

## 目录

<b>1</b>	<b>2k500 下启动方式指南</b>	<b>2</b>
1.1	大致思路 . . . . .	2
1.2	QEMU 使用方式 . . . . .	2
1.2.1	QEMU 的 tftp 配置 . . . . .	2
1.2.2	uImage 生成与持久化启动 . . . . .	2
1.2.3	手动与自动启动 . . . . .	3
1.2.4	某些终端的问题 . . . . .	7
1.3	龙芯 2k500 先锋板启动方式 . . . . .	7
1.3.1	设备连接 . . . . .	7
1.3.2	tftp 搭建 . . . . .	9
1.3.3	终端启动 . . . . .	9
1.3.4	非持久化启动方式 . . . . .	12
1.4	调试器启动方式 . . . . .	16
<b>2</b>	<b>从 NPUcore-LA 的启动看 LoongArch 启动的步骤介绍</b>	<b>16</b>

## 1 2k500 下启动方式指南

### 1.1 大致思路

2k500 使用 u-boot 启动，因此我们使用 mkimage 封装编译出的二进制文件，通过 tftp 传输后从内存启动

### 1.2 QEMU 使用方式

#### 1.2.1 QEMU 的 tftp 配置

QEMU 的 tftp 是直接映射到文件夹的，设置较为简单。

打开 util/qemu-2k500/gz/runqemu2k500，可见有：(23 行开始)

```
-drive if=mtd,file="$OS" \  
-net nic -net user,net=192.168.1.2/24,tftp=$TFTP_DIR \  
-net nic -net user,net=10.0.3.0/24\  

```

这里 TFTP\_DIR 为其 TFTP 的文件夹，这一选项可以自行配置。我们配置为 ‘easy-fs-fuse’

刷入的系统必须重命名为 uImage，文件系统需要是 rootfs-ubifs-ze.img

#### 1.2.2 uImage 生成与持久化启动

2k500 的引导文件生成过程如下：

1. cargo build 出 ELF 文件 (参见 os/Makefile:130, 以 debug 为例)

```
@cargo build --no-default-features \  
--features "board_$(BOARD) $(LOG_OPTION)" --target $(TARGET)
```

2. objcopy 剥离出纯二进制 (os/Makefile:90)

```
@$(OBJCOPY) $(KERNEL_ELF) $@ --strip-all -O binary &
```

其中, KERNEL\_ELF 的位置一般在 os/target/loongarch64-unknown-linux-gnu/debug

或者如果 MODE=release 则在 release 上

### 3. 制作 uImage(os/Makefile:94)

```
../util/mkimage -A loongarch -O linux -T kernel -C none -a $(LA_LOAD_ADDR)\  
-e $(LA_ENTRY_POINT) -n NPUcore+ -d $(KERNEL_BIN) $(KERNEL_UIMG)
```

之所以命名为 uImage 是因为 2k500 的 u-boot 版本固定加载 tftp 上的 uImage 作为复制和刷新的内容

最后拷贝到 easy-fs-fuse 中 (os/Makefile:97)

```
@cp -f $(pwd)/target/$(TARGET)/$(MODE)/os.ui $(U_IMG)
```

这种引导方式的特点是, 如果不在虚拟介质中将操作系统的代码擦除, 则可以在重启后不需要刷入代码, 直接运行.

另外, LoongArch 的 mkimage 暂未主线化, 可能需要单独编译: 在龙芯官方提供的 2k500 的 u-boot 代码中可以找到 tools 中:

make

然后生成 mkimage 即可. 本软件的 repo 中附带了 Linux 下的 mkimage.

官方附赠的资料包如下: <https://pan.baidu.com/s/1CfAeV3mSDJPw6zyw5Mv0ww>, 提取码: nl80

### 1.2.3 手动与自动启动

启动流程较为简单, 首先在当前位置按下 c 可以进入命令行, 按下 m 可以进入菜单. 我们可以利用 m 进入菜单.(如果手动建议多次按下直到进入功能)

```

uncachelock base = 0x90000000 size = 0x40000

      _      _      _      _      _      _      _      _      _      /      _      _      \
      |      |      |      |      |      |      |      |      |      |      |      |      |      |
      |_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|

Trying to boot from SPI

U-Boot 2022.04-geba9c536 (Feb 05 2023 - 08:44:03 +0800)

CPU:      LA264
Speed: Cpu @ 64 MHz/ Mem @ 1 MHz/ Bus @ 300 MHz
Model: loongson-2k500
Board: LS2K500-MINI-DP
DRAM: 512 MiB
Core: 51 devices, 22 uclasses, devicetree: board
NAND: using 4-bit/512 bytes BCH ECC
device found, Manufacturer ID: 0x2c, Chip ID: 0xda
Micron NAND 256MiB 3,3V 8-bit
256 MiB, SLC, erase size: 128 KiB, page size: 2048, OOB size:
 64
256 MiB
Loading Environment from SPIFlash... SF: Detected gd25q128 wi
th page size 256 Bytes, erase size 4 KiB, total 16 MiB
OK
In:      serial
Out:     serial vidconsole
Err:     serial vidconsole
dpm_en:0x0
dpm_sts:0x0
Net:     eth0: ethernet@0x1f020000
***** Notice *****
Press c to enter u-boot console, m to enter boot menu
*****
Bus ehci@1f050000: USB EHCI 1.00
Bus ohci@1f058000: USB OHCI 1.0
scanning bus ehci@1f050000 for devices... 1 USB Device(s) fou
nd
scanning bus ohci@1f058000 for devices... 1 USB Device(s) fou
nd

```

如果错过菜单, 进入命令行, 可以通过 `bootmenu` 命令重新进菜单  
然后, 我们可以使用 `Update Kernel` 进入内核刷写. 如果需要刷入文件系统, 则使用 `3: update rootfs`

```
*** U-Boot Boot Menu ***

[1] System boot select
[2] Update kernel
[3] Update rootfs
[4] Update u-boot
[5] Update ALL
[6] System install or recover
[7] U-Boot console

Press UP/DOWN to move, ENTER to select, ESC/CTRL+C to quit
```

```
*** U-Boot Boot Menu ***

[1] Update kernel (uImage) to nand flash (by usb)
[2] Update kernel (uImage) to nand flash (by tftp)
[3] Return

Press UP/DOWN to move, ENTER to select
```

刷写完成后即可 `bootm` 命令进入系统

```

update kernel.....
Speed: 1000, full duplex
Using ethernet@0x1f020000 device
TFTP from server 192.168.1.2; our IP address is 192.168.1.20
Filename 'uImage'.
Load address: 0x9000000003000000
Loading: #####
          27.4 MiB/s

done
Bytes transferred = 488544 (77460 hex)
SF: Detected gd25q128 with page size 256 Bytes, erase size 4
KiB, total 16 MiB
update kernel to nand.....

NAND erase.part: device 0 offset 0x0, size 0xe00000
Scanning device for bad blocks
Erasing at 0xde0000 -- 100% complete.
OK

NAND write: device 0 offset 0x0, size 0x77460
488544 bytes written: OK

#####
### update target: kernel
### update way   : tftp
### update result: success
#####

=> bootm

```

我们使用 ‘expect’ 软件进行终端自动化执行，其指令如下：

```

#!/usr/bin/expect -f
set direct [lindex $argv 0];
set qemu_arg [lindex $argv 1];
set timeout -1
spawn "$direct/runqemu2k500" $qemu_arg
expect "Device(s) found"
send "m"
expect "Device(s) found"

```

```

send "m"
expect "Press UP/DOWN to move, ENTER to select, ESC/CTRL+C to quit"
send "\033\[B\r"
expect "Press UP/DOWN to move, ENTER to select"
send "\033\[B\r"
expect "update result: success"
send "bootm\r"
send "cd syscall\r"
send "./run-all.sh\r"
send "cd ..\r"
send "lua_testcode.sh\r"
send "busybox_testcode.sh\r"
interact

```

chmod +x run\_script 给文件赋予执行权限后, 可以直接执行该程序  
 Makefile 中, 使用./run\_script 即可.

其中 send 是发送字符操作, expect 是检测特定的屏幕输出

注意, 需要将 set timeout -1, 否则会随机出现超时运行失败的情况

#### 1.2.4 某些终端的问题

由于方向键是通过 ANSI 字符实现的, 在 Emacs, Vim 等某些编辑器自带的终端上可能需要将终端传输字符改为 OA 而不是方括号,

否则会出现一旦输入错误的字符就退出 TUI 菜单的问题

### 1.3 龙芯 2k500 先锋板启动方式

#### 1.3.1 设备连接

**串口连接** 如图, 将 2k0500 先锋版按照图中的方式连接:



注意, 线序是: 上面绿色, 下面左黑右白, 这是 Debug 串口 (UART2), 其对应的地址是: 0x1FF40800(见手册)

**有线网络连接** 有时候, 通过串口传输的速率较慢, 需要使用网线, 但网线传输可能比较慢, 需要在进入后终端 20sec 后才能应答成功, 因此需要较长时间的等待.



### 1.3.2 tftp 搭建

tftp 服务器安装:

```
sudo apt-get install tftp-hpa tftpd-hpa
```

配置文件默认在/etc/default/tftpd-hpa 将其修改为

```
# /etc/default/tftpd-hpa
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/path/to/project1466467-172876/easy-fs-fuse"
TFTP_ADDRESS=":69"
TFTP_OPTIONS="-l -c -s"
```

然后

```
sudo service tftpd-hpa restart
```

**注意**部分 tftp(Linux 下) 可能不支持连接的文件, 如果使用 ln 进行加载, 可能会失败.

### 1.3.3 终端启动

在 Linux 或其他终端下 (VMWare 需要手动开启端口映射, WSL 请自行搜索串口教程), 需要使用串口和开发板进行通信.

常用的有 minicom 或者 miniterm, 但前者默认不支持换行符号, 需要单独配置;

minicom 启动的命令如下:

```
minicom -w -c on -D /dev/ttyUSB0
```

几个选项的意义是: -w 为自动换行, -c 为彩色字符 (开启用于启用终端色彩). -D 后面的是设备文件名, 要根据实际情况配置.


且为了适应不同程序的不同换行符, 需要修改或者创建配置文件 ~/.minirc.dfl:(可以参考 util/.minirc.dfl)

# 电脑产生的文件 - 使用 minicom 中的设置菜单以变更参数。

```
pu rtscts          No
pu port            /dev/ttyUSB0
pu updir           /path/to/project1466467-172876/easy-fs-fuse
pu addcarreturn    Yes
```

如果配置文件编写失败 (上述命令启动后会冻结屏幕, 一直不出现” 欢迎使用 minicom 2.8”), 则建议使用下列方法: 进入窗口后,

#### 1. 按下 Ctrl-A 然后 Z



```
dragon@dragon-Lenovo-K2450: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Image Type:  LoongArch Linux Kernel Image (uncompressed)
Data Size:
Load Addr:
Entry Poi:
Verifying:
Loading K:
Warning: inv
[Kernel] UAR
[bootstrap_i] 拨号电话本.....D 运行脚本.....G | 清除屏幕.....C
[kernel] Con] 发送文件.....S 接收文件.....R | 设置 Minicom.....O
last 53080 P] 通信参数.....P 加入换列字符.....A | 暂停 minicom.....J
.text [0x0, | 截取 开/关.....L 挂断.....H | 离开并重置.....X
.rodata [0x6] 送出 break.....F 初始化调制解调器.....M | 离开而不重置.....Q
[ta [0x800] 终端设置.....T 运行 Kermit.....K | 光标键模式.....I
.bss [0x8700] 自动换行 开/关.....W 本机回应 开/关.....E | 说明画面.....Z
[ping .tex] 粘贴文件.....Y 时间戳记切换.....N | 回卷.....B
mapping .rod] 加入换列字符.....U
mapping .dat
mapping .bss
mapping phys
mapping memory-mapped registers
[get_timer_freq_first_time] clk freq: 100000000(from CPUCFG)
[CPUCFG 0x0] 1351680
[CPUCFG 0x1] 31617662
[CPUCFG 0x2] 7389191
[CPUCFG 0x3] 3263
[CPUCFG 0x4] 100000000
[CPUCFG 0x5] 65537
[CPUCFG 0x6] 32563
[CPUCFG 0x10] 93
[CPUCFG 0x11] 101122051
[CPUCFG 0x12] 101122051
[CPUCFG 0x13] 101318663
[CPUCFG 0x14] 0
Misc ( 32-bit addr plv(1,2,3):: false,false,false, rdtime allowed for plv(1,2,3):: false,false
,false, Disable dirty bit check for plv(0,1,2):: false,false,false, Misalignment check for plv
(0,1,2,4):: false,false,false,false )
RVACfg ( rbits: 0 )
[machine_init] MMAP_BASE: 0xffffffff00000000
[Kernel] Hello, world!
NPUCore:/#
```

然后按下 u, 之后会提示已经打开加入换列字符

```
NPUCore: /#
关闭加入换列字符
打开加入换列字符
CTRL-A Z for help | 115200 8N1 | NOR
```

按下 Ctrl-A 然后 Z 然后 O,

```
arning. Invalid device tree. Used Linux default dcb
ernel] UART address: 0x800000001ff40800
bootstrap_init] PRCfg1 { SAVE reg. number: 4, Timer bi
ernel] Con+-----[设置]-----+
st 53080 P| 文件名和路径 |
ext [0x0, | 文件传输协定 |
odata [0x6| 串口设置 |
ata [0x800| 调制解调器和拨接 |
ss [0x8700| 屏幕和键盘 |
pping .tex| 保存设置为 df1 |
pping .rod| 另存设置为... |
pping .dat| 离开本画面 |
pping .bss+-----+
pping physical memory
```

上下键移动光标选择保存设置为 df1

miniterm 的启动命令则是:

```
python3 -m serial.tools.miniterm --eol LF --dtr 0 --rts 0 --filter \
    direct /dev/ttyUSB0 115200
```

miniterm 在某些屏幕显示的情况下会产生错误, 且部分功能比较简单 (没有通过 uart 传输文件的功能),

因此需要演示, debug 和串口传输文件的场景建议采用 minicom. 但对于纯粹的自动化测评, miniterm 其实更加合适

### 1.3.4 非持久化启动方式

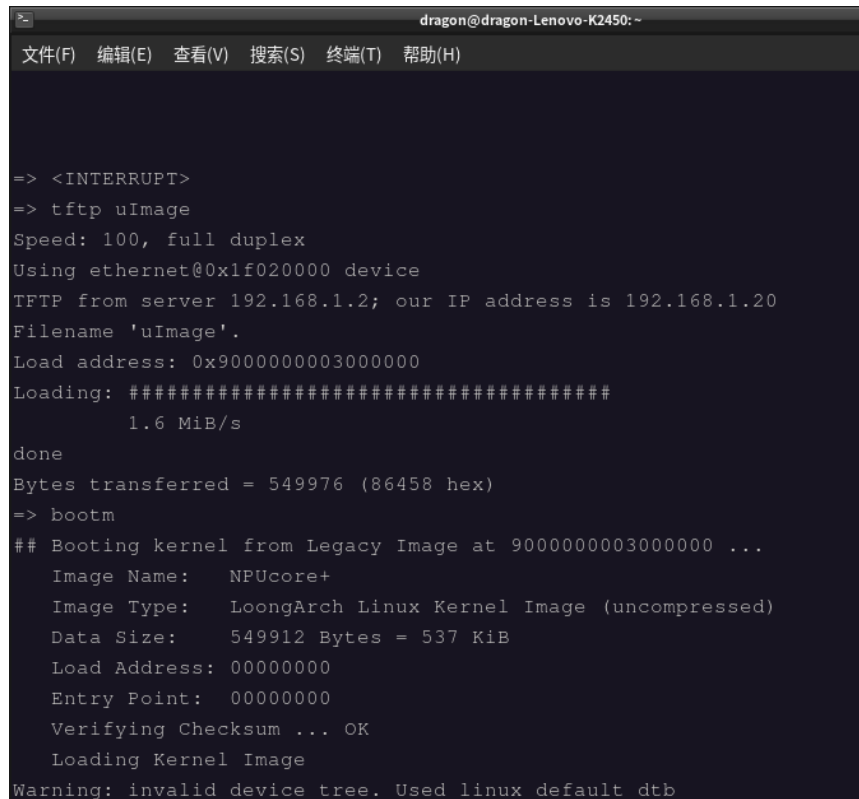
上文介绍了 QEMU 的持久化启动方式, 但这里我们再介绍一种非持久化的启动方式. 由于本脚本方案没有使用该方法, 因此并没有专门设计自动化脚本给这个方案.

**使用 uImage 直接引导** 可以直接使用 tftp 加载 uImage 到内存 (在之前介绍的 c 进入命令行, 或者在上文的菜单进入 U-Boot console)

```
tftp uImage
```

```
bootm
```

屏幕会提示:



```
dragon@dragon-Lenovo-K2450: ~
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

=> <INTERRUPT>
=> tftp uImage
Speed: 100, full duplex
Using ethernet@0x1f020000 device
TFTP from server 192.168.1.2; our IP address is 192.168.1.20
Filename 'uImage'.
Load address: 0x9000000003000000
Loading: #####
          1.6 MiB/s
done
Bytes transferred = 549976 (86458 hex)
=> bootm
## Booting kernel from Legacy Image at 9000000003000000 ...
   Image Name:   NPUCore+
   Image Type:   LoongArch Linux Kernel Image (uncompressed)
   Data Size:    549912 Bytes = 537 KiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Loading Kernel Image
Warning: invalid device tree. Used linux default dtb
```

即可引导系统

**使用 os.bin 直接引导** 如果不想制作 uImage, 也可以采用 os.bin 引导:

```
tftp os.bin 0x9000000000000000
go 0x9000000000000000
```

系统可能会提示覆盖了部分保留的内存, 但实际效果上看可能没有太大的问题

```
=> tftp os.bin
Speed: 100, full duplex
Using ethernet@0x1f020000 device
TFTP from server 192.168.1.2; our IP address is 192.168.1.20
Filename 'os.bin'.
Load address: 0x9000000003000000
Loading: #####
          2.7 MiB/s
done
Bytes transferred = 549912 (86418 hex)
=> tftp os.bin 0x9000000000000000
Speed: 100, full duplex
Using ethernet@0x1f020000 device
TFTP from server 192.168.1.2; our IP address is 192.168.1.20
Filename '0x9000000000000000'.

TFTP error: trying to overwrite reserved memory...
=> go 0x9000000003000000
```

**使用串口发送** 如果不想连接网线, 只希望使用串口发送, 可以尝试下列方法: 首先按照之前给出的 minicom 方式启动进入 U-Boot 命令行, 用 loadx 命令加载,

```
loadx
```

然后是 minicom 操作: Ctrl-A 然后 s



然后选择 xmodem



```
=> loadx 0x9000000000000000
## Ready for binary (xmodem) download to 0x9000000000000000 at 115200 bps...
CCCCCCC## Total Size      = 0x00086418 = 549912 Bytes
## Start Addr              = 0x9000000000000000
=> go 0x9000000003000000
```

其他的几个方法也可以实验 (loadb 对应 kermit, load[xyz] 对应 [xyz]modem). 但是该方法虽然不需要网线, 可是速度较慢, 适合某些网线被其他功能占用的方法.

## 1.4 调试器启动方式

如果你获得了龙芯中科的调试器 (黑盒), 黑盒的本质就是一个开发板, 所以可以直接连接 USB 串口启动, 连接好网线和 jtag 线 (到 2k500 开发板) 然后在终端执行下列命令:

```
busybox ip addr add 192.168.1.3/24 dev eth0 #设置ip地址
```

```
cd tmp/ejtag-debug
```

```
./la_dbg_tool_gpio #执行远程gdbserver
```

```
source configs/config.ls1a500 # 注意, 配置文件是ls1a500而不是2k500, 如果配置文件不对,
```

```
# 如果正常则可以继续, 否则会卡死
```

```
# 最后可以
```

```
gdbserver
```

然后在 PC 端, 可以用 gdb:

```
target remote 192.168.1.3:50010
```

即可连接调试器

## 2 从 NPUcore-LA 的启动看 LoongArch 启动的步骤介绍

我们用注释并修改过 (删除了部分的启动自检信息) 的启动代码解释 LoongArch 下的启动流程.

首先是 NPUcore 的主函数 (见 os/src/arch/la64/mod.rs):



```

#[no_mangle]
pub fn rust_main() -> ! {
    bootstrap_init();
    mem_clear();
    console::log_init();
    println!("[kernel] Console initialized.");
    mm::init();
    machine_init();
    println!("[kernel] Hello, world!");

    //machine independent initialization
    fs::directory_tree::init_fs();
    task::add_initproc();

    // note that in run_tasks(), there is yet *another* pre_start_init()
    // which is used to turn on interrupts in some archs like LoongArch.
    task::run_tasks();
    panic!("Unreachable in rust_main!");
}

```

我们可以看到, 除了 `bootstrap_init()` 和 `machine_init()`, 其他都是平台无关的, 因此这里主要介绍平台相关的启动流程.

其中, `machine_init()` 的发生在 `mm::init()` 后, 这是因为 `machine_init` 中都是关于

```

pub fn bootstrap_init() {
    // ECfg: 设置允许时钟中断触发
    ECfg::empty()
        .set_line_based_interrupt_vector(LineBasedInterrupt::TIMER)
        .write();
    EUEn::read().set_float_point_stat(true).write(); // 开启浮点模块
    // Timer & other Interrupts
    TIClr::read().clear_timer().write(); // 清除时钟
    TCfg::read().set_enable(false).write(); // 关闭时钟中断
    CrMd::read()
        .set_watchpoint_enabled(false) // 关闭Watchpoint
        .set_paging(true) // 开启页表映射模式
        // (注意: LA下页表映射和直接映射不能同时开启, 详见龙芯手册)
        .set_ie(false) // 内核台启动时默认关闭中断
        .write();
}

```

我们可以看到, 在上面的启动过程中, 龙芯实际上中断要触发有几个使能层:

1. 中断本身的使能, 来自 `ECfg`
2. 中断源的使能 (如果存在), e.g. 时钟中断来自 `TCfg`
3. `CrMd`: 当前模式寄存器的中断使能

之后是 Trap Handler 相关的设置

```
// Trap/Exception Handler initialization.
set_kernel_trap_entry(); // 设置内核态的中断入口，用于处理异常
set_machine_err_trap_ent(); // 设置内核态的机器异常入口
TLBREntry::read().set_addr(srfill as usize).write();
// TLB重填异常的入口，LoongArch下TLB miss后需要手动重填
// 相关的内容和原理见mm.pdf
```

最后是内存相关的初始化

```
// MMU Setup
// 设置无缓存的DMW2，用于驱动程序的访存。注意：驱动IO端口必须无缓存
// 保留DMW1用于RWX执行权限
DMW2::read()
    .set_plv0(true) // 允许plv0访问,其他不允许.
    .set_plv1(false) // LA下有4级特权级，其中plv0最高
    .set_plv2(false)
    .set_plv3(false) // 用户程序一般在plv3
    .set_vesg(SUC_DMW_VESG) // 设置这类地址区段号为常数SUC_DMW_VESG
    // 无缓存
    .set_mat(MemoryAccessType::StronglyOrderedUncached)
    .write();
DMW3::empty().write();
// 设置单页大小为4K页
STLBPS::read().set_ps(PTE_WIDTH_BITS).write();
// 设置
TLBREHi::read().set_page_size(PTE_WIDTH_BITS).write();

// 设置页表的大小和每级的位数，我们模拟的是Sv39，所以使用3层,4K页
// PWCL和PWCH的解释详见龙芯的官方手册
PWCL::read()
    .set_ptbase(PAGE_SIZE_BITS) // 最低一级页表，从12位开始
    .set_ptwidth(DIR_WIDTH) // 长9位
    .set_dir1_base(PAGE_SIZE_BITS + DIR_WIDTH)
    // 512*512*4096 should be enough for 256MiB of 2k500.
    .set_dir1_width(DIR_WIDTH)
    .set_dir2_base(0)
    .set_dir2_width(0)
    .set_pte_width(PTE_WIDTH) // 页表项长度为12位
    .write();
// 2k500的LoongArch的页表最高层似乎必须是dir3或者dir1,
// 不可以是dir4
PWCH::read()
    .set_dir3_base(PAGE_SIZE_BITS + DIR_WIDTH * 2)
    .set_dir3_width(DIR_WIDTH)
    .set_dir4_base(0)
    .set_dir4_width(0)
    .write();
}
```

之后是机器相关初始化，可以看到主要是时钟相关的内容。

之所以将这些内容放到这里，是因为 LoongArch 下的恒等时钟频率是

可以通过指令获取的, 如果因此我们获取后存入静态数据区域, 所以需要在 bss 段被清 0 后才开始处理时钟相关中断

```
pub fn machine_init() {  
    // 设置内核态的中断处理函数(用于不对齐读写异常等)  
    register::EEntry::read()  
        .set_exception_entry(__kern_trap as usize)  
        .write()  
    // 获取时钟频率  
    get_timer_freq_first_time();  
    // 准备, 开启时钟中断(但不在CrMd中允许时钟)  
    trap::enable_timer_interrupt();  
}
```