr7, %db[0]
addd 0, %dr8, %db[1]
disp %ctpr1, update_pcsp_ind
ipd  3
call %ctpr1, wbs = 0x10
addd 0, %db[0], %dr7
```

SETJMP

```
std %dr1, [%dr0 + E2K_JMBBUFF_CR0_HI]
std %dr2, [%dr0 + E2K_JMBBUFF_CR1_LO]
std %dr3, [%dr0 + E2K_JMBBUFF_CR1_HI]
std %dr4, [%dr0 + E2K_JMBBUFF_USD_LO]
std %dr5, [%dr0 + E2K_JMBBUFF_USD_HI]
std %dr6, [%dr0 + E2K_JMBBUFF_PSP_HI]
std %dr7, [%dr0 + E2K_JMBBUFF_PCSP_HI]

/* Enable interrupts */
addd 0, %dr9, %db[0]
disp %ctpr1, ipl_restore
ipd  3
call %ctpr1, wbs = 0x10

/* return 0 */
adds 0, 0, %r0
E2K_ASM_RETURN
```

LONGJMP


Подготовка пользовательского стека
Запрет прерываний
Синхронизация регистровых файлов (flush)
Восстановление служебных регистров
Восстановление операционных регистров
Разрешение прерываний
Возврат

LONGJMP

```
C_ENTRY(longjmp):
setwd wsz = 0x14, nfx = 0x0
setbn rsz = 0x3, rbs = 0x10, rcur = 0x0

/* We must disable interrupts here */
disp %ctpr1, ipl_save
ipd  3
call %ctpr1, wbs = 0x10
/* Store current IPL to dr9 */
addd 0, %db[0], %dr9

/* We have to flush both RF and CF to memory because saved values
* of P[C]SHTP can be not valid here. */
E2K_ASM_FLUSH_CPU
```

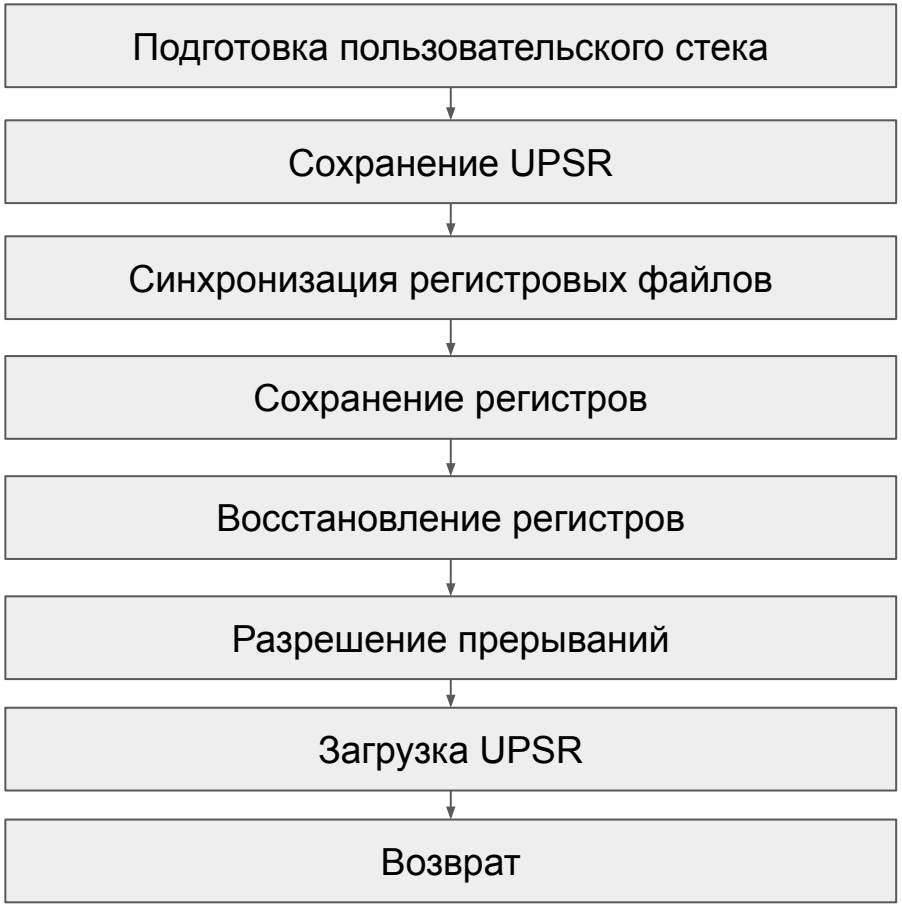
LONGJMP

```
/* Load registers previously saved in setjmp. */
ldd [%dr0 + E2K_JMBBUFF_CR0_HI], %dr2
ldd [%dr0 + E2K_JMBBUFF_CR1_LO], %dr3
ldd [%dr0 + E2K_JMBBUFF_CR1_HI], %dr4
ldd [%dr0 + E2K_JMBBUFF_USD_LO], %dr5
ldd [%dr0 + E2K_JMBBUFF_USD_HI], %dr6
ldd [%dr0 + E2K_JMBBUFF_PSP_HI], %dr7
ldd [%dr0 + E2K_JMBBUFF_PCSP_HI], %dr8

rwd %dr2, %cr0.hi
rwd %dr3, %cr1.lo
rwd %dr4, %cr1.hi
rwd %dr5, %usd.lo
rwd %dr6, %usd.hi
rwd %dr7, %psp.hi
rwd %dr8, %pcsp.hi
```

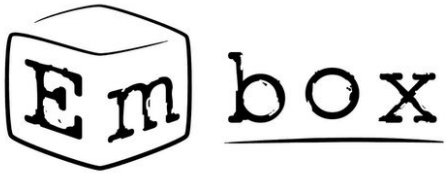
LONGJMP

```
/* Enable interrupts */
addd 0, %dr9, %db[0]
disp %ctpr1, ipl_restore
ipd  3
call %ctpr1, wbs = 0x10

/* Actually, we return to setjmp caller with second
* argument of longjmp stored on r1 register. */
adds 0, %r1, %r0
E2K_ASM_RETURN
```

context_switch()



Contacts



Essential toolbox for embedded development

Embox Project Homepage

- **http://embox.github.io/**

Anton Bondarev

- **anton.bondarev2310@gmail.com**