

# Other Global Variables

## objectTab

### Declaration

```
//file core/kernel/objdir.c
const struct object *objectTab[OBJ_NO_CLASSES]={[0 ... OBJ_NO_CLASSES-1] = 0};

typedef xm_s32_t (*readObjOp_t)(xmObjDesc_t, void *, xmSize_t, xm_u32_t *);
typedef xm_s32_t (*writeObjOp_t)(xmObjDesc_t, void *, xmSize_t, xm_u32_t *);
typedef xm_s32_t (*seekObjOp_t)(xmObjDesc_t, xmSize_t, xm_u32_t);
typedef xm_s32_t (*ctrlObjOp_t)(xmObjDesc_t, xm_u32_t, void *);
struct object {
    readObjOp_t Read;
    writeObjOp_t Write;
    seekObjOp_t Seek;
    ctrlObjOp_t Ctrl;
};
```

This is used for c object-oriented programming. Each object instance will assign their operations to these Read, Write, Seek, Ctrl fucntion pointers.

### Description

An array of pointers of XM objects. Objects here are sampling ports, queuing ports, health monitor, console, memory, etc.. It is easy to invoke an operation of a certain object by simply calling `objectTab[OBJ_CLASS_CONSOLE]->Read(...)`.

### Initialization

Initialized by assigned with object's address.

### Functions

1. ReadObjectSys & WriteObjectSys & SeekObjectSys & CtrlObjectSys  
Get object's offset on the array from object descriptor. Check if the object is not NULL and has the corresponding function.
2. OBJDESC\_GET\_CLASS

```
// Object descriptor:
// VALIDITY | CLASS | vCPUID | PARTITIONID | ID
// 1 | 7 | 4 | 10 | 10
return ((oD>>24)&OBJDESC_CLASS_MASK)
```

## \_\_nrCpus

### Declaration

```
//file core/kernel/setup.c
xm_u16_t __nrCpus = 0;
```

### Description

The number of real CPUs. Initially, it is set to 0. And it is updated by calling `SET_NRCPU((SparcGetNoCpus()<xmcTab.hpv.noCpus)?SparcGetNoCpus():xmcTab.hpv.noCpus)`, where `SparcGetNoCpus()` is implemented by:

```
xm_u8_t SparcGetNoCpus(void) {
    return (LoadIoReg(GET_PIC_BASE(0)+MPROC_STATUS_REG)>>28)+1;
}
```

### Initialization

Setup and SMP configuration

### Functions

1. GET\_NRCPU  
Usually used as the terminat condition of for-loop.
2. SET\_NRCPU  
This function is mentioned above

## contextTab

### Declaration

```
//file core/kernel/arch/head.S
ENTRY(contextTab)
    .zero CTXTTABSIZ
```

Description

Save threads' contexts. contextTab is updated when saving / setting up L1 page table.

Initialization

Filled up during \_start ENTRY in core/kernel/arch/head.S. First fill up l3, l2, l1 page table and copy l1 content to the memory location pointed by contextTab.

Functions

1. hypercall SparcWritePtdL1Sys
- This function is EMPTY
2. SetupPtdL1
- //file core/kernel/arch/vmmap.c
- First, CloneXMPtdL1; save \_pgTables content to ptdL1
- Second, update current guest's mmuCtxt, which is the current page table's backup
- Save page table backup into contextTab
3. LoadPartitionPageTable & SetMmuCtxt
- //file core/include/arch/processor.h
- Restore backuped page table when resetting kthreads and partitions
4. ASM arch/head.S
- load contextTab's physical address to a pointer.

pgTables[], ptdL1[], ptdL2[], ptdL3[]

Declaration

```
//file core/kernel/arch/head.S
.align 1024
ENTRY(_pgTables)
ENTRY(_ptdL1)
    .zero PTDL1SIZE = 1024
/*???
(16MB)
-----
4096 4KPAGES
1 L1
1 L2
64 L3
-----
*/
.align 256
ENTRY(_ptdL2)
    .zero NO_PTDL2_XMVMAP*PTDL2SIZE = 1 * 256

.align 256
ENTRY(_ptdL3)
    .zero NO_PTDL3_XMVMAP*PTDL3SIZE = 64 * 256
```

Description

SparcV8.pdf page 241. Level 1 size 1024, 256 enries. level 2 size 256, 64 entries. level 3 size 256, 64 entries. \_pgTables starts at the same location as ptdL1.

Initialization

Filled up during \_start ENTRY in core/kernel/arch/head.S.

Functions

1. CloneXMPtdL1

```
//write _pgTables -> ptdL1
WriteByPassMmuWord(&ptdL1[l1e], _pgTables[l1e]);
```

1. \_start
- //file core/kernel/arch/head.S
- Write ptdL3, store ptdL3 to ptdL2
- Write ptdL2, store into ptdL1 (CONFIG\_XM\_OFFSET and CONFIG\_XM\_LOAD\_ADDR)
- Write ptdL1 to contextTab entry
2. SetupVmMap
- Put hypervisor's physical memory into ptdL3.

And clean the frame.

## irqHandlerTab[CONFIG\_NO\_HWIRQS]

### Declaration

```
//file core/kernel/irqs.c
struct irqTabEntry irqHandlerTab[CONFIG_NO_HWIRQS];
```

```
struct irqTabEntry {
    irqHandler_t handler;
    void *data;
};
```

Type irq handler is given bellow. Irq handlers are assigned to this struct. A certain irq handler is triggered by invoking the irqTabEntry->handler that has the same index.

Usage:

```
if (irqHandlerTab[ctxt->irqNr].handler)
    (*(irqHandlerTab[ctxt->irqNr].handler))(ctxt, irqHandlerTab[ctxt->irqNr].data);
else
    DefaultIrqHandler(ctxt, 0);
```

### Description

This array contains `CONFIG_NO_HWIRQS` of `irqTabEntry`. The struct contains irq handler and pointer to data. irq handler is function of the following format:

```
typedef void (*irqHandler_t)(cpuCtxt_t *, void *);
```

### Initialization

Several irq's sets its handler and data by invoking function `SetIrqHandler`

```
//file core/kernel/arch/leon_timers.c
SetIrqHandler(TIMER_IRQ, TimerIrqHandler, 0);
SetIrqHandler(CLOCK_IRQ, ClockIrqHandler, 0);
//file core/kernel/arch/irqs.c SMP support
SetIrqHandler(HALT_ALL_IPI_VECTOR, SmpHaltAllHndl, 0);
SetIrqHandler(SCHED_PENDING_IPI_VECTOR, SmpSchedPendingIPIHndl, 0);
//file core/kernel/sched.c
SetIrqHandler(irqNr, SchedSyncHandler, 0);
```

### Functions

1. SetIrqHandler

2. DoIrq

Calling irq's corresponding handler.

3. SetupIrqs

Find each irq that has owner (a certain partition). Then the handler is set to `SetPartitionHwIrqPending`, which is used to set all running threads' flag for trigger irq.

And set trap handler to 0.

## trapHandlerTab[NO\_TRAPS]

### Declaration

```
//core/kernel/irqs.c
trapHandler_t trapHandlerTab[NO_TRAPS];
```

Comparing to irq handler entry, trap handler is just a function pointer type

```
typedef xm_s32_t (*trapHandler_t)(cpuCtxt_t *, xm_u16_t *);
```

### Description

Similar as above

### Initialization

```
SetTrapHandler(7, SparcFpFault);
SetTrapHandler(4, SparcTrapPageFault);
SetTrapHandler(1, SparcTrapPageFault);
SetTrapHandler(16, SparcTrapPageFault);
SetTrapHandler(15, SparcTrapPageFault);
```

### Functions

1. ArchSetupIrqs

2. SetTrapHandler
3. DoTrap
4. SetupIrqs

---

## hwIrqCtrl[CONFIG\_NO\_HWIRQS]

### Declaration

```
//file core/kernel/irqs.c
hwIrqCtrl_t hwIrqCtrl[CONFIG_NO_HWIRQS]
```

### Description

This array keeps the functions of every irq.

Another OOP:

```
typedef struct {
    void (*Enable)(xm_u32_t irq);
    void (*Disable)(xm_u32_t irq);
    void (*Ack)(xm_u32_t irq);
    void (*End)(xm_u32_t irq);
    void (*Force)(xm_u32_t irq);
    void (*Clear)(xm_u32_t irq);
} hwIrqCtrl_t;
```

### Initialization

//file core/kernel/arch/leon\_pic.c

function InitPic()

```
for (e=0; e<CONFIG_NO_HWIRQS; e++) {
    hwIrqCtrl[e].Enable=APicEnableIrq;
    hwIrqCtrl[e].Disable=APicDisableIrq;
    hwIrqCtrl[e].Ack=APicDisableIrq;
    hwIrqCtrl[e].End=APicEnableIrq;
    hwIrqCtrl[e].Force=APicForceIrq;
#ifdef CONFIG_LEON3
    hwIrqCtrl[e].Clear=APicClearIrq;
#endif
}
```

### Functions

1. HwIrqGetMask
  2. Hw Disable|Enable|Ack|End|Force|Clear Irq
- //file core/kernel/arch/leon\_pic.c operation ack == operation diable
3. InitPic

---

## \*trap2Str[]

### Declaration

```
//file core/kernel/arch/irqs.c
```

Put trap name to string in C.

```
xm_s8_t *trap2Str[]={
    _STR(DATA_STORE_ERROR), // 0
    _STR(INSTRUCTION_ACCESS_MMU_MISS), // 1
    _STR(INSTRUCTION_ACCESS_ERROR), // 2
    _STR(R_REGISTER_ACCESS_ERROR), // 3
    _STR(INSTRUCTION_ACCESS_EXCEPTION), // 4
    _STR(PRIVILEGED_INSTRUCTION), // 5
    _STR(ILLEGAL_INSTRUCTION), // 6
    _STR(FP_DISABLED), // 7
    _STR(CP_DISABLED), // 8
    _STR(UNIMPLEMENTED_FLUSH), // 9
    _STR(WATCHPOINT_DETECTED), // 10
    _STR(MEM_ADDRESS_NOT_ALIGNED), // 11
    _STR(FP_EXCEPTION), // 12
    _STR(CP_EXCEPTION), // 13
    _STR(DATA_ACCESS_ERROR), // 14
    _STR(DATA_ACCESS_MMU_MISS), // 15
    _STR(DATA_ACCESS_EXCEPTION), // 16
    _STR(TAG_OVERFLOW), // 17
    _STR(DIVISION_BY_ZERO), // 18
};
```

### Description

Used for debug information printing

Initialization

Functions

localCpuInfo

Declaration

```
//file core/kernel/setup.c
localCpu_t localCpuInfo[CONFIG_NO_CPUS];
```

Description

Array of localCpu\_t, contains flags, irqNestingCounter and globalIrqMask. Most used attribute is IrqMask.

Initialization

Allocate memory in CreateLocalInfo and set IrqMask to 0xffffffff. Setup at function ArchSetupIrqs for SMP support.

Functions

1. GET\_LOCAL\_CPU
- (&localCpuInfo[GET\_CPU\_ID()]) //SMP localCpuInfo //Not SMP
- GET\_CPU\_ID() = **GetCpuId()** = **cpuid>>28** asm volatile\_\_ ("rd %%asr17, %0\n\t" : "=r" (cpuid)); //Processor configuration registe PCR ars17; 31~28 is PI (Processor ID)
2. ArchSetupIrqs
3. CreatePartition
- Load localIrqMask from cpu global IrqMask and update it according to hwIrqTable.
- Assign updated local IrqMask to p->kThread[i]

cpuKhz

Declaration

```
//file core/include/guest.h
xm_u32_t cpuKhz;
```

Description

GPU's frequency

Initialization

//file core/kernel/arch/processor.c EarlySetupCpu

Functions

1. GetCpuKhz
2. EarlySetupCpu
3. InitPitClock
- Set clock, register and irq handler
4. SetupPct
- partCtrlTab->cpuKhz=cpuKhz

sysHwClock

Declaration

```
//file core/kernel/arch/leon_timers.c
hwClock_t *sysHwClock=&pitClock;
```

Hardware clock and its functions, shown as the Initialization section below.

```
typedef struct hwClock {
    char *name;
    xm_u32_t flags;
#define HWCLOCK_ENABLED (1<<0)
#define PER_CPU (1<<1)
    xm_u32_t freqKhz;
    xm_s32_t (*InitClock)(void);
    xmTime_t (*GetTimeUsec)(void);
    void (*ShutdownClock)(void);
} hwClock_t;
```

Description

System shared clock.

Initialization

Hardware clock is the pit clock:

```
static hwClock_t pitClock={
    .name="LEON clock",
    .flags=0,
    .InitClock=InitPitClock,
    .GetTimeUsec=ReadPitClock,
    .ShutdownClock=ShutdownPitClock,
};
```

Functions

- 1. GetSysClockUsec  
Return sysHwClock's usec
- 2. SetupSysClock  
Called at setup, after InitSche();  
Invoke InitPitClock() at file core/kernel/arch/leon\_timers.c

sysHwTimer

Declaration

```
//file core/include/ktimer.h
hwTimer_t *sysHwTimer;
```

This struct is an attribute of localTime\_t .

```
typedef struct hwTimer {
    xm_s8_t *name;
    xm_u32_t flags;
#define HWTIMER_ENABLED (1<<0)
    xm_u32_t freqKhz;
    xm_s32_t irq;
    xm_s32_t (*InitHwTimer)(void);
    void (*SetHwTimer)(xmTime_t);
    // This is the maximum value to be programmed
    xmTime_t (*GetMaxInterval)(void);
    xmTime_t (*GetMinInterval)(void);
    timerHandler_t (*SetTimerHandler)(timerHandler_t);
    void (*ShutdownHwTimer)(void);
} hwTimer_t;
```

Description

Hardware timer is used to trigger next event at the correct time.

Initialization

Initialized during setup time. GetSysHwTimer returns pitTimer according to CPU\_ID.

Functions

- 1. SetHwTimer  
Set timer according to hardware clock
- 2. SetupKTimers  
Init globalActiveKTimers list and set corresponding local hwtimer timer handler.
- 3. SetupHwTimer  
Setup hardware time at setup() time. localTimeInfo is also initialized here.

localTimeInfo[]

Declaration

```
//file core/kernel/setup.c
localTime_t localTimeInfo[CONFIG_NO_CPUS];
```

As delivered in the Description part below, the dynamic list is used to maintain the active timers.

The flags is used to indicate whether the next act is valid. sysHwTime is the reference to HwTimer that is attached to the CPU.

```
typedef struct {
    xm_u32_t flags;
    hwTimer_t *sysHwTimer;
#define NEXT_ACT_IS_VALID 0x1
    xmTime_t nextAct;
    struct dynList globalActiveKTimers;
} localTime_t;
```

## Description

An array that stores local time struct. `localTime_t` contains flags, sysHwTimer, nextAct time and a linked-list of active timers.

GET\_LOCAL\_TIME is used to access this array and get localTime\_t's address.

## Initialization

Described above.

## Functions

- InitKTimer
- InitVTimer

use InitKTimer. set vTimer->kTimer and thead k.

## systemStatus

### Declaration

```
//file core/objects/status.c
xmSystemStatus_t systemStatus;
```

```
typedef struct {
    xm_u32_t resetCounter;
    /* Number of HM events emitted. */
    xm_u64_t noHmEvents; /* [[OPTIONAL]] */
    /* Number of HW interrupts received. */
    xm_u64_t noIrqs; /* [[OPTIONAL]] */
    /* Current major cycle interation. */
    xm_u64_t currentMaf; /* [[OPTIONAL]] */
    /* Total number of system messages: */
    xm_u64_t noSamplingPortMsgsRead; /* [[OPTIONAL]] */
    xm_u64_t noSamplingPortMsgsWritten; /* [[OPTIONAL]] */
#ifdef defined(CONFIG_DEV_TTNOC)||defined(CONFIG_DEV_TTNOC_MODULE)
    xm_u64_t noTTnocPortMsgsRead; /* [[OPTIONAL]] */
    xm_u64_t noTTnocPortMsgsWritten; /* [[OPTIONAL]] */
#endif
    xm_u64_t noQueuingPortMsgsSent; /* [[OPTIONAL]] */
    xm_u64_t noQueuingPortMsgsReceived; /* [[OPTIONAL]] */
} xmSystemStatus_t;
```

## Description

xmSystemStatus\_t contains a seriels of counter. Such as irqs counter, reset counter, port msg read written counter.

Used only when define `CONFIG_OBJ_STATUS_ACC`.

## Initialization

## Functions

## partitionStatus

### Declaration

```
//file core/objects/status.c
xmPartitionStatus_t *partitionStatus;
```

```
typedef struct {
    /* Current state of the partition: ready, suspended ... */
    xm_u32_t state;
#define XM_STATUS_IDLE 0x0
#define XM_STATUS_READY 0x1
#define XM_STATUS_SUSPENDED 0x2
#define XM_STATUS_HALTED 0x3

    /*By compatibility with ARINC*/
    xm_u32_t opMode;
#define XM_OPMODE_IDLE 0x0
#define XM_OPMODE_COLD_RESET 0x1
#define XM_OPMODE_WARM_RESET 0x2
#define XM_OPMODE_NORMAL 0x3

    /* Number of virtual interrupts received. */
    xm_u64_t noVirqs; /* [[OPTIONAL]] */
    /* Reset information */
    xm_u32_t resetCounter;
    xm_u32_t resetStatus;
    xmTime_t execClock;
    /* Total number of partition messages: */
    xm_u64_t noSamplingPortMsgsRead; /* [[OPTIONAL]] */
    xm_u64_t noSamplingPortMsgsWritten; /* [[OPTIONAL]] */
    xm_u64_t noQueuingPortMsgsSent; /* [[OPTIONAL]] */
    xm_u64_t noQueuingPortMsgsReceived; /* [[OPTIONAL]] */
} xmPartitionStatus_t;
```

### Description

Similar as above struct

### Initialization

### Functions

## resetStatusInit

### Declaration

```
extern xm_u32_t resetStatusInit[];
```

### Description

Only the first entry of this array is used. Used at `ResetPartition` and `ResetThread` . Assigned to k->ctrl.g->partCtrlTab->resetStatus, but not used anymore.

### Initialization

```
resetStatusInit = .;
LONG(0);
```

### Functions

## hypercallFlagsTab

### Declaration

```
//file core/kernel/hypercalls.c
```

noArgs is used to indicate how many arguments of this hypercall is using.

```
extern struct {
    xm_u32_t noArgs;
#define HYP_NO_ARGS(args) ((args)&-0x80000000)
} hypercallFlagsTab[NR_HYPERCALLS];
```

### Description

This array is used to keep all hypercalls' argument numbers. The number of arguments of a hypercall is used during construct hypercall using assembly code.

### Initialization

```
//file core/kernel/arch/xm.lds.in
```



```

.rodata ALIGN(8) : AT (ADDR (.rodata) + PHYSOFFSET) {
    asmHypercallsTab = .;
    *(.ahypercallstab)
    fastHypercallsTab = .;
    *(.fhypercallstab)
    hypercallsTab = .;
    *(.hypercallstab)
    hypercallFlagsTab = .;
    *(.hypercallflagstab)
    ...
    ...
}

```

``hypercallFlagsTab`` is pointed to ``hypercallflagstab``. While ``hypercallflagstab`` is initialized by macro and assembly code

```

#define HYPERCALLR_TAB(_hc, _args) \
__asm__ (".section .hypercallstab, \"a\"\\n\\t\" \\"
        ".align 4\\n\\t\" \\"
        ".long \"#_hc\"\\n\\t\" \\"
        ".previous\\n\\t\" \\"
        ".section .hypercallflagstab, \"a\"\\n\\t\" \\"
        ".long (0x80000000|\"#_args\")\\n\\t\" \\"
        ".previous\\n\\t\"")
#define HYPERCALL_TAB(_hc, _args) \
__asm__ (".section .hypercallstab, \"a\"\\n\\t\" \\"
        ".align 4\\n\\t\" \\"
        ".long \"#_hc\"\\n\\t\" \\"
        ".previous\\n\\t\" \\"
        ".section .hypercallflagstab, \"a\"\\n\\t\" \\"
        ".long (\"#_args\")\\n\\t\" \\"
        ".previous\\n\\t\"")

```

### Functions

- MulticallSys
 

Execute a sequence of hypercalls. There will be several hypercalls from `startAddr` to `endAddr`. The iterater's offset depends on the number of arguments of a certain hypercall.
- AuditHCall
 

Only when CONFIG\_AUDIT\_EVENTS

## WindowOverflowTrap[], EWindowOverflowTrap[], WindowUnderflowTrap[], EWindowUnderflowTrap[], SIRetCheckRetAddr[], EIRetCheckRetAddr[]

### Declaration

```

//file core/kernel/arch/entry
//line 270+
ENTRY(WindowOverflowTrap)
ENTRY(EWindowOverflowTrap)
ENTRY(WindowUnderflowTrap)
ENTRY(EWindowUnderflowTrap)
//line 900+
ENTRY(SIRetCheckRetAddr)
ENTRY(EIRetCheckRetAddr)

```

### Description

These entry is used to mark 3 trap in entry.S assembly code. @function ArchTrapIsSysCtxt, if current context's pc is located between any pair of these entry, it is not system trap.

### Initialization

### Functions

- DoTrap
 

If ArchTrapIsSysCtxt, mark hmLog.opCodeH |= HMLLOG\_OPCODE\_SYS\_MASK.
- ArchTrapsSysCtxt

## ArchStartupGuest

### Declaration

```

//This should be a function
//file core/kernel/arch/head.S

```

```

ENTRY(ArchStartupGuest)
    ldd [%sp], %o0
    jmpL %g4, %g0
    add %sp, 8, %sp

```

Description

Initialization

```
void SetupKStack(kThread_t *k, void *StartUp, xmAddress_t entryPoint) {
    //only called from ResetKThread()
    extern xm_u32_t ArchStartupGuest;
    k->ctrl.kStack=(xm_u32_t *)&k->kStack[CONFIG_KSTACK_SIZE-MIN_STACK_FRAME-8];
    *--(k->ctrl.kStack)=(xm_u32_t)0; /* o1 */
    *--(k->ctrl.kStack)=(xm_u32_t)entryPoint; /* o0 */
    *--(k->ctrl.kStack)=(xm_u32_t)&ArchStartupGuest; /* %g5 */
    *--(k->ctrl.kStack)=(xm_u32_t)StartUp; /* %g4 */
    *--(k->ctrl.kStack)=(xm_u32_t)GetPsr()&~(PSR_CWP_MASK|PSR_ICC_MASK);/* %PSR (%g7) */
    *--(k->ctrl.kStack)=(xm_u32_t)2; /* %WIM (%g6) */
}
```

Functions

- 1. SetupKStack

=====

sldr[], eldr[]

Declaration

```
//TODO
//file core/ldr/ldr.sparv8.lds.in
```

```
_sldr = .;
. = (XM_PCTRLTAB_ADDR)-256*1024-(4096*18);
//...
//...
_eldr = .;
/DISCARD/ : {
    *(.note)
    *(.comment*)
}
```

Description

start and end of partition loader.

Initialization

Functions

- 1. SetupLdr

=====

smpStartBarrier

Declaration

```
//file core/kernel/setup.c
barrier_t smpStartBarrier = BARRIER_INIT;
```

```
typedef struct {
    volatile xm_s32_t v;
} barrier_t;
```

Description

Instead of taking this as a barrier, it is more like a simple spinlock, mutex or semaphore.

```
static inline void BarrierWait(barrier_t *b) {
    while(b->v);
}
static inline void BarrierLock(barrier_t *b) {
    b->v=1;
}
static inline void BarrierUnlock(barrier_t *b) {
    b->v=0;
}
```

Initialization

Functions

- 1. Setup
  - First CPU set lock BarrierLock before InitSched() and unlock it during FreeBootMem, before Schedule().
  - Second CPU will be polling smpStartBarrier. Once it is unlocked, second CPU can go for Schedule().
- 2. FreeBootMem

---

## ***sxm*[], *exm*[], *physXmcTab*[]**

### **Declaration**

### **Description**

This should be the start and end address of xm.

### **Initialization**

Initialized in core/kernel/arch/xm.ldr.in //START and END of xm //line 29 *sxm* = .; //line 139 *exm* = . + PHYSOFFSET;

// used to indicate the start of customFileTab, part of `__XMHDR` //line 138 *physXmcTab* = . + PHYSOFFSET;

### **Functions**

---

## **sysResetCounter**

### **Declaration**

```
//file core/kernel/arch/xm.ldr.in
sysResetCounter = .;
LONG(0);
```

### **Description**

A variable that keeps the reset times.

### **Initialization**

Shown in declaration.

### **Functions**

1. ResetSystem

IF WARM\_RESET then increase the counter.

ELSE (COLD\_RESET) then set counter to 0.

2. CtrlStatus

Assign current sysResetCounter to `systemStatus.resetCounter`, which will be returned to `CtrlStatus` caller.

This function is used in hypercall CtrlObjectSys.

---

## **exPTable[]**

### **Declaration**

```
//file core/kernel/arch/xm.ldr.in
exPTable = .;
*(.exptable)
LONG(0);
LONG(0);
```

### **Description**

Each element contains wwo variables: `exPTable[e].a`, and `exPTable[e].b`.

### **Initialization**

```

#define ASM_EXPTABLE(_a, _b) \
".section .exptable, \"a\"\\n\\t\" \
".align 4\\n\\t\" \
".long \"#_a\"\\n\\t\" \
".long \"#_b\"\\n\\t\" \
".previous\\n\\t\"

//_s for size; SB, SH, STUB, UH, etc....
#define ASM_RW(_s, _tmp) \
__asm__ volatile__ ("orn %0, %%g0, %0\\n\\t\" : \"=r\" (ret)); \
__asm__ volatile__ ("1:ld\"_s\" [%2], %1\\n\\t\" \
"2:st\"_s\" %1, [%2]\\n\\t\" \
"mov %%g0, %0\\n\\t\" \
"3:\\n\\t\" \
ASM_EXPTABLE(1b, 3b) \
ASM_EXPTABLE(2b, 3b) \
: \"=r\" (ret), \"=r\" (_tmp) : \"r\" (addr));

#define ASM_RD(_s, _tmp) \
__asm__ volatile__ ("orn %0, %%g0, %0\\n\\t\" : \"=r\" (ret)); \
__asm__ volatile__ ("1:ld\"_s\" [%2], %1\\n\\t\" \
"mov %%g0, %0\\n\\t\" \
"2:\\n\\t\" \
ASM_EXPTABLE(1b, 2b) \
: \"=r\"(ret), \"=r\" (_tmp) : \"r\" (addr))

```

exptable is store with pc location: “1b”, “3b”; “2b”, “3b”

So generally, exPTable[e].a is the execution code address, and exPTable[e].b is used to indicate where to go in order to skip current execution.

### Functions

#### 1. IsInPartExTable

This function is called at DoTrap @core/kernel/irqs.c.

If current trap happens when PC is equal to exPTable[e].a, then jump to exPTable[e].b to skip.

//TODO

## partitionTab

### Declaration

```

//file core/kernel/sched.c
partition_t *partitionTab;

```

Linked-list of kThread\_t to shown the current execution thread in this partition. cfg points to che partition configuration from xml parser. Partition original ID, physicalMemoryAreasOffset, noVCpus and noPorts are those not kept in partition\_t but xmcPartition.

```

typedef struct partition {
    kThread_t **kThread;
    xmAddress_t pctArray;
    xmSize_t pctArraySize;
    xm_u32_t opMode;
    xmAddress_t imgStart; /*Partition Memory address in the container*/
    ///??? container?
    xmAddress_t vLdrStack; /*Stack address allocated by XM*/
    struct xmcPartition *cfg;
} partition_t;

```

Where struct `kThread_t` is:

```

typedef union kThread {
    struct __kThread {
        // Harcoded, don't change it
        xm_u32_t magic1;
        // Harcoded, don't change it
        xmAddress_t *kStack;
        spinLock_t lock;
        volatile xm_u32_t flags;
    } __kThread;
    // [3...0] -> scheduling bits
#define KTHREAD_FP_F (1<<1) // Floating point enabled
#define KTHREAD_HALTED_F (1<<2) // 1:HALTED
#define KTHREAD_SUSPENDED_F (1<<3) // 1:SUSPENDED
#define KTHREAD_READY_F (1<<4) // 1:READY
#define KTHREAD_FLUSH_CACHE_B 5
#define KTHREAD_FLUSH_CACHE_W 3
#define KTHREAD_FLUSH_DCACHE_F (1<<5)
#define KTHREAD_FLUSH_ICACHE_F (1<<6)
#define KTHREAD_CACHE_ENABLED_B 7
#define KTHREAD_CACHE_ENABLED_W 3
#define KTHREAD_DCACHE_ENABLED_F (1<<7)
#define KTHREAD_ICACHE_ENABLED_F (1<<8)

#define KTHREAD_NO_PARTITIONS_FIELD (0xff<<16) // No. partitions
#define KTHREAD_TRAP_PENDING_F (1<<31) // 31: PENDING

    struct dynList localActiveKTimers;
    struct guest *g;
    void *schedData;
    cpuCtxt_t *irqCpuCtxt;
    xm_u32_t irqMask;
    xm_u32_t magic2;
} kThread_t;

```

For `guest` struct:

```

struct guest { #define PART_VCPU_ID2KID(partId, vCpuId) ((vCpuId)<<8)|((partId)&0xff) #define KID2PARTID(id) ((id)&0xff) #define KID2VCPUID(id) ((id)>>8)

```

For `vTimer_t` in `guest` :

```

typedef struct {
    xmTime_t value;
    xmTime_t interval;
#define VTIMER_ARMED (1<<0)
    xm_u32_t flags;
    kTimer_t kTimer;
} vTimer_t;

```

And `vClock_t` :

```

typedef struct {
    xmTime_t acc;
    xmTime_t delta;
    xm_u32_t flags;
#define VCLOCK_ENABLED (1<<0)
} vClock_t;

```

## Description

## Initialization

Initialized as zero in function `InitSched` at `core/kernel/sched.c`.

## Functions

### 1. CtrfStatus

Get partition ID from obj description and local sched. Use partition ID to access / update `partitionTab[partId]`.

### 2. HmRaiseEvent

Get partition ID from log. Take action according to `partitionTab[partitionId].cfg->hmTab[eventId].action`.

### 3. CopyArea

Get partition IDs of src partition and dst partition. Check if the area is available or not.

### 4. TriggerIrqHandler

Use `SetPartitionHwIrqPending` to partition indicated by `ctxt`.

### 5. hypercall HaltPartitionSys

Find first unflaged VCpu. If exist, partition is about to halt. Then call `HALT_PARTITION` , which just `partitionTab[id].opMode=XM_OPMODE_IDLE`.

### 6. hypercall SuspendPartitionSys, ResumePartitionSys, ResetPartitionSys

This is similar as the above one.

### 7. hypercall RaisePartitionIpmiSys, RaiselpviSys

Similar to `RaiselpviSys` as mentioned above. If partition did not set `nolpvi` pending, set `irq` pending to every VCpu.

### 8. CreatePartition

Init partition and its threads.

9. SUSPEND\_VCPU, RESUME\_VCPU, HALT\_VCPU

Take partition ID and VCpuld as input. Set the flag of partition's certain thread with `KTHREAD_XXXX_F`.

10. SUSPEND\_PARTITION, RESUME\_PARTITION, SHUTDOWN\_PARTITION, HALT\_PARTITION

Similar as above. Use loop to iterate among all VCpus.

11. GetPartition

Take current thread and the partition ID it stores to find the reference of partition.