华中科技大学

2019

系统能力综合训练　课程设计报告

题　　目：	X86 模拟器设计
专　　业：	计算机科学与技术
班　　级：	CS1601
学　　号：	U201614531
姓　　名：	刘本嵩
电　　话：	+1 (669) 266-7699
邮　　件：	root@recolic.org
完成日期：	2020-01-02

计算机科学与技术学院

目录

1　课程设计概述

1.1 课设目的

理解"程序如何在计算机上运行"的根本途径是从"零"开始实现一个完整的计算机系统. 南京大学计算机科学与技术系计算机系统基础课程的小型项(Programming Assignment, PA)将提出 x86 架构的一个教学版子集 n86，指导学生实现一个功能完备的 n86 模拟器 NEMU(NJU EMUlator)，最终在 NEMU 上运行游戏"仙剑奇侠传"，来让学生探究"程序在计算机上运行"的基本原理. NEMU 受到了 QEMU 的启发，并去除了大量与课程内容差异较大的部分. PA 包括一个准备实验(配置实验环境)以及 5 部分连贯的实验内容:

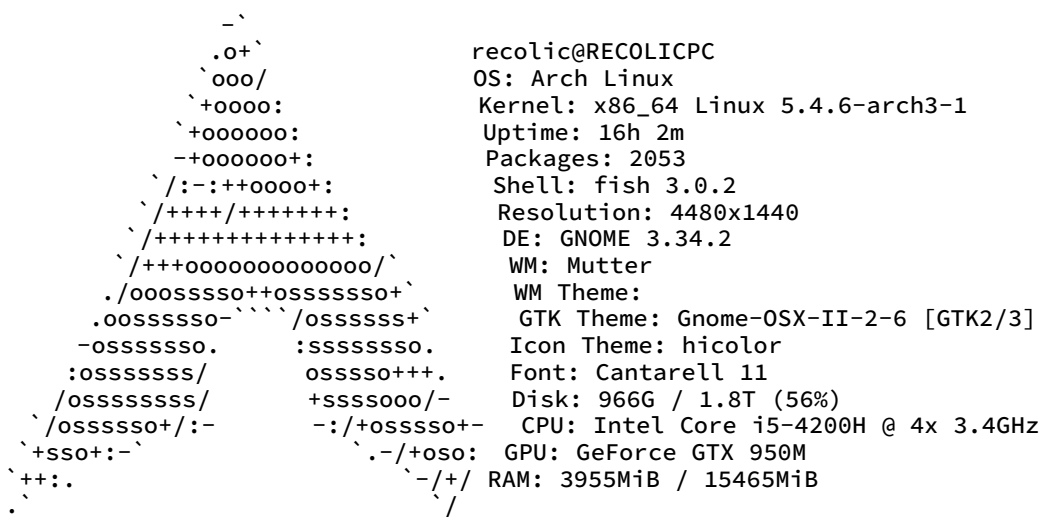
1.2 课设任务

1. 配置开发环境。

2. 实现 debugger 的基础功能，例如 x p w d c n 等指令。实现表达式的 evaluation，实现 watchpoint 等功能。

3. 依次实现 i386 手册规定的大多数指令，实现一个精简版的 libc，实现设备管理相关功能，实现时间和 GUI 绘图，以便运行打字小游戏、性能测试、超级马里奥游戏等 x86 的 app。

1.3 实验环境


recolic@RECOLICPC
OS: Arch Linux
Kernel: x86_64 Linux 5.4.6-arch3-1
Uptime: 16h 2m
Packages: 2053
Shell: fish 3.0.2
Resolution: 4480x1440
DE: GNOME 3.34.2
WM: Mutter
WM Theme:
GTK Theme: Gnome-OSX-II-2-6 [GTK2/3]
Icon Theme: hicolor
Font: Cantarell 11
Disk: 966G / 1.8T (56%)
CPU: Intel Core i5-4200H @ 4x 3.4GHz
GPU: GeForce GTX 950M
RAM: 3955MiB / 15465MiB

2 实验过程

2.1 PA0

我有一台在 2016 年安装好的 Arch Linux 计算机，其中早已配置好所有必要的开发环境，因此 PA0 已经完成。

2.2 PA1

2.2.1 总体设计

我做了以下几件事:

1. 修改 Makefile 和提交脚本,允许用户透明化的设置自己使用的反向代理 host。包括 git(ssh)和 https 支持。

2. 改造框架代码和 Makefile，将 nemu 的 C 代码全部转换为 C++。

3. 修改 Makefile，将他在每次 Make 后自动生成一个无意义 commit 的行为改为，可以使用 make git-gen M="Some thing."这种命令来手动控制 commit 生成。修改了脚本的其他无意义的举动。

4. 修改其他配套 script，因为老师提供的所有 shell script 在关键位置没有任何错误检查。这会导致学生在反代环境的不稳定网络下体验极差。

5. 搭建校园网反向代理服务，提供反向代理服务文档。校园网内的一个闲置 Linux 路由器(武汉)，proxy.recolic.net(香港)，git.recolic.net(彰化縣)，storage.recolic.net(法国巴黎)，共同支持了此服务的运行。

6. 增加public-template分支,这允许用户不再面对我已经解决过的问题。同时，这允许 imzhwk 等用户更方便的二次开发出 Docker 镜像。

7. 改造框架代码,优化 common.h 中不合理的命名,引入自己实现的 C++ 库(rlib)，修改不合理的 gitignore。

8. 实现 PA1 要求的 Debugger 功能。

2.2.2 详细设计

PA1 的详细功能中，首先修改 reg.h，增加需要的 9 个 32bit 寄存器。

实现 info r 和 x 指令，打印寄存器和内存。寄存器可以直接使用 rlib::printfln()和 C++ iomanip 库进行打印。内存则直接访问 vaddr_read 即可。

```
static int cmd_info(char *_args) {
    if(_args == NULL)
        throw std::runtime_error("Usage: info <what>");
    if(string(_args).strip() == "r") {
        println("Registers:");
        printfln("%eax={}, %ebx={}, %ecx={}, %edx={}",
dumpReg(cpu.eax), dumpReg(cpu.ebx), dumpReg(cpu.ecx),
dumpReg(cpu.edx));
        printfln("%esp={}, %ebp={}, %esi={}, %edi={}",
dumpReg(cpu.esp), dumpReg(cpu.ebp), dumpReg(cpu.esi),
dumpReg(cpu.edi));
        printfln("%eip={}, CF/OF/SF/ZF/DF={}/{}/{}/{}/{}",
dumpReg(cpu.eip), cpu_eflags::get<cpu_eflags::CF>(),
cpu_eflags::get<cpu_eflags::OF>(),
cpu_eflags::get<cpu_eflags::SF>(),
cpu_eflags::get<cpu_eflags::ZF>(),
cpu_eflags::get<cpu_eflags::DF>());
    }
    else if(string(_args).strip() == "w") {
        println("Watchpoints:");
        println(watchpoints);
    }
    return 0;
}
```

实现一个表达式求值工具。这很简单，我使用 bison 配合 flex，并稍微修

改 Makefile，使主 Makefile 在拉取 cc 源文件列表前，先运行这个子目录的 generate 目标。在 bison 的帮助下，我在几十分钟内完成了它。一次通过。

```
%%
entry: {*result = 0;}
| expr {*result = $1;}
;

expr: T_INT { $$ = $1; }
| expr '+' expr { $$ = $1 + $3; }
| expr '-' expr { $$ = $1 - $3; }
| expr '*' expr { $$ = $1 * $3; }
| expr '/' expr { $$ = $1 / $3; }
| expr T_EQUAL expr { $$ = ($1 == $3); }
| expr T_NEQUAL expr { $$ = ($1 != $3); }
| expr T_LOGICAL_AND expr { $$ = ($1 && $3); }
| '(' expr ')' { $$ = $2; }
| '-' expr { $$ = 0 - $2; }
| '*' expr { $$ = ryy_read_memory_value((uint32_t)$2); }
| T_REG { $$ = ryy_read_cpu_reg_value($1); }
;
%%
```

然后要实现 watchpoints。本来以为文档里说的 watchpoint 池是一个池，没想到只是一个普通的 list。可惜 rlib 里有一个已经实现好的通用对象池。这样直接实现 watchpoint 类然后用 std::list 装起来即可。

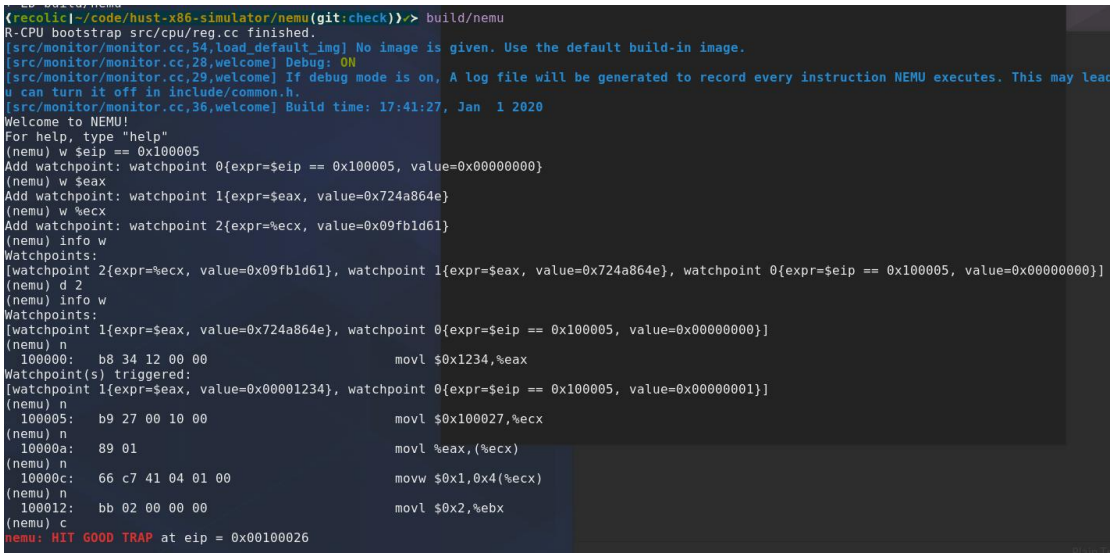
```
struct WP {
    std::string expr;
    uint32_t curr_value;
    int id;

    bool evalulate() {
        auto new_value = evaluate_expr(expr);
        std::swap(new_value, curr_value);
        return new_value != curr_value && curr_value != 0;
    }
    WP(std::string e, int id) : expr(e), id(id) {
        evalulate(); // initial expr value.
    }
};
```

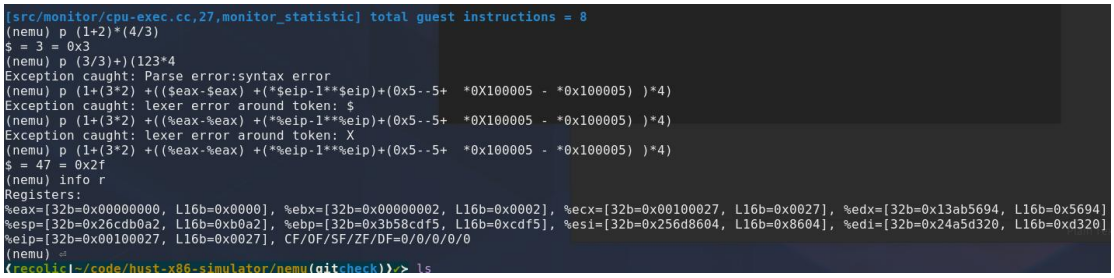
```
inline std::list<WP> watchpoints;
// WARNING: Not thread-safe
inline int max_watchpoint_id;
```

```
inline std::ostream & operator<< (std::ostream &os, const WP
&watchpoint) {
    return os << "watchpoint " << watchpoint.id << "{expr=" <<
watchpoint.expr << ", value=" << num2hex(watchpoint.curr_value) <<
"}";
}
```

值得注意的是，这里我有几个与约定不符的地方，例如我用 n 而不是 si 表示单步执行，用%eax 而不是$eax 表示寄存器，用 1 表示第一个 watchpoint 而不是 0，在下边缘也会触发 watchpoint，不支持大写的 0X100005。这些修改补丁我都在 pa2 这个分支(最终检查之前)才增加(同时支持新旧写法)。

2.2.3 运行结果





2.2.4 问题解答

1. 理解基础设施

由于问题不清晰，我们这里假设：每次”用于调试”的编译中，仅获取并分析一个（而不是 0 个或多个）信息。

则，你会在调试上花费 500*0.9*30s = 13500s 时间。

简易调试器可以节约 500*0.9*20s = 9000s 时间。

2. 查阅 i386 手册

Q: EFLAGS 寄存器中的 CF 位是什么意思?

A: Page 419 of 421, Appendix C Status Flag Summary:
0 -- CF -- Carry Flag -- Set on high-order bit carry or borrow; cleared otherwise.

Q: ModR/M 字节是什么?

A: Page 241-242 of 421, Section 17.2.1 ModR/M and SIB Bytes
The ModR/M byte contains three fields of information:
● The mod field, which occupies the two most significant bits of the byte, combines with the r/m field to form 32 possible values: eight registers and 24 indexing modes.
● The reg field, which occupies the next three bits following the mod field, specifies either a register number or three more bits of opcode information. The meaning of the reg field is determined by the first (opcode) byte of the instruction.
● The r/m field, which occupies the three least significant bits of the byte, can specify a register as the location of an operand, or can form part of the addressing-mode encoding in combination with the field as described above.

Q: mov 指令的具体格式是怎么样的?

A: Page 345-347 of 421

MOV — Move Data

Opcode	Instruction	Clocks	Description
88 /r	MOV r/m8,r8	2/2	Move byte register to r/m byte
89 /r	MOV r/m16,r16	2/2	Move word register to r/m word
89 /r	MOV r/m32,r32	2/2	Move dword register to r/m dword
8A /r	MOV r8,r/m8	2/4	Move r/m byte to byte register
8B /r	MOV r16,r/m16	2/4	Move r/m word to word register
8B /r	MOV r32,r/m32	2/4	Move r/m dword to dword register
8C /r	MOV r/m16,Sreg	2/2	Move segment register to r/m word
8D /r	MOV Sreg,r/m16	2/5,pm=18/19	Move r/m word to segment register
A0	MOV AL,moffs8	4	Move byte at (seg:offset) to AL
A1	MOV AX,moffs16	4	Move word at (seg:offset) to AX
A1	MOV EAX,moffs32	4	Move dword at (seg:offset) to EAX
A2	MOV moffs8,AL	2	Move AL to (seg:offset)
A3	MOV moffs16,AX	2	Move AX to (seg:offset)
A3	MOV moffs32,EAX	2	Move EAX to (seg:offset)
B0 + rb	MOV reg8,imm8	2	Move immediate byte to register
B8 + rw	MOV reg16,imm16	2	Move immediate word to register
B8 + rd	MOV reg32,imm32	2	Move immediate dword to register
Ciiiiii	MOV r/m8,imm8	2/2	Move immediate byte to r/m byte
C7	MOV r/m16,imm16	2/2	Move immediate word to r/m word
C7	MOV r/m32,imm32	2/2	Move immediate dword to r/m dword

MOV — Move to/from Special Registers

Opcode	Instruction	Clocks	Description
0F 20 /r	MOV r32,CR0/CR2/CR3	6	Move (control register) to (register)
0F 22 /r	MOV CR0/CR2/CR3,r32	10/4/5	Move (register) to (control register)
0F 21 /r	MOV r32,DR0 -- 3	22	Move (debug register) to (register)
0F 21 /r	MOV r32,DR6/DR7	14	Move (debug register) to (register)
0F 23 /r	MOV DR0 -- 3,r32	22	Move (register) to (debug register)
0F 23 /r	MOV DR6/DR7,r32	16	Move (register) to (debug register)
0F 24 /r	MOV r32,TR6/TR7	12	Move (test register) to (register)
0F 26 /r	MOV TR6/TR7,r32	12	Move (register) to (test register)

3. shell 命令

声明：回答此问题使用的 PA1 指 hash 为 99f069f03c4d34824bf1af5bc2eb3fd3a608f5de 的 commit。未来 PA1 的 HEAD 指针的移动不影响此题答案。

1) *.c 的文件的代码数。

由于此工程的 nemu 部分已经被更换为 C++，因此.c 文件中**共有 0 行代码。除去空行后，共有 0 行代码。**

2) *.h 的文件的代码数。

为了统计.h 文件的代码数，先使用命令

```
find . -name '*.h' -exec cp '{}' ~/tmp/hust/ ';'
```

把所有 h 文件从工程目录中移出，防止.hpp 文件和.cc 文件的干扰。然后使用 sloc 工具，结果如下。(共有 **950** 行代码，除去空行后，共有 **760** 行代码)

Language	Files	Code	Comment	Blank	Total
Total	24	675	90	190	950
C	24	675	90	190	950

3) 真正实用的代码数统计。

直接在 nemu 目录下使用 sloc 工具，结果如下。

Language	Files	Code	Comment	Blank	Total
Total	70	5533	1156	1520	7957
C++	37	4581	1046	1250	6631
C	25	701	94	200	990
YACC	1	91	2	20	112
Make	4	87	12	35	134
Shell	1	33	1	6	40
Markdown	1	24	0	2	26
Lex	1	16	1	7	24

4. 使用 man

1) gcc 9.2.0 中-Wall 的作用如下:

```
        This enables all the warnings about constructions that some users consider questionable,
and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros.
This also enables some language-specific warnings
        described in C++ Dialect Options and Objective-C and Objective-C++ Dialect Options.

        -Wall turns on the following warning flags:

        -Waddress -Warray-bounds=1 (only with -O2) -Wbool-compare -Wbool-operation
-Wc++11-compat -Wc++14-compat -Wcatch-value (C++ and Objective-C++ only) -Wchar-subscripts
-Wcomment -Wduplicate-decl-specifier (C and Objective-C only)
        -Wenum-compare (in C/ObjC; this is on by default in C++) -Wformat -Wint-in-bool-context
-Wimplicit (C and Objective-C only) -Wimplicit-int (C and Objective-C only)
-Wimplicit-function-declaration (C and Objective-C only) -Winit-self
        (only for C++) -Wlogical-not-parentheses -Wmain (only for C/ObjC and unless
-ffreestanding) -Wmaybe-uninitialized -Wmemset-elt-size -Wmemset-transposed-args
-Wmisleading-indentation (only for C/C++) -Wmissing-attributes -Wmissing-braces
        (only for C/ObjC) -Wmultistatement-macros -Wnarrowing (only for C++) -Wnonnull
```

-Wnonnull-compare -Wopenmp-simd -Wparentheses -Wpointer-sign -Wreorder -Wrestrict -Wreturn-type -Wsequence-point -Wsign-compare (only in C++)

-Wsizeof-pointer-div -Wsizeof-pointer-memaccess -Wstrict-aliasing -Wstrict-overflow=1 -Wstringop-truncation -Wswitch -Wtautological-compare -Wtrigraphs -Wuninitialized -Wunknown-pragmas -Wunused-function -Wunused-label -Wunused-value

-Wunused-variable -Wvolatile-register-var

Note that some warning flags are not implied by -Wall. Some of them warn about constructions that users generally do not consider questionable, but which occasionally you might wish to check for; others warn about constructions that

are necessary or hard to avoid in some cases, and there is no simple way to modify the code to suppress the warning. Some of them are enabled by -Wextra but many of them must be enabled individually.

2) gcc 9.2.0 中-Werror 的作用如下:

Make all warnings into errors.

3) 为什么要使用-Wall 和-Werror?

-Wall 有利于及时发现代码风格的部分不严谨之处,并在当次提交之前得以改正。

我反对-Werror 的使用，我在此实验中也没有使用此选项。相比于冗长无趣且浪费纸张的抱怨，我选择直接引用这位开发者的博客。我的观点与他相同。

https://embeddedartistry.com/blog/2017/05/22/werror-is-not-your-friend/

2.3 PA2

2.3.1 总体设计

我做了以下几件事:

1. 修改 runall 脚本，允许更好的日志报告机制。

2. 修改 Makefile 和 icc 编译校本，允许一键使用 icc 进行较优化的编译，包括指导 icc 在编译的过程中尝试运行代码,以便根据 benchmark 运行结果反过来优化编译过程。仅在编译方法的修改上, 这可以减少 40%的代码运行时间。

3. 增加 gitlab-ci，在每次提交后和 Merge Request 中，我的 Gitlab

Runner 会自动检查我的代码是否能通过 gcc 的编译、icc 的编译、和 runall.sh 的自动测试。并直观的显示在 gitlab 界面上。

4. 改造框架代码，将 EHelper、DHelper 等不恰当的宏使用，改为更优的 namespace 方案。增加重载，允许 rlib::print 家族函数直接打印 Operand 等结构体。

5. 依次增加实验需要用到的指令，并通过所有的 CPU test。

6. 依次实现实验需要用到的设备函数和 libc 函数，并通过 time test，video test，和其他 benchmark 和超级马里奥。

7. 根据 Intel Vtune 给出的性能分析结果，改造框架代码，优化热点代码，并采用多线程异步渲染画面。这可以更有效的利用 CPU 时间，额外减少约 30% 的代码运行时间。

2.3.2 详细设计

值得注意的是，用函数名拼接的方法实现 namespace 的功能给 IDE 开发和 debug 带来了困难，因此我把 DHelper 和 EHelper 都装进了 C++原生的 namespace。

```
#define make_EHelper(name) void name(vaddr_t *eip)
namespace EHelperImpl {
make_EHelper(mov);
make_EHelper(...);
}
```

依次添加了 CF OF SF ZF DF 寄存器，实现所有汇编指令。同时实现 IO 相关函数，在 paddr_read/write 函数中加入 MMIO 相关逻辑。

```
namespace cpu_eflags {
    enum {CF = 0, PF, AF, ZF, SF, OF, DF, SIZE_OF_EFLAGS};
    template <size_t Which>
    inline bool &get() {
```

```
        static bool actual_buffer[SIZE_OF_EFLAGS];
        return actual_buffer[Which];
    }
}

__attribute__((hot)) uint32_t paddr_read(paddr_t addr, int len) {
    static const uint32_t niddle[] = {0, 0xff, 0xffff, 0xffffff,
0xffffffff};

#ifndef DISABLE_MMIO
    if(const auto mmio_id = is_mmio(addr); RLIB_MACRO_LIKELY(-1 ==
mmio_id)) {
#endif
        return pmem_rw(addr, uint32_t) & niddle[len];
#ifndef DISABLE_MMIO
    }
    else {
        return mmio_read(addr, len, mmio_id);
    }
#endif
}
```

修改 device_update，将性能影响较大的 device 更新操作放入新线程。

```
void device_update() {
#ifdef ENABLE_ASYNC_RENDER
#else
    if(device_update_flag.exchange(false)) {
        device_update_impl();
    }
#endif
}

[[maybe_unused]] static void device_update_thread_daemon() {
    while(true) {
        if(device_update_flag.exchange(false)) {
            device_update_impl();
        }
        // At most, 1000FPS
      std::this_thread::sleep_for(std::chrono::milliseconds(1));
    }
}
```

修复 SDL 多线程问题，将 SDL 初始化操作也移入更新线程。

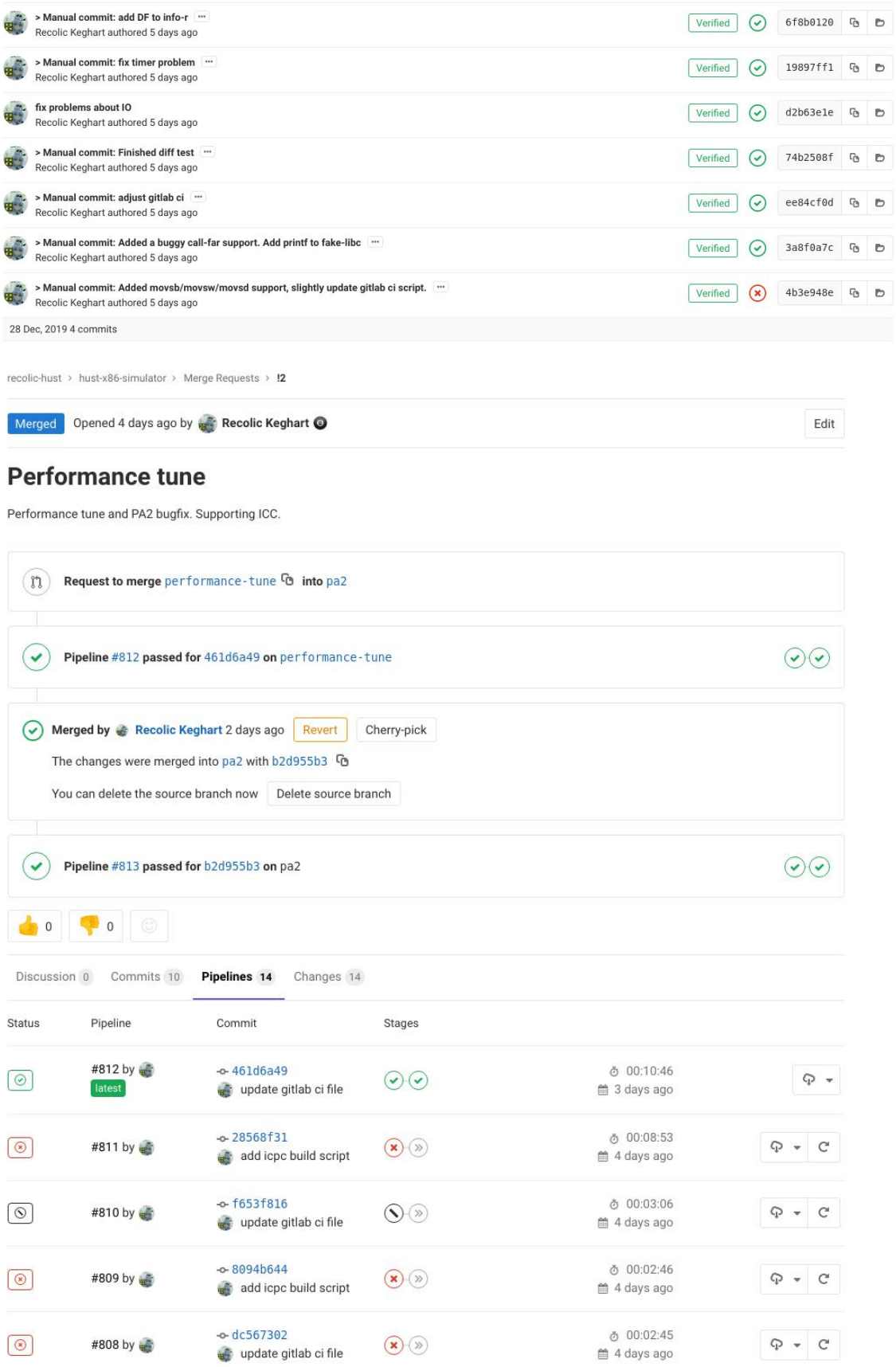
```
void update_screen() {
#ifndef DISABLE_MMIO
#ifdef ENABLE_ASYNC_RENDER
    if(window == nullptr) init_vga_impl();
#endif
    SDL_ErrorCheck(SDL_UpdateTexture(texture, NULL, vmem, SCREEN_W
* sizeof(vmem[0][0])));
    SDL_ErrorCheck(SDL_RenderClear(renderer));
    SDL_ErrorCheck(SDL_RenderCopy(renderer, texture, NULL, NULL));
    SDL_RenderPresent(renderer);
#endif
}

void init_vga() {
#ifdef ENABLE_ASYNC_RENDER
// Because of fucking SDL design, vga_init should be done in updating
thread.
// Do nothing in main thread.
#else
    init_vga_impl();
#endif
}
```

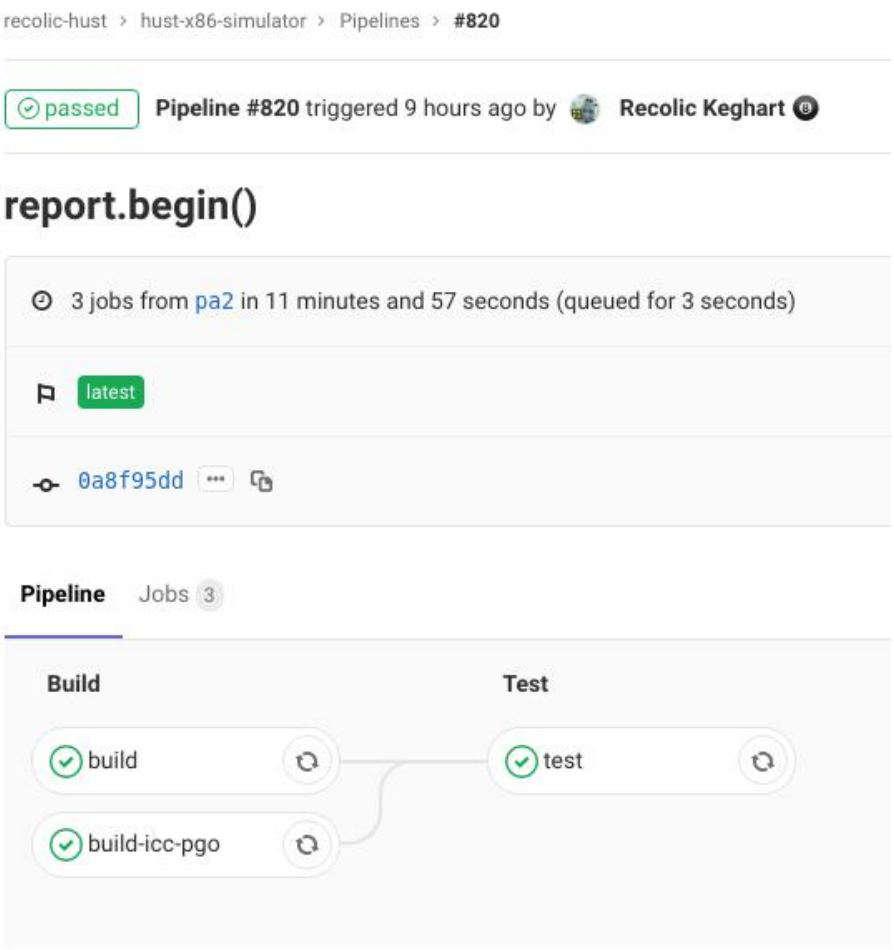
实现了时钟和 IO 等操作。

值得注意的是，我的 libc 的 string 实现直接移植了 glibc，memcpy 等操作来自 uclibc，printf 的实现则来自另一个嵌入式设备的 libc 实现。这些开源实现在提升性能的同时，也迫使我额外实现了 movs 等字符串操作汇编指令。

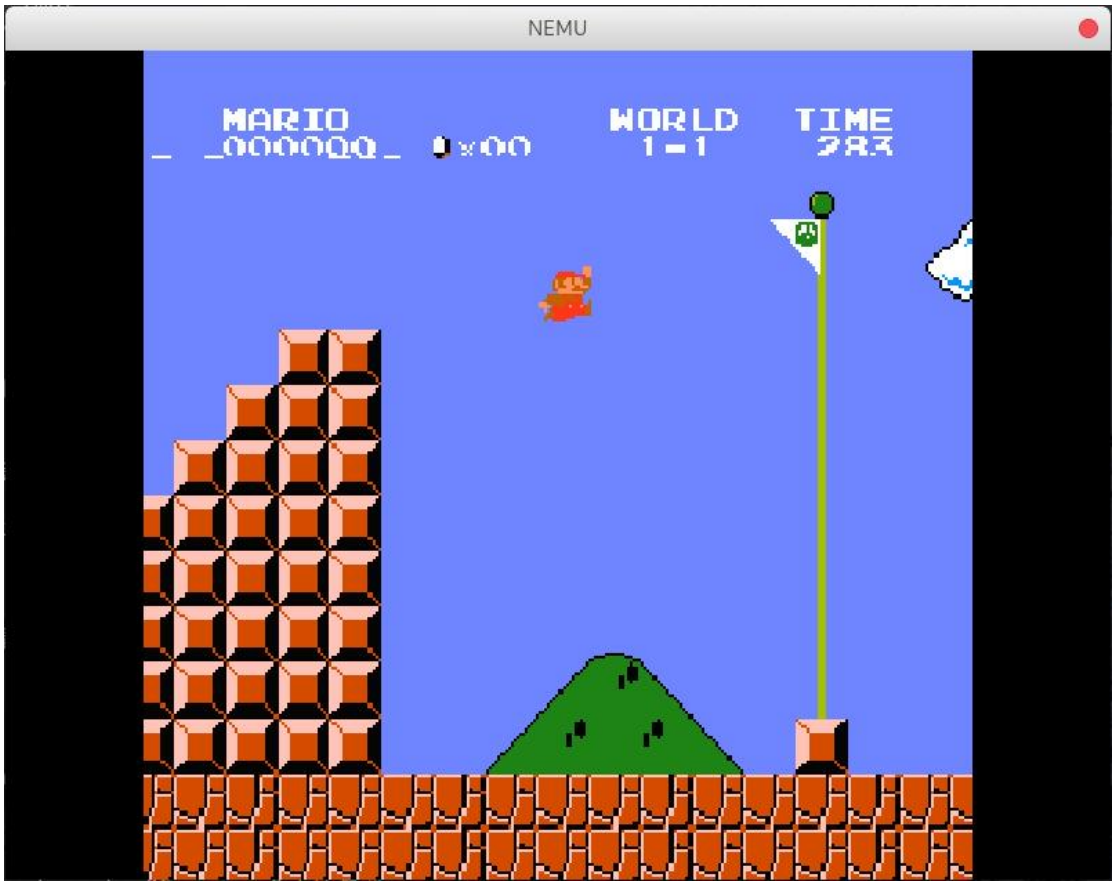
2.3.3 运行结果

值得注意的是，CI 提供的，在另一台全新的机器上的自动测试，对自动发现代码的问题有很大作用，我因此发现了关于 icc、开发环境和依赖库的很多问题。


> Manual commit: add DF to info-r
Recolic Keghart authored 5 days ago
Verified
6f8b0120
> Manual commit: fix timer problem
Recolic Keghart authored 5 days ago
Verified
19897ff1
fix problems about IO
Recolic Keghart authored 5 days ago
Verified
d2b63e1e
> Manual commit: Finished diff test
Recolic Keghart authored 5 days ago
Verified
74b2508f
> Manual commit: adjust gitlab ci
Recolic Keghart authored 5 days ago
Verified
ee84cf0d
> Manual commit: Added a buggy call-far support. Add printf to fake-libc
Recolic Keghart authored 5 days ago
Verified
3a8f0a7c
> Manual commit: Added movsb/movsw/movsd support, slightly update gitlab ci script.
Recolic Keghart authored 5 days ago
Verified
4b3e948e
28 Dec, 2019 4 commits
recolic-hust › hust-x86-simulator › Merge Requests › !2
Merged
Opened 4 days ago by Recolic Keghart
Edit
Performance tune
Performance tune and PA2 bugfix. Supporting ICC.
Request to merge performance-tune into pa2
Pipeline #812 passed for 461d6a49 on performance-tune
Merged by Recolic Keghart 2 days ago
Revert
Cherry-pick
The changes were merged into pa2 with b2d955b3
You can delete the source branch now
Delete source branch
Pipeline #813 passed for b2d955b3 on pa2
0
0
Discussion 0
Commits 10
Pipelines 14
Changes 14
Status
Pipeline
Commit
Stages
#812 by
latest
461d6a49
update gitlab ci file
00:10:46
3 days ago
#811 by
28568f31
add icpc build script
00:08:53
4 days ago
#810 by
f653f816
update gitlab ci file
00:03:06
4 days ago
#809 by
8094b644
add icpc build script
00:02:46
4 days ago
#808 by
dc567302
update gitlab ci file
00:02:45
4 days ago



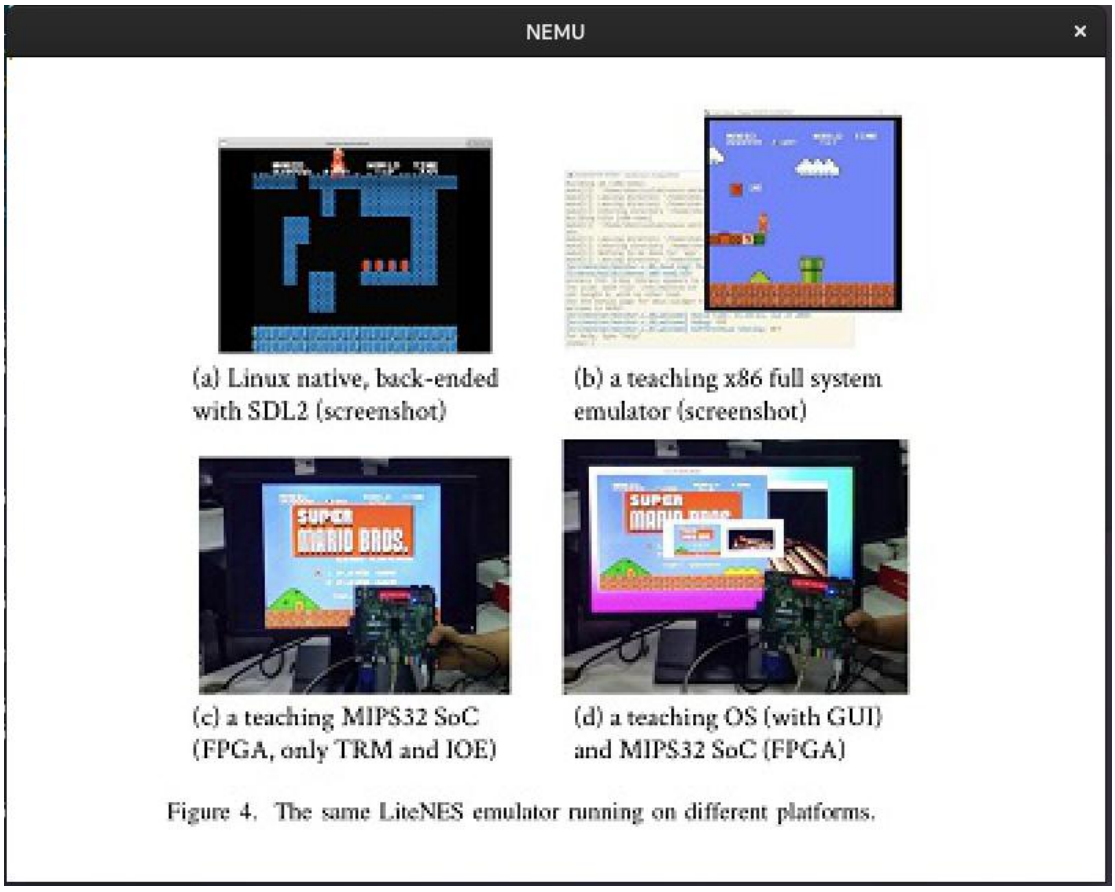
下面是代码的运行结果。图为超级马里奥游戏过程。


NEMU
MARIO
WORLD
TIME
1-1


NEMU
MARIO
WORLD
TIME

下面是 micro benchmark 的结果，781 分的性能勉强支撑起超级马里奥的游戏体验(稳定约 4fps，至少能玩，并且有一定乐趣)。

下面是其他几个 test 的结果，依次展示。



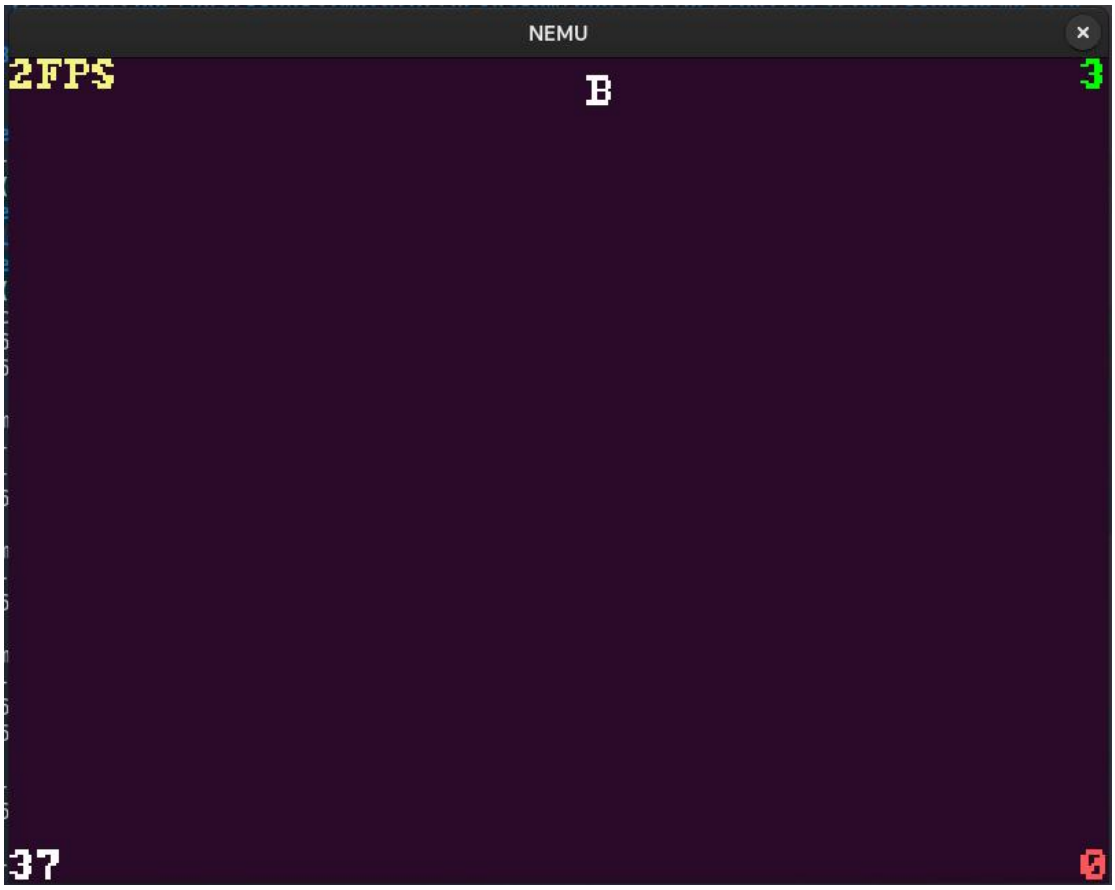
(a) Linux native, back-ended with SDL2 (screenshot)

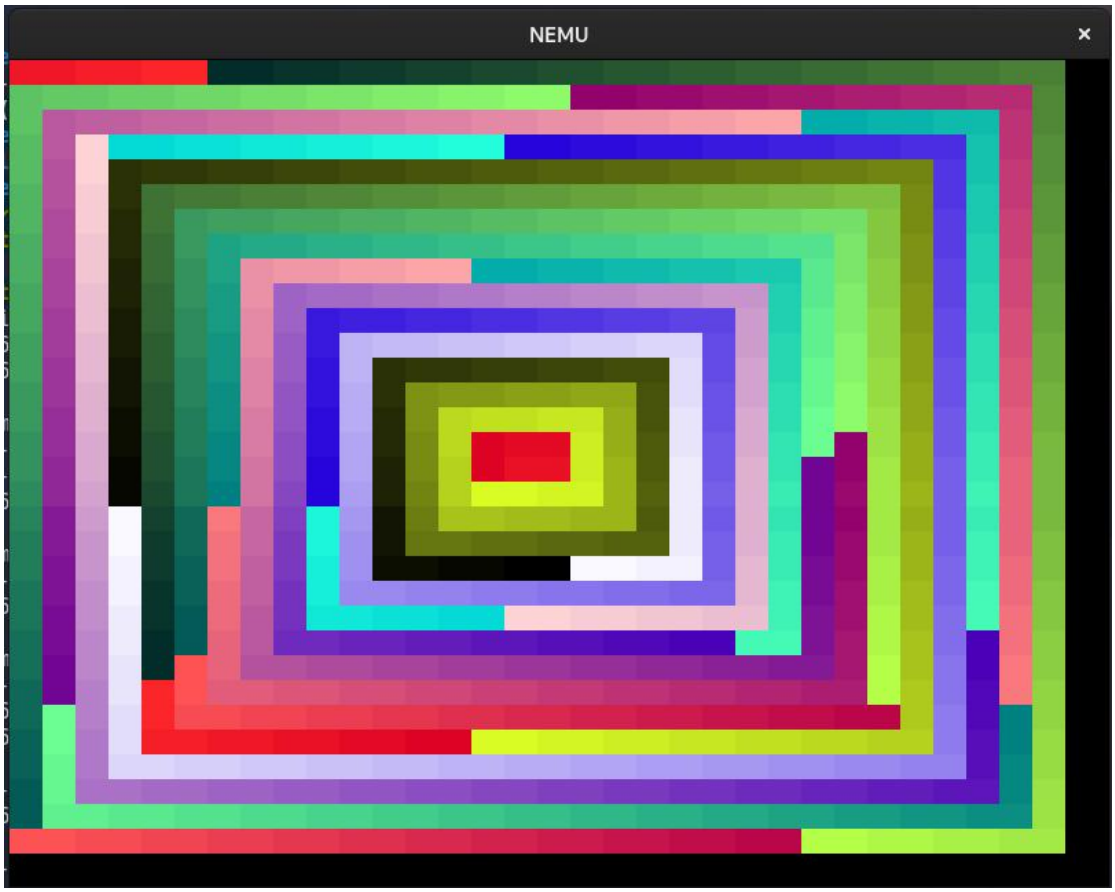
(b) a teaching x86 full system emulator (screenshot)

(c) a teaching MIPS32 SoC (FPGA, only TRM and IOE)

(d) a teaching OS (with GUI) and MIPS32 SoC (FPGA)

Figure 4. The same LiteNES emulator running on different platforms.





2.3.4 问题解答

1. 编译与链接

去掉 static，将不会发生错误。我不了解应当如何解释一个不发生的错误。

去掉 inline，编译时不会发生错误，链接时会发生错误。LD 在链接不同的 object 时，会发现同一个符号在不同 object 中被多次定义，并且我们没有告诉 LD 在这种情形下应当怎么进行链接，因此报错。inline 避免链接器报错的原因是，C++标准就是这么规定的。下面的文字中解释了 C++标准这样的规定如何得以实现。

```
It specifies the scope of the function to be that of a translation unit. So, if an inline function appears in foo.cpp (either because it was written in it, or because it #includes a header in which it was written, in which case the preprocessor basically makes it so). Now you compile foo.cpp, and possibly also some other bar.cpp which also contains an inline function with the same signature (possibly the exact same one; probably due to both #includeing the same header). When the linker
```

links the two object files, it will not be considered a violation of the ODR, as the inline directive made each copy of the file local to its translation unit (the object file created by compiling it, effectively). This is not a suggestion, it is binding.

2. 编译与链接

声明：回答此问题使用的 PA2 指 hash 为 0a8f95dd3f89bb7902fa984b01f7be49432417d6 的 commit。未来 PA2 的 HEAD 指针的移动不影响此题答案。

声明：此问题没有指定，新增的代码在头文件中的位置。我在此假定：新增的代码位于 header guarder macro 的下面一行。即，头文件头部的#ifndef 宏的后面。

声明：我的 PA2 使用 g++作为 CC 和 Linker，使用-std=c++17 编译选项。原框架代码使用的 C 语言标准与 C++17 标准出现不一致的结果请谅解。

1) 29 个。

因为按照 C++标准，实体数应等同于被链接的包含了 common.h 的 object 的个数。然后做一个简单的实验，在 common.h 添加一行 int dummy;，然后看 LD 报错多少行，就知道有多少个 object 包含了 common.h 了。

2) 29 个。

用与上一小题同样的方式做实验即可。实验结果的解释：他就是有这么多的 object 包含了 debug.h 头文件。

3) 没有出现任何问题。

3. 了解 Makefile

声明：我修改了 Makefile 的工作方式。请参阅 (https://git.recolic.org/recolic-hust/hust-x86-simulator/blob/pa2/nemu/Makefile)

开头的大量变量定义就是代码字面意思，过于冗长略去。值得注意的是，它

定义了默认 target 为 app，以及包含了 Makefile.git，用于在每次编译时自动生成一个，美观度存在争议的 commit。

随后，Makefile 会运行 SUBDIR 的 generate 这个 target，让 bison 和 flex 自动生成所需的头文件和.cc 源文件。

然后，shell 命令统计本目录下所有的.cc 源文件，自动生成对应的.o 的 target。

随后,GNU make 执行到 app 这个 default target,并 invoke $(BINARY) 这个 target。此 target 假装依赖于所有的 object, 并使得所有 object 被编译，并在$(BINARY)中进行链接，生成最终的二进制文件，结束。

3　设计总结与心得

3.1 课设总结

在本次实验中，我基于逐条指令模拟的方式，实现了一个软件 x86 模拟器和一个简单的 libc。模拟器支持大多数 i386 指令，并且有 IO、vga 设备和计时器设备可用。模拟器使用自己实现的简易 libc，而不是 glibc、uclibc-ng 等更广泛使用的实现。

它能够运行大多数 x86 程序，例如打字小游戏、nes 模拟器等，但基于逐条指令模拟的方式实现的模拟器是有性能瓶颈的。即使将此实验中的代码经过充分重构和优化，也不可能达到与二进制翻译相似的性能水平。如果要做到像 qemu 那样，拥有堪比 5 年前 CPU 的性能（我们将其称为”真正可用”），就只能用 jit 的方法，将大多数指令尽可能放到宿主机 CPU 直接执行。

3.2 课设心得

在本次课设中，我意识到了基础设施的重要性，了解了分布式版本控制系统的使用，和 GNU 工具链的使用（包括 GNU Make，Bash 等全套 Unix 传统工具链）。遵从 Unix 哲学而设计出的工具链拥有与 NT 工具链不同的美感，他们易于组合并实现复杂的自动化功能。

小即是美。

让程序只做好一件事。

尽可能早地建立原型。

可移植性比效率更重要。

数据应该保存为文本文件。

尽可能地榨取软件的全部价值。

使用 shell 脚本来提高效率和可移植性。

避免使用可定制性低下的用户界面。

所有程序都是数据的过滤器。

同时，在本次课设中，我更加熟悉了计算机的工作方式，熟悉了 i386 指令集、图形程序和 IO 操作、文件与分页、保护模式和系统调用等特性的实现方式，对巩固 CS 基础知识有很大作用。

3.3 框架代码改进建议

https://git.recolic.org/recolic-hust/hust-x86-simulator/merge_requests/4/diffs

参考文献

[1]DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第 4 版).北京：机械工业出版社.

[2]David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社

[3]秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011 年.

[4]袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011 年.

[5]张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008 年.