HAL - registers
Couches basses du noyau
Organisation des sources

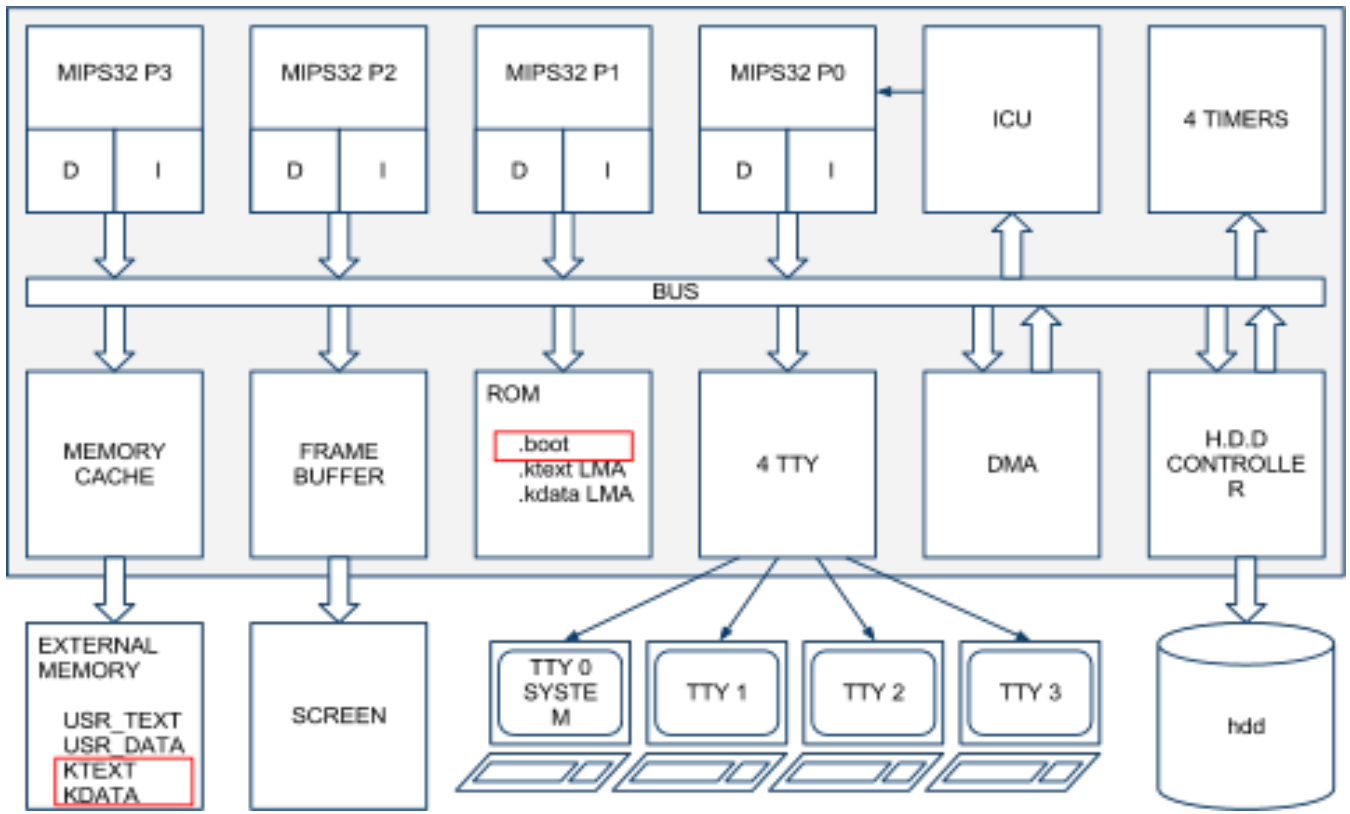
MI074 - 4

plan

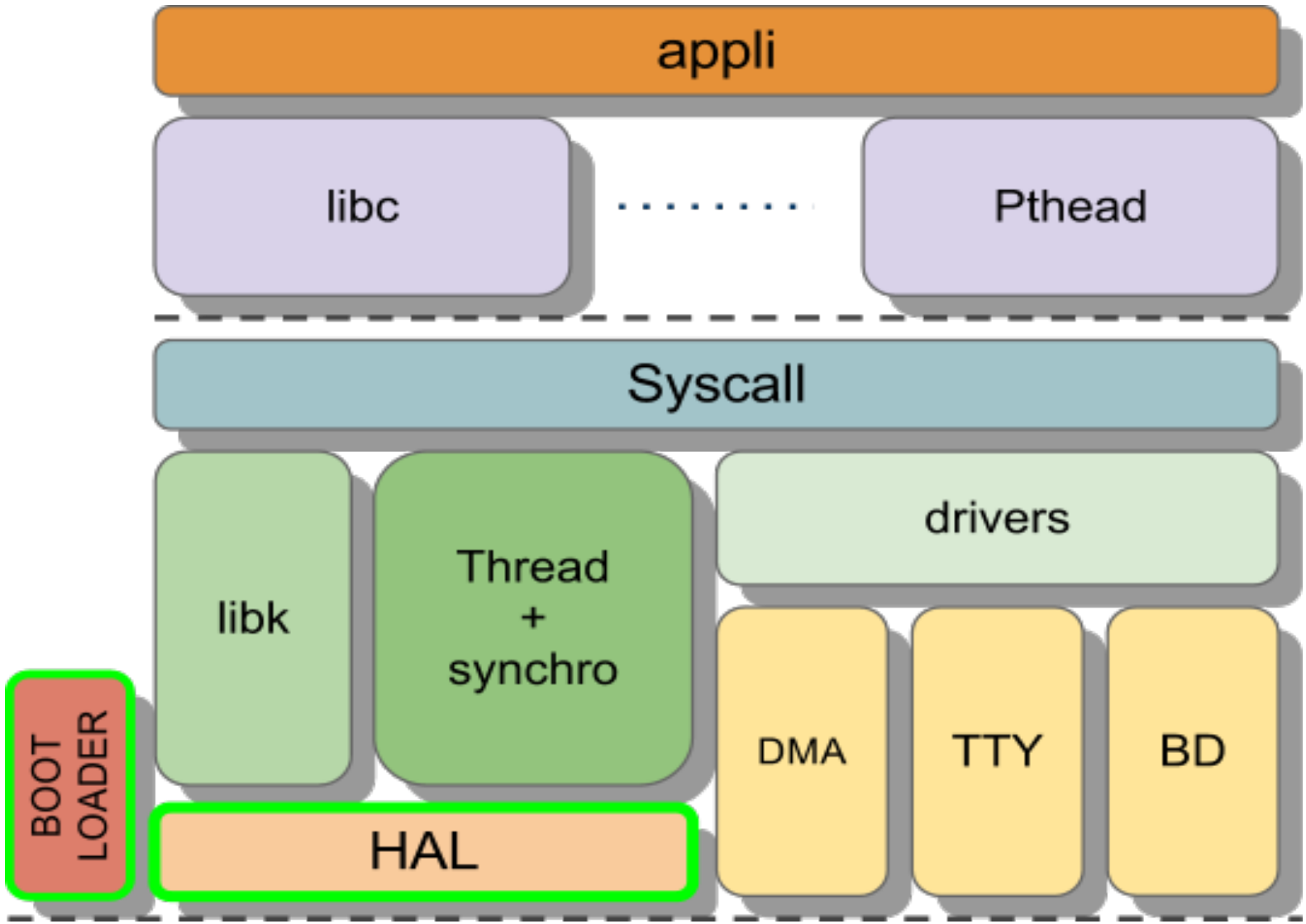
- Organisation du code
- Couche d'abstraction du CPU
- Le kentry et les trois points d'entrée du système
- Travaux à venir

Retour sur le bootloader

Le travail du bootloader permet d'exécuter le code du noyau dans ktext et d'avoir ses données dans kdata.



Architecture de l'OS final



Organisation du code

```
|-- arch          HAL architecture
|   `-- soclib      spécifique à la plateforme (p.ex. segmention.h)
|-- cpu           HAL processeur
|   `-- mipsel      spécifique au mips (p.ex. accès registres)
|-- drivers       HAL périphérique (p.ex. tty, dma, ...)
|-- kernel        code du noyau
|-- libk          bibliothèque C pour le noyau
`-- mm            allocateur mémoire
```

Usage des répertoires

```
|-- Makefile
|-- apps
|-- build
|-- hdd-img.bin
`-- src_sys
    |-- arch
    |   `-- soclib
    |       |-- devices.h
    |       |-- arch_init.c
    |       `-- segmentation.h
    |-- cpu
    |   `-- mipsel
    |       `-- boot.S
    |-- kernel
    |   |-- __boot_loader.c
    |   |-- __do_init.c
    |   |-- __do_interrupt.c
    |   `-- config.h
    |-- libk
    |   |-- tty.h
    |   `-- tty.c
    `-- mm
```

Les fichiers doivent se distribuer dans les répertoires de la manière suivante.

Il va falloir modifier le Makefile

```
KERNEL      = kernel-soclib.bin
ARCH        = soclib
CPU         = mipsel

SYS_DIR     = src_sys
SRC_SUBDIR  = cpu/$(CPU) arch/$(ARCH) kernel libk drivers mm
BUILD_DIR   ?= build
SIM_DIR     ?= ../../bin

#-------------------------------------------------------------------------
SRC_DIR    = $(addprefix $(SYS_DIR)/,$(SRC_SUBDIR))
INCLUDE    = $(foreach d,$(SRC_DIR),$(wildcard $(d)/*.h))
SRC_C      = $(foreach d,$(SRC_DIR),$(wildcard $(d)/*.c))
SRC_S      = $(foreach d,$(SRC_DIR),$(wildcard $(d)/*.S))

I_INC      = $(addprefix -I,$(SRC_DIR))
KOBJ_C     = $(addprefix $(BUILD_DIR)/,$(notdir $(SRC_C:%.c=%.o)))
KOBJ_S     = $(addprefix $(BUILD_DIR)/,$(notdir $(SRC_S:%.S=%.o)))

#-------------------------------------------------------------------------
CCTOOLS     ?=
CC          = $(CCTOOLS)/bin/$(CPU)-unknown-elf-gcc
LD          = $(CCTOOLS)/bin/$(CPU)-unknown-elf-ld
OD          = $(CCTOOLS)/bin/$(CPU)-unknown-elf-objdump
RM          = /bin/rm
SIMUL       = $(SIM_DIR)/simulation_trace.x
SIMUL       = $(SIM_DIR)/simulation.x

CFLAGS      = $(I_INC) -fno-builtin -fomit-frame-pointer -O3 -G0 -Wall -Werror -mips32r2
CFLAGS      += -std=c99
LDFLAGS     =
TRASH       = /dev/null||true
```

```
#-----------------------------------------------------------------------------
.PHONY      : clean realclean simul pdf depend

all    : depend $(KERNEL)

$(KERNEL)   : $(KOBJ_S) $(KOBJ_C) $(BUILD_DIR)/kldscript
echo [LD] $@;\
$(LD) -o $@ $(KOBJ_S) $(KOBJ_C) $(LDFLAGS) -T$(BUILD_DIR)/kldscript;\
$(OD) $@ -D > $@.dump

$(BUILD_DIR)/kldscript : $(SYS_DIR)/arch/$(ARCH)/kldscript.h $(SYS_DIR)/arch/$(ARCH)/segmentation.h
echo [CP] `basename $@`;\
cpp $< | egrep -v "#|//" | grep . > $@

clean:
$(RM) vcitty* *.bak $(BUILD_DIR)/* *.pdf \
    .DS_Store */.DS_Store */*/.DS_Store */*/*/.DS_Store \
    *~ */*~ */*/*~ */*/*/*~ *.dump tags
    #2> $(TRASH)

realclean: clean
$(RM) $(KERNEL)

simul: $(KERNEL)
$(SIMUL)

pdf:
echo [LP] `basename $$PWD`.pdf;\
a2ps -1 --medium=A4 --file-align=fill -o - -l100 \
       Makefile $(INCLUDE) $(SRC_S) $(SRC_C) |\
ps2pdf -sPAPERSIZE=a4 - `basename $$PWD`.pdf
```

```
#---------------------------------------------------------------------------
$(BUILD_DIR)/%.o: */*/*/%.c ; echo [CC] $*.o;$(CC) $(CFLAGS) -c $< -o $@
$(BUILD_DIR)/%.o: */*/%.c ; echo [CC] $*.o;$(CC) $(CFLAGS) -c $< -o $@
$(BUILD_DIR)/%.o: */*/*/%.S ; echo [CC] $*.o;$(CC) $(CFLAGS) -c $< -o $@

depend:
    ctags -w $(INCLUDE) $(SRC_C) $(SRC_S)                                          ;\
    awk '(NR==1),/^\t*# .* AUTOMATIC DEPENDANCIES/' Makefile                       |\
    grep -v "^[ \t]*# .* AUTOMATIC DEPENDANCIES"                    > Makefile.new ;\
    echo "# DO NOT DELETE THIS LINE : AUTOMATIC DEPENDANCIES"          >> Makefile.new ;\
    gcc -MM $(I_INC) $(SRC_C) $(SRC_S)|sed 's/^\([^ ]\)/$$(BUILD_DIR)\/\1/' >> Makefile.new ;\
    mv Makefile.new Makefile
# DO NOT DELETE THIS LINE : AUTOMATIC DEPENDANCIES
```

HAL : Hardware Abstraction Layer

Comme son nom l'indique, une HAL est définie par un ensemble de structures de données et de fonctions qui permettent d'accéder au matériel de manière uniforme indépendamment du matériel réel.

La HAL est défini à travers plusieurs interfaces

- Interface Kernel / CPU
- Interface Kernel / Arch
- Interface Kernel / Devices

Nous allons voir d'abord l'interface CPU puis les interfaces Arch et Devices dans les prochains cours.

HAL-CPU : redéfinition des types entier

- Tous les processeurs n'ont pas la même largeur de mot, donc la largeur type int dépend du CPU.
- Il faut rédéfinir les types standards dans cpu/mipsel :

  types.h
  - long : mot machine
  - int : 4 octets

```
#ifndef _TYPES_H_
#define _TYPES_H_

#ifndef NULL
#define NULL                    (void*)0
#endif

typedef unsigned long       uint_t;
typedef signed long         sint_t;

typedef unsigned char       uint8_t;
typedef signed char         sint8_t;

typedef unsigned short
int  uint16_t;
typedef signed short
int    sint16_t;

typedef unsigned
int         uint32_t;
typedef signed
int           sint32_t;

typedef uint32_t            size_t;
typedef sint32_t            ssize_t;

typedef sint32_t            error_t;
```

Code de la HAL

- L'interface de la HAL est présente dans un .h dans le répertoire kernel.
- Puisque la plupart des services sont courts, ils sont décrits dans des fonctions inlinées, donc dans un .h et pas dans .c
- dans kernel, on a :
  hal-cpu.h
  qui inclue le fichier présent dans cpu/mipsel
  hal-cpu-code.h
- C'est dans le Makefile en fonction d'un paramètre CPU que l'on indique quel hal-cpu-code.h inclure.

HAL : Interface Kernel / CPU

Les services : fonctions inlinées

registres spéciaux
numéro de proc, timestamp, numéro thread.
irq du CPU
masquage, démasquage.
spinlock
trylock, lock, unlock, **...**
opérations atomiques
incrément, décrément, addition, ...
caches
invalidation de ligne de cache data
context
création, destruction, chargement, sauvegarde.

HAL : Time, identity & special registers

dans hal-cpu.h

- static inline uint_t cpu_get_id(void);
- static inline uint_t cpu_time_stamp(void);
- static inline struct thread_s *cpu_current_thread(void);
- static inline void cpu_set_current_thread(struct thread_s * thread);

dans hal-cpu-code.h

```
static inline uint_t cpu_get_id(void)
{
    register unsigned int proc_id;
    asm volatile ("mfc0  %0, $0":"=r" (proc_id));
    return proc_id;
}
static inline uint_t cpu_time_stamp(void)
{
    register uint_t cycles;
    asm volatile ("mfc0  %0, $9":"=r" (cycles));
    return cycles;
}
static inline struct thread_s *cpu_current_thread(void)
{
    register void *thread;
    asm volatile ("mfc0 %0, $4, 2":"=r" (thread));
    return thread;
}
static inline void cpu_set_current_thread(struct thread_s *thread)
{
    asm volatile ("mtc0 %0, $4, 2"::"r" (thread));
}
```

HAL : IRQ register/enable/disable/restore

dans hal-cpu.h

- static inline void cpu_disable_single_irq(uint_t irq_num, uint_t * old);
- static inline void cpu_enable_single_irq(uint_t irq_num, uint_t * old);
- static inline void cpu_disable_all_irq(uint_t * old);
- static inline void cpu_enable_all_irq(uint_t * old);
- static inline void cpu_restore_irq(uint_t old);

Notez que seules les interruptions du CPU nous interresse ici. Les interruptions qui passe par l'ICU seront abstraites dans l'architecture.

HAL : Spinlock trylock/unlock/init

Attente actives dans hal-cpu.h

- static inline bool_t cpu_spinlock_trylock(uint_t *lock);
- static inline void cpu_spinlock_lock(uint_t *lock);
- static inline void cpu_spinlock_unlock(uint_t *lock);
- static inline void cpu_spinlock_init(uint_t *lock);
- static inline void cpu_spinlock_destroy(uint_t *lock);

Dans notre cas seule la première fonction est délicate puisqu'elle utilise du code assembleur et les instruction ll et sc.

```
static inline void cpu_spinlock_init(uint_t * lock)
{
   __asm__ volatile (
      "sync            \n" // to be sure that previous store are acheived
      "sw $0, (%0)     \n" // unlock
      "sync            \n" // to empty the write buffer
      ::"r" (lock));
}
static inline bool_t cpu_spinlock_trylock(uint_t * lock)
{
   register bool_t state;
   __asm__ volatile (
      ".set noreorder   \n" // do not modify the sequence
      "lw    %0, (%1)   \n" // load lock state
      "bnez  %0, 32     \n" // forgive whenever lock is busy (i.e. != 0)
      "nop             \n" // delayed slot
      "ll    %0, (%1)   \n" // load lock value in $2 and try to register in memory
      "nop             \n" // delayed slot
      "bnez  %0, 16     \n" // forgive whenever lock is busy
      "ori   %0, 1      \n" // preload 1
      "sc    %0, (%1)   \n" // sc returns in %0 <- 1 if free, 0 if busy
      "nop             \n" // delayed slot
      "xori  %0, %0,  1 \n" // trylock returns 0 if free, 1 if busy
      ".set  reorder    \n" // return to normal mode
      :"=r" (state)
      :"r" (lock));
   return state;
}
static inline void cpu_spinlock_lock(uint_t * lock) {
   while ((cpu_spinlock_trylock(lock)));
}
static inline void cpu_spinlock_unlock(uint_t * lock) {
   cpu_spinlock_init(lock);
}
static inline void cpu_spinlock_destroy(uint_t * lock){}
```

kentry : l'entrée du système

- On entre dans le système à la suite de 3 événements:

    1. une interruption provenant d'un périphérique.
    2. une exception détectée par le matériel
    3. une demande de service par l'utilisateur

- Pour le moment, on ne va traiter que les deux premiers

- L'entrée du système est à l'adresse kentry
  Le code est spécifique au CPU

kentry : comportement

Dans les cas 1. et 2. l'entrée est imprévisible,
Dans le cas 3. il est prévu

kentry:
test de la cause d'appel
si c'est un **appel système alors**
    on change de pile et on appelle la fonction **__do_syscall**
sinon si c'est une **interruption alors**
    si on est en mode user **alors** on change de pile **fsi**
    on sauve les registres "temporaires" et on appelle la fonction **__do_interrupt**
      sinon
    on sauve tous les registres et on appelle **__do_exception**
      fsi

```
#include <mips_regs.h>
#-----------------------------------------------------------------------------
# Kernel entry point (Exception/Interrupt/System call) for MIPS32 ISA compliant
# processors. The base address of the segment containing this code
#-----------------------------------------------------------------------------
    .section   .kentry,"ax",@progbits
    .extern    __do_interrupt
    .extern    __do_exception
    .ent       kentry
    .set       noat
    .set       noreorder
    .org       0x180

#number of arguments of called functions
#define NBA         2

# Kernel Entry point
#-----------------------------------------------------------------------------
kentry:
   # alloc memory in stack to be able to save all registers
   addiu   $29,    $29,    -(SAVE_REG_NB+NBA)*4  # max registers to save + args

   # just save temporary registers
   sw      $1,     (NBA+AT)*4($29)
   sw      $2,     (NBA+V0)*4($29)
   sw      $3,     (NBA+V1)*4($29)
   sw      $4,     (NBA+A0)*4($29)
   sw      $5,     (NBA+A1)*4($29)
   sw      $6,     (NBA+A2)*4($29)
   sw      $7,     (NBA+A3)*4($29)
   sw      $8,     (NBA+T0)*4($29)
   sw      $9,     (NBA+T1)*4($29)
   sw      $10,    (NBA+T2)*4($29)
   sw      $11,    (NBA+T3)*4($29)
   sw      $12,    (NBA+T4)*4($29)
   sw      $13,    (NBA+T5)*4($29)
   sw      $14,    (NBA+T6)*4($29)
   sw      $15,    (NBA+T7)*4($29)
   sw      $24,    (NBA+T8)*4($29)
   sw      $25,    (NBA+T9)*4($29)
   sw      $28,    (NBA+GP)*4($29)
   sw      $31,    (NBA+RA)*4($29)
   mflo    $1
   mfhi    $2
   mfc0    $3,     $14             # Read EPC
   mfc0    $5,     $13             # read CR (used later)
   sw      $1,     (NBA+LO)*4($29)
   sw      $2,     (NBA+HI)*4($29)
   sw      $3,     (NBA+EPC)*4($29)    # Save EPC

   # test cause register jump to cause_int if it is an interrupt
   andi    $6,     $5,     0x3         # apply interrupt mask (used later)
   beq     $6,     $0,     cause_int
   mfc0    $4,     $0              # CPU_ID, 1th arg
```

```
#ifndef _MIPS_REGS_H_
#define _MIPS_REGS_H_

#define     AT          0
#define     V0          1
#define     V1          2
#define     A0          3
#define     A1          4
#define     A2          5
#define     A3          6
#define     T0          7
#define     T1          8
#define     T2          9
#define     T3          10
#define     T4          11
#define     T5          12
#define     T6          13
#define     T7          14
#define     T8          15
#define     LO          16
#define     T9          17
#define     SR          18
#define     HI          19
#define     RA          20
#define     EPC         21
#define     GP          22
#define     S0          23
#define     S1          24
#define     S2          25
#define     S3          26
#define     S4          27
#define     S5          28
#define     S6          29
#define     S7          30
#define     CR          31
#define     SP          32
#define     S8          33
#define     SAVE_REG_NB  34

#endif
```

```
# Exception
# ----------------------------------------------

   # since this is an exception, save all the remaining registers
   mfc0   $1,    $12              # Read current SR (used later)
   addiu  $2,    $29,   (SAVE_REG_NB+NBA)*4
   sw     $1,    (NBA+SR)*4($29)    # Save SR
   sw     $2,    (NBA+SP)*4($29)
   sw     $5,    (NBA+CR)*4($29)
   sw     $16,   (NBA+S0)*4($29)
   sw     $17,   (NBA+S1)*4($29)
   sw     $18,   (NBA+S2)*4($29)
   sw     $19,   (NBA+S3)*4($29)
   sw     $20,   (NBA+S4)*4($29)
   sw     $21,   (NBA+S5)*4($29)
   sw     $22,   (NBA+S6)*4($29)
   sw     $23,   (NBA+S7)*4($29)
   sw     $30,   (NBA+S8)*4($29)

   # call function __do_exception( cpu_id, regs_table)
   la     $27,   __do_exception
   or     $5,    $0,    $29    # regs_tbl, 2th arg
   jr     $27
   addiu  $5,    $5,    NBA*4

# Interrupt
# ----------------------------------------------
cause_int:

   # call function __do_interrupt(cpu_id, irq_state)
   la     $27,   __do_interrupt
   srl    $5,    $5,    10     # extract irq state
   jal    $27
   andi   $5,    $5,    0x3F   # 6 HW IRQ LINES, 2th arg is irq_state
```

```
# Kentry exit
# ----------------------------------------------

   # only restore temporary register
   lw     $1,    (NBA+LO)*4($29)
   lw     $2,    (NBA+HI)*4($29)
   lw     $3,    (NBA+EPC)*4($29)
   mtlo   $1
   mthi   $2
   mtc0   $3,    $14
   lw     $1,    (NBA+AT)*4($29)
   lw     $2,    (NBA+V0)*4($29)
   lw     $3,    (NBA+V1)*4($29)
   lw     $4,    (NBA+A0)*4($29)
   lw     $5,    (NBA+A1)*4($29)
   lw     $6,    (NBA+A2)*4($29)
   lw     $7,    (NBA+A3)*4($29)
   lw     $8,    (NBA+T0)*4($29)
   lw     $9,    (NBA+T1)*4($29)
   lw     $10,   (NBA+T2)*4($29)
   lw     $11,   (NBA+T3)*4($29)
   lw     $12,   (NBA+T4)*4($29)
   lw     $13,   (NBA+T5)*4($29)
   lw     $14,   (NBA+T6)*4($29)
   lw     $15,   (NBA+T7)*4($29)
   lw     $24,   (NBA+T8)*4($29)
   lw     $25,   (NBA+T9)*4($29)
   lw     $30,   (NBA+GP)*4($29)
   lw     $31,   (NBA+RA)*4($29)

   addiu  $29,   $29,   (SAVE_REG_NB+NBA)  # max registers to save + args
   eret

   .set reorder
   .set at
   .end kentry
#------------------------------------------------------------------------------
```

Résumé

- Après le boot tous les processeurs démarrent __do_init().
- Un processeur va être chargé de demander l'initialisation de l'architecture avec la fonction arch_init() puis d'initialiser les structures du noyau et de créer les threads de départ.
- Nous allons nous intéresser à ce processeur de démarrage. Les autres vont rester en attente.
- Pour le TME à venir nous allons donc
  - organiser le code dans les répertoires
  - changer le Makefile en conséquence
  - ajouter la HAL incomplete
  - ajouter la fonction arch_init()
  - ajouter le kentry
  - traiter la première interruption