

计算机系统综合实验报告

计 53 师天麾 2015011305

计 53 陈秋昊 2015011283

计 53 何琦 2015011299

目录

MIPS32CPU	3
1、实验目标.....	3
2、CPU 内部设计及总线设计	3
3、物理地址划分与地址映射	4
4、各模块简述.....	5
REG_DIR	5
IF_PC_REG	5
IF2ID	5
ID	5
ID_EX	5
EX.....	5
EX2MEM.....	5
MEM.....	6
MEM2WB.....	6
SRAM.....	6
CTRL.....	6
Wishbone_bus	6
Serial.....	7
CP0	7
Vir2phy&MMU	7
ROM	7
Flash	7
VGA（HDMI）	7
5、异常处理.....	8
6、实验结果.....	8
Decaf 编译器修改	9
1、实验目标.....	9
2、指令生成修改.....	9
3、编写库函数.....	9
4、修改函数传参方式.....	9
5、实验结果.....	9
在实验板上运行单元测试.....	11
1、实验目标.....	11
2、实验过程与结果.....	11
ucore 操作系统	12

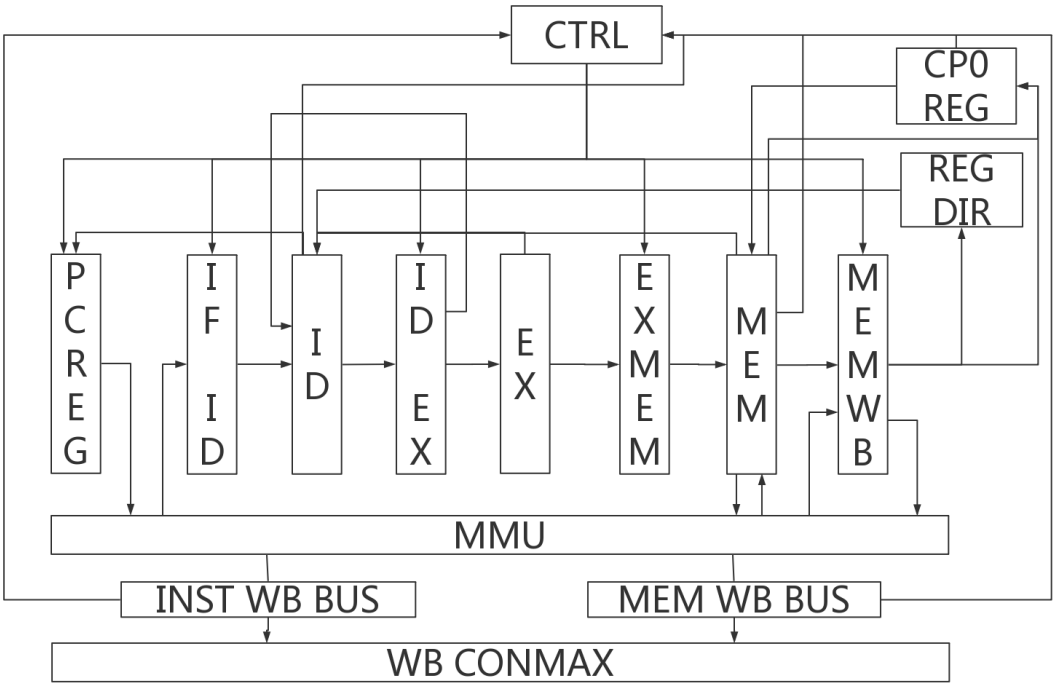
MIPS32CPU

1、实验目标

实现一个 32 位 CPU，能够执行 MIPS 指令集的一个子集（我们称之为 MIPS32S），支持中断、异常和 TLB，支持 SRAM、串口、ROM、Flash、HDMI 等外设，能够运行 32 位监控程序（<https://github.com/z4yx/supervisor-mips32>）以及 ucore 操作系统。

2、CPU 内部设计及总线设计

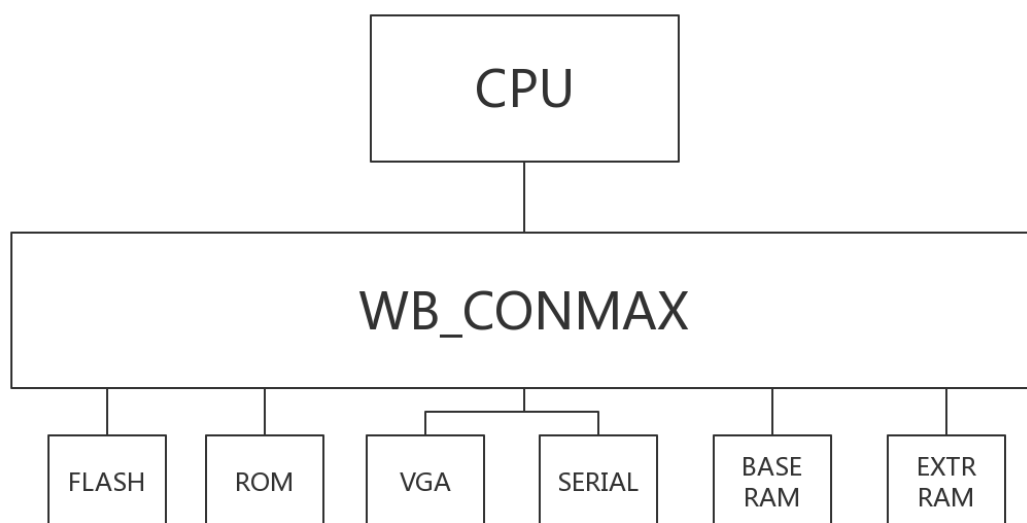
CPU 内部采用五级流水线（取指、译码、执行、访存、写回）设计，各级流水线之间设置流水线寄存器。寄存器组中包括通用寄存器、HILO 寄存器以及 CPO 寄存器。CTRL 负责 PC 的跳转、流水线的暂停与清除。MMU 负责地址映射与内存管理。图一是 CPU 内部的设计图。



图一

我们采用了 WishBone 总线协议，除了 CPU 的指令总线和数据总线外，RAM、ROM、FLASH、VGA（实际上是 HDMI）和串口也都挂在总线上，我们将在之后的章节进行详细说明。图二是总线设计图。

如图二所示，我们的 VGA（HDMI）与 SERIAL 连在一起接在总线上，这是因为我们的 VGA（HDMI）对操作系统而言是透明的，它会将串口所有读写的内容直接显示。



图二

3、物理地址划分与地址映射

设备	物理地址范围	说明
ROM	[1FC00000,1FC00FFF]	存储 BootLoader 并从此启动
RAM	[00000000,007FFFFFF]	8MB 内存
FLASH	[1E000000,1EFFFFFF]	因为 flash 仅支持 16 位的读取与存储，所以地址需要左移一位
SERIAL	[1FD003F8,1FD003FC]	1FD003F8 为数据，1FD003FC 为状态寄存器。从 3F8 读出数据会自动清除中断。
HDMI	1FD003F8	与串口共用地址，在写串口时获取数据并显示

我们按照 MIPS 规范，将整个地址空间划分为 Useg、Kseg0、Kseg1 和 Kseg2。其中 Kseg0 和 Kseg1 不使用 TLB 而是直接将高三位置 0 进行静态映射，其余区段采用查 TLB 表的方式进行，这是第一次地址映射。由于我们使用了 Wishbone 总线协议，所以在 Wishbone_bus 模块中，进行了第二次地址映射，将第一次映射得到的地址的高四位进行修改，表示要访问的从设备编号。

4、各模块简述

REG_DIR

管理 CPU 的 32 个寄存器的存储与读取，写优先。

IF_PC_REG

管理 PC 寄存器的变动，正常情况下每个时钟周期+4，当出现跳转或异常信号时跳转到相应地址。

IF2ID

将 IF 阶段得到的 PC、指令等相关数据传递给 ID，以时钟周期上升沿控制时序，并且在接收 bubble 和 flush 等控制信号时进行流水线的暂停和清空。

ID

分析指令类型，读取相应的寄存器的值并判断是否需要跳转，同时完成 instinvaId 异常的判断。

ID_EX

将 ID 阶段处理的数据传递给 EX，以时钟周期上升沿控制时序，并且在接收 bubble 和 flush 等控制信号时进行流水线的暂停和清空。

EX

根据 ID 传递而来的操作类型处理操作数据得到需要的结果，并判断地址未对齐异常。

EX2MEM

将 EX 阶段处理的数据传递给 MEM，以时钟周期上升沿控制时序，并且在接收 bubble

和 flush 等控制信号时进行流水线的暂停和清空。

MEM

此模块主要完成以下功能：

①访存。将虚拟地址传给 MMU，由 MMU 得到物理地址后传给相应外设，之后进行读写操作。同时会通过旁路将读取的数据向前传至 ID、EXE 阶段。

②构造 TLB entry，写 TLB。

③处理异常。在 MEM 之前的流水线阶段发生的异常，统一等到 MEM 阶段来进行处理，将异常信息传至 CPO 和 CTRL 模块进行处理。

MEM2WB

将 MEM 阶段的处理的数据传递给 WB 阶段，以时钟周期上升沿控制时序，并且在接收 bubble 和 flush 等控制信号时进行流水线的暂停和清空。

SRAM

此模块负责指令 RAM 和数据 RAM 的读写。由于 SRAM 的读写需要一定时间，所以我们使用了一个状态机来实现 SRAM 的读写。

CTRL

此模块负责流水线的暂停与清空。当取指、译码或访存阶段发出暂停流水线的请求时，此模块要将相应的部分流水线暂停。当发生异常时，此模块要将相应部分的流水线清空以实现精确异常处理，同时要将 pc 值设置为要跳转到的异常处理程序的入口地址。

Wishbone_bus

此模块负责主设备（在我们的 cpu 中，也就是指令总线和数据总线）向总线发送请求。cpu 想访问任何总线上的从设备，都需要通过 Wishbone_bus 来向总线发送请求。在请求成功完成前（得到 ack 前），Wishbone_bus 都会向 CTRL 模块发送信号，暂停之后的流水线。

在这个模块中，也进行了第二次地址映射，将物理地址的高四位修改，表示接下来要访问的从设备编号。

Serial

串口模块，控制串口的读写，若写繁忙则会等待，读入后会清空数据，同时提供串口的状态信息。使用直连串口信号控制串口的读写。

CP0

此模块实现 cp0 寄存器以及对它们的读写。在异常发生时，根据异常类型，此模块将相应的 cp0 寄存器和域进行填写。

Vir2phy&MMU

此模块维护了一个有 16 个 entry 的 TLB 表，根据虚拟地址所在的地址空间来转换为物理地址。

与 TLB 相关的异常也由 MMU 模块检测。

ROM

此模块实现了读取 ROM。ROM 中存储 bootloader，它可以将操作系统从 Flash 中读取并写入到 RAM 里。

Flash

此模块实现读取 Flash。每次读取出的数据只有低 16 位有效。由于 ucore 的 bootloader 实现方式比较特殊，在我的 Flash 中，会将输入的地址强制右移一位后再读取 Flash。Flash 的读取比较缓慢，所以我们用 16 个周期完成一次 Flash 的读取。

VGA（HDMI）

控制图像的输出，物理地址与串口数据共用，在写串口时获取相关的数据信息并存储显示。使用 IP 核构建显示字符的缓存，并且在 VGA（HDMI）输出时通过字体模块将字符信息转换成图像信息输出。刷屏回车等相关控制通过维护有效数据域来实现，不会将所有空白区域清空以保证该模块不会处于繁忙状态。

5、异常处理

我们的 cpu 实现了对监控程序及 ucore 所需的异常的支持。为了使异常处理的顺序与指令执行的顺序一致（而与异常产生的顺序无关），当异常发生在 MEM 之前的阶段时（比如发生在 EXE 阶段），我们只记录异常而不进行处理，在 MEM 阶段进行统一处理。

在 MEM 阶段，异常信息会被发送至 CP0 和 CTRL 模块进行处理。CP0 模块会填写某些 cp0 寄存器（的一些字段），比如 status 和 epc 寄存器；若异常与 TLB 相关，还需要填写 entryhi、badvaddr 等寄存器。CTRL 模块负责清空 MEM 阶段之前的流水线（以实现精确异常处理），同时，根据异常信息，将 pc 跳转至相应的异常处理程序入口。

6、实验结果

```
entry 0x800000f4 (phys)
etext0x8002c400 (phys)
edata0x80222690 (phys)
end0x80226170 (phys)
Kernel executable memory footprint: 2024KB
memory management: buddy_pmm_manager
memory map:
    [80000000, 80800000]

freemem start at: 00237000
free pages: 000005c9
## 00000020
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
----- BEGIN -----
----- END -----
check_slab() succeeded!
kmalloc_init() succeeded!
check_vma_struct() succeeded!
check_pgfault() succeeded!
check_vmm() succeeded.
sched class: RR_scheduler
ramdisk_init(): initrd found, magic: 0x2f8dbe2a, 0x00000fa0 secs
sfs: mount: 'simple file system' (400/100/500)
vfs: mount disk0.
kernel_execve: pid = 2, name = "sh".
user sh is running!!!
$
```

```
$ ls
0 ls [directory] 2(hlinks) 19(blocks) 4864(bytes) : 0'..'
[d] 2(h) 19(b) 4864(s) .
[d] 2(h) 19(b) 4864(s) ..
[-] 1(h) 21(b) 84936(s) pgdir
[-] 1(h) 22(b) 86361(s) math
[-] 1(h) 21(b) 84946(s) faultreadkernel
[-] 1(h) 21(b) 84940(s) faultread
[-] 1(h) 21(b) 85999(s) fibonacci
[-] 1(h) 21(b) 84937(s) badarg
[-] 1(h) 21(b) 84936(s) yield
[-] 1(h) 21(b) 85260(s) ls
[-] 1(h) 22(b) 89249(s) sh
[-] 1(h) 21(b) 84936(s) hello
[-] 1(h) 21(b) 85131(s) pwd
[-] 1(h) 21(b) 84982(s) cat
[-] 1(h) 21(b) 84965(s) forktest
[-] 1(h) 1(b) 21(s) test.txt
[-] 1(h) 21(b) 84957(s) exit
[-] 1(h) 26(b) 105831(s) blackjack
[-] 1(h) 21(b) 84959(s) sleep
lsdir: step 4
```


Decaf 编译器修改

1、实验目标

修改 decaf 编译器，使其能够编译出可以运行在我们的 MIPS32CPU 和 ucore 操作系统上的代码。

2、指令生成修改

在 Mips 类中将原来生成的汇编代码中的 add、sub 指令修改为 addu，subu，将 mul 拆分为 mult 和 mflo，以适应 CPU。同时在 TransPass2 类中将 div、rem 翻译成相应的库函数调用。

3、编写库函数

我们使用 C 语言编写了 PrintString、ReadLine、DIV 和 REM 等 9 个库函数。将其与 decaf 生成的汇编程序链接即可。

4、修改函数传参方式

由于 decaf 内部函数调用时使用栈传参，而 ucore 使用寄存器传参。所以我们将 decaf 调用我们编写的库函数时，在库函数外面包了一层函数，现将栈里的参数拿出来放到寄存器中，然后再调用相应的库函数。

5、实验结果

可以在 ucore 操作系统里成功运行测试程序 math、fibonacci 和 blackjack。如下图所示：

```
$ math
1
8
4
2
1
```

```
$ fibonacci
```

```
1
1
2
3
5
8
13
21
34
55
89
10946
```

```
Welcome to CS143 BlackJack!
```

```
-----
Please enter a random number seed: 100
```

```
Shuffling...done.
```

```
How many players do we have today? 2
```

```
What is the name of player #1? cqh
```

```
What is the name of player #2? sth
```

```
First, let's take bets.
```

```
cqh, you have $1000.
```

```
How much would you like to bet? 1000
```

```
sth, you have $1000.
```

```
How much would you like to bet? 1000
```

```
Dealer starts. Dealer was dealt a 3.
```

```
cqh's turn.
```

```
cqh was dealt a 11.
```

```
cqh was dealt a 10.
```

```
cqh, your total is 21.
```

```
Would you like a hit? (0=No/1=Yes) 0
```

```
cqh stays at 21.
```

```
sth's turn.
```

```
sth was dealt a 3.
```

```
sth was dealt a 11.
```

```
sth, your total is 14.
```

```
Would you like a hit? (0=No/1=Yes)
```

```
Would you like a hit? (0=No/1=Yes) 1
```

```
sth was dealt a 11.
```

```
sth, your total is 15.
```

```
Would you like a hit? (0=No/1=Yes) 1
```

```
sth was dealt a 10.
```

```
sth, your total is 15.
```

```
Would you like a hit? (0=No/1=Yes) 1
```

```
sth was dealt a 10.
```

```
sth busts with the big 25!
```

```
Dealer's turn.
```

```
Dealer was dealt a 5.
```

```
Dealer was dealt a 8.
```

```
Dealer was dealt a 10.
```

```
Dealer busts with the big 26!
```

```
Time to resolve bets.
```

```
cqh, you won $1000.
```

```
sth, you lost $1000.
```

```
Do you want to play another hand? (0=No/1=Yes) 0
```

```
cqh, you have $3000.
```

```
sth, you have $0.
```

```
Thank you for playing...come again soon.
```

```
CS143 BlackJack Copyright (c) 1999 by Peter Mork.
```

```
(2001 mods by jdz)
```

在实验板上运行单元测试

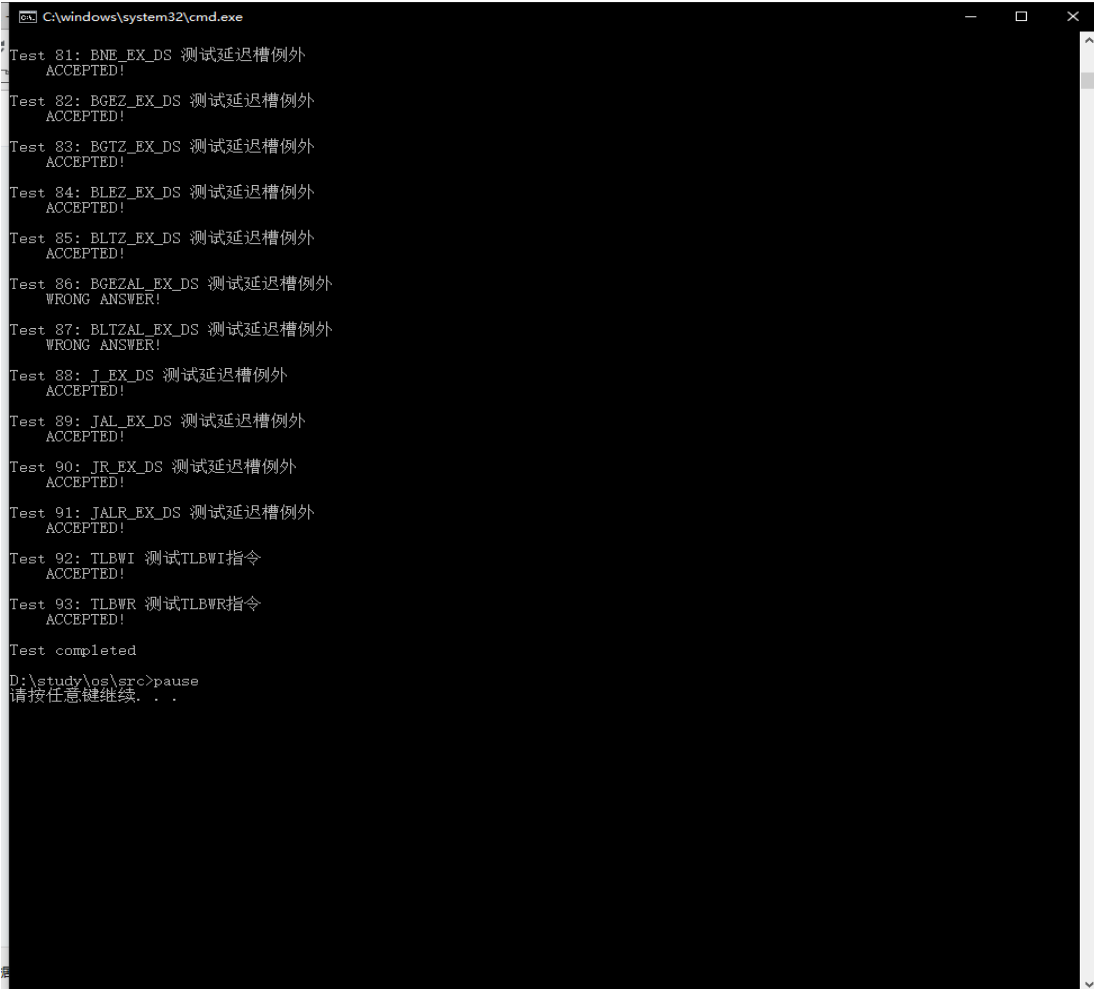
1、实验目标

将课程提供的单元测试（<https://github.com/oscourse-tsinghua/cpu-testcase>）在MIPS32CPU上运行。

2、实验过程与结果

在部分单元测试中，用到了 `add` 和 `addi` 指令，我们并没有实现这两条指令。又由于在该单元测试中并没有用到这两条指令会报算术溢出异常的特性，所以我们可以将其修改为 `addu` 和 `addiu`，即可通过该单元测试。

我们自己编写了一个脚本，将测试结果可视化，如下图（一部分单元测试）所示：



```
C:\windows\system32\cmd.exe
Test 81: BNE_EX_DS 测试延迟槽例外
ACCEPTED!
Test 82: BGEZ_EX_DS 测试延迟槽例外
ACCEPTED!
Test 83: BGTZ_EX_DS 测试延迟槽例外
ACCEPTED!
Test 84: BLEZ_EX_DS 测试延迟槽例外
ACCEPTED!
Test 85: BLTZ_EX_DS 测试延迟槽例外
ACCEPTED!
Test 86: BGEZAL_EX_DS 测试延迟槽例外
WRONG ANSWER!
Test 87: BLTZAL_EX_DS 测试延迟槽例外
WRONG ANSWER!
Test 88: J_EX_DS 测试延迟槽例外
ACCEPTED!
Test 89: JAL_EX_DS 测试延迟槽例外
ACCEPTED!
Test 90: JR_EX_DS 测试延迟槽例外
ACCEPTED!
Test 91: JALR_EX_DS 测试延迟槽例外
ACCEPTED!
Test 92: TLBWI 测试TLBWI指令
ACCEPTED!
Test 93: TLBWR 测试TLBWR指令
ACCEPTED!
Test completed
D:\study\os\src>pause
请按任意键继续. . .
```

其中，错误的单元测试是由于我们没有实现相应指令。

ucore 操作系统

QEMU-THUMIPS Setup

```
git clone git@github.com:chyh1990/qemu-thumips.git
cd qemu-thumips
./configure --python=python2
make
```

If docs are not able to be built, add `*--disable-docs*` to configure.

If `*timer_gettime()*` link error, add `*-lz -lrt*` to Makefile.target, line 37:

```
LIBS+=-lz -lrt -lm
```

If `*error: field 'info' has incomplete type*`, replace `*struct siginfo*` with `*siginfo_t*` in linux-user/signal.c, line 3247, 3466, 3468.

After the make FAILED, execute the following command:

```
cd mipsel-softmmu
make
```

And you will get `*qemu-system-mipsel*` here. (I don't know whether it is runnable.)

UCORE-THUMIPS Setup

Better to keep the git repo next to the `*qemu-thumips*` repo.

Do not use other versions of gcc, use `*mips-sde-elf-*` only.

Change `*run*` with the correct location of `*qemu-system-mipsel*`, as for mine, it's

```
../qemu-thumips/mipsel-softmmu/qemu-system-mipsel -M mipssim -m 32M -serial
stdio -bios boot/loader.bin
```

Then the ucore will be able to compile. However, several ops are not supported in our default thumips-insn.txt, so we add them:

```
MULT 000000ssssssttttt00000000000011000
```

Apply this patch will remove a warning:

<https://git.qemu.org/?p=qemu.git;a=commitdiff;h=98cf48f60aa4999f5b2808569a193a401a390e6a>

GDB

gdb-mips-linux-gnu has a broken dependency of libpython2.6. So I chose to compile it.

```
./configure --target=mips-linux
```

LAB8 to MIPS

boot

boot/bootasm.S: Rewrite in MIPS

kern

debug/: Replace `cprintf` with `kprintf`

driver/: Completely rewrite with respect to `mipsregs.h` and `thumips.h`

fs/: Replace `ROUNDDOWN` macro with `ROUNDDOWN_2N`, and define `SHIFT` together with `SIZE`

include/: Remove complex support.

init/: Rewrite `entry.S` with MIPS, simplify `init.c` (using `tlb_invalidate_all()`)

libs/: Used MIPS library.

mm/: Use `thumips_tlb`, changed `memlayout.h` due to hardware.

proc/: Rewrite `entry.S` with MIPS, rewrite `switch.S` for the new trapframe, modify `proc.c` for trapframe.

schedule/: Fallback to naive sched.

sync/: Removed monitor and check for LAB7

syscall/: Modify registers.

trap/: Setup new `vectors.S`, rewrite `exception.S` and replace the old `trapentry.S`. Use new `mips_trapframe`.

user

libs/: Add `intrinsic.c` & `entry.S` for decaf support.