Programming Assignment Lecture I

Introduction

Xie

xiemhemail@gmail.com

Sep 8th, 2017

Basic Info

- Course Content
  - PA1-PA??
- Time
  - Friday each odd weak
- Assessment methods
  - Submit project and report before **deadline**.
  - An act of plagiarism is absolutely forbidden.
  - For more details, see SUBMIT REQUIREMENT WEBPAGE.
- Instruction
  - https://nju-ics.gitbooks.io/ics2017-programming-assignment/
  - Please check the "news" module of website at least once each day.

Teacher and T.A.s

Teacher	Liang Wang(wl@nju.edu.cn)
T.A.	Ruize Tang(151220100@smail.nju.edu.cn)
	Minhui Xie(xiemhemail@gmail.com)

Task load index

Here are 3 questionnaires to assess your perceived workload. We sincerely hope you finish them in each stage for our teaching research. But it isn't mandatory, **all by voluntary**!

- Q1.doc Please fill it up before the start of PA **and** fill it up when submitting **each** large stage.
- Q2.doc id.
- Q3.doc Please fill it up when submitting **each** large stage.

Submission format

Compress them with your report and PA. Submission of the first time of Q1 and Q2 (*i.e. Before the start of PA*) can be postponed to the submission of PA1. But make sure you have distinguish them in file name such like Q1_1st.doc,Q1_2nd.doc.

Resources

Plantform and tools IA-32 + GNU/Linux + gcc + C

Guidebook https://nju-ics.gitbooks.io/ics2017-programming-assignment/

Skeleton https://github.com/NJU-ProjectN/ics-pa

Tip

You can download the PDF or epub version of guide in github.

1. From Computer System, to ICS, to PA
2. How we emulate a computer? $\longrightarrow$ The story of computer
3. Help you do it! $\longrightarrow$ Brand new PA based on AM

Why we need learn ICS?

Motivation

Question

```
int main()
{
    printf("Hello World");
    return 0;
}
```

What the computer are doing when you execute the program aboveïij§

Tip

This may appear in exams.

System Stack

Application
Algorithm
Programming Language
Operating System/Virtual Machines
Instruction Set Architecture
Micro-architecture
Register-Transfer Level
Gates
Circuits
Devices
Physics

What you'll get after finishing PA

You'll

- **Get Systems thinking**
- Understand how program run on a computer
- Enhance **coding** ability
- Prepare for later courses (OS,Compiling)

What you'll get after finishing PA

You'll
- **Get Systems thinking**
- Understand how program run on a computer
- Enhance **coding** ability
- Prepare for later courses (OS,Compiling)

Way Complete a tiny but entire computer system and run program on it.

What you'll get after finishing PA

You'll
- **Get Systems thinking**
- Understand how program run on a computer
- Enhance **coding** ability
- Prepare for later courses (OS,Compiling)

Way Complete a tiny but entire computer system and run program on it.

PA NEMU(*i.e. NJU Emulator*)

Question

What is an *emulator*?

What is an *emulator*?

Emulator –Wikipedia

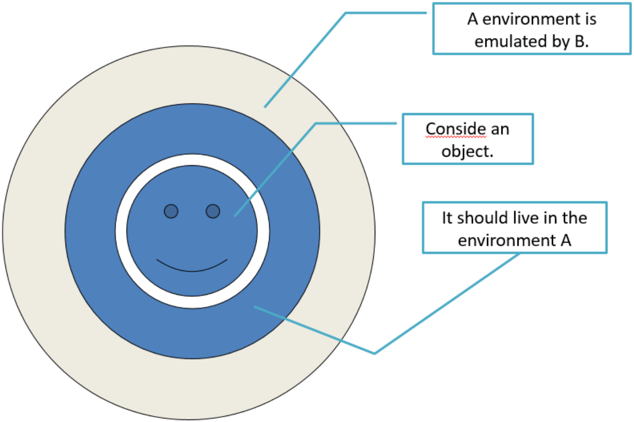
In computing, an **emulator** is hardware or software that enables one computer system (called the *host*) to behave like another computer system (called the *guest*).
An emulator typically enables the host system to run software or use peripheral devices designed for the guest system.

What is an *emulator*?

What is an *emulator*?

debian

vmware®



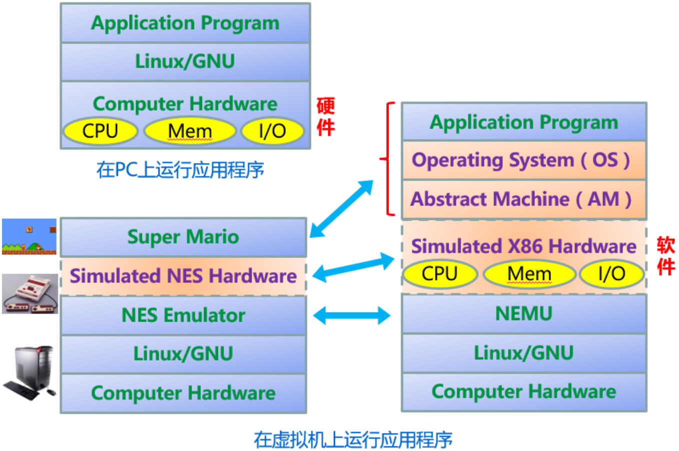
Question

Does the object know he is living in a virtualized environment?

Task of PA

Task of PA

To realize NEMU, a simplified x86 system-wide **emulator**.



Transition

So how we emulate a computer?

Programming Assignment Lecture I

1. From Computer System, to ICS, to PA
2. How we emulate a computer? $\longrightarrow$ The story of computer
3. Help you do it! $\longrightarrow$ Brand new PA based on AM

DLC

- Most of you have completed the Digital Logic Circuit Course.(Taught by zzs whj)
  - Adder
  - Register
  - Multiplexer
  - *etc.*

DLC

- Most of you have completed the Digital Logic Circuit Course.(Taught by zzs whj)
  - Adder
  - Register
  - Multiplexer
  - *etc.*
- Logic Gate ⟶ digital logic device ⟶ Computer

The simplest computer- Turing machine

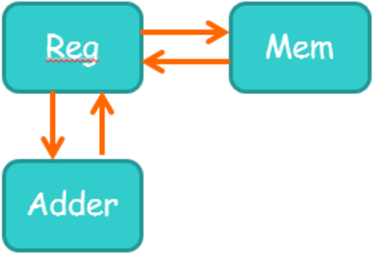
Ultimate goal of computer

Run programs on it.

To place programs Memory

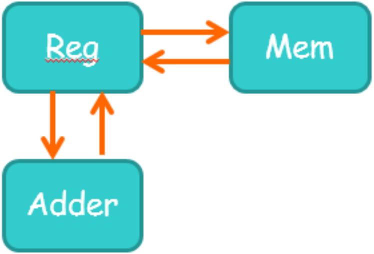
To process data Adder

To store temporary results efficiently Reg



The simplest computer- Turing machine

- TRM = Reg + Adder + Mem
  - These are all what we learned in DLC!
- Computers read data from Mem, store them to Reg, do some calculate and write the result to Mem.

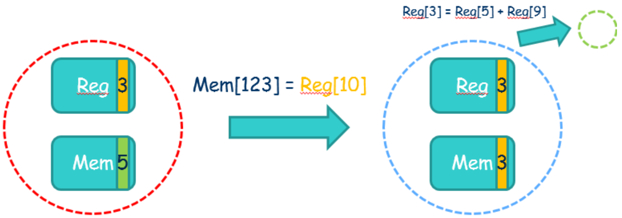


State of TRM

- Reg and Mem are all Sequential Logic Device. They can store value (i.e. state).
- The process of computer working = These sequential logical devices transfer one state to another state

Instructions of TRM

- This large state machine does have abundant states.
- But how one state transfer to another is not at liberty.
- Instructions = Commands that guide the state transition of computer
  - R/W Mem ,Computing
- Program = A lot of instructions



Instruction Set

- Instruction Set = All things computer can do
  - mov, inc,jmp

Question

What can TRM do with just these instructions?

Instruction Set

- Instruction Set = All things computer can do
  - mov, inc,jmp

Question

What can TRM do with just these instructions?

- Computability theory shows that TRM can do anything!

loop	jmp
add	inc,inc,inc,inc,inc...
multiply	add,add,add,add...
function call	mov (transfer of parameters),jmp(transfer of control)

Instruction Set

- However,the speed of TRM is so slow.
- Why do we add more powerful instructions to the computer?
- Modern computer Instruction Set(x86, mipsâĂę)
  - mul,div, cmp, bit operation, string manipulationâĂę
  - Adder⟶ALU

Input & Output

Question

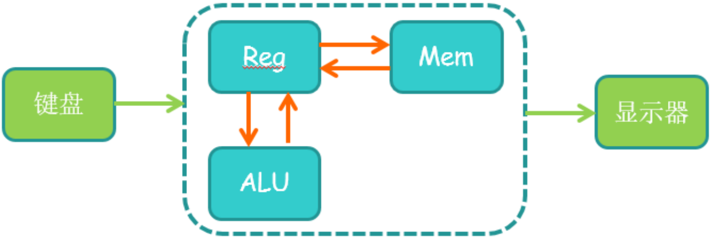
Now,it seems that we have a powerful computer.Are there any unsatisfactory drawbacks?

Input & Output

Question

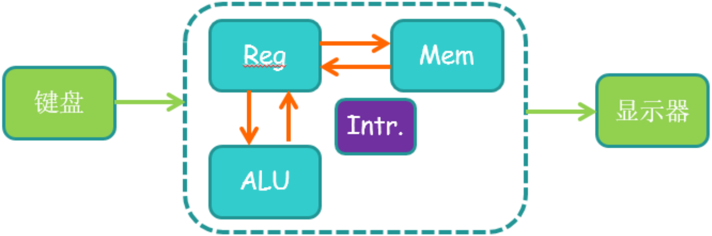
Now,it seems that we have a powerful computer.Are there any unsatisfactory drawbacks?

- It can only do computing!
- To interact with the outside world, we need to add I/O devices
- plus with some relative instructions.



Interrupt & Exception

- Besides running regular program, computer should deal with emergencies whenever possible.
- Such as,
  - Inner exception: division by 0, access violation, trap etc.
  - Outer interruption:Keyboard, device ready,etc.
- We need add asynchronous processing unit to deal with emergencies.



Multitask

Question again

Now,it seems that we have a powerful computer.Are there any unsatisfactory drawbacks?

Multitask

Question again

Now,it seems that we have a powerful computer.Are there any unsatisfactory drawbacks?

- Monotask ⟶ Mutitask
- Time-division multiplexing: Improve the efficiency of resources usages.
  - CPU, I/O Device: Take turns!

Question

Can tasks use memory exclusively by turns?What else should we do in practice?

Multitask

Question again

Now,it seems that we have a powerful computer.Are there any unsatisfactory drawbacks?

- Monotask ⟶ Mutitask
- Time-division multiplexing: Improve the efficiency of resources usages.
  - CPU, I/O Device: Take turns!

Question

Can tasks use memory exclusively by turns?What else should we do in practice?

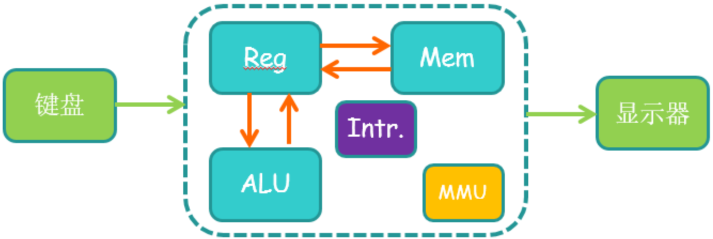
Answer

Certainly not.Exclusive memory using will cause inefficiency.
We should add MMU for memory protection.

AM(Abstract Machine)

$$\text{AM} = \text{TRM} + \text{IOE} + \text{ASYE} + \text{PTE} + \text{MPE}$$

- TRM(Turing Machine)
- IOE(I/O Extension)
- ASYE(Asynchronous Extension)
- PTE(Protection Extension)
- MPE(Multi-Processor Extension) (not included in PA)



How to realize NEMU?

By now, we have known what a modern computer should look like, then how we emulate it? Or what is the constituent of NEMU?

How to realize NEMU?

By now, we have known what a modern computer should look like, then how we emulate it? Or what is the constituent of NEMU?

What tools we already have?

How to realize NEMU?

By now, we have known what a modern computer should look like, then how we emulate it? Or what is the constituent of NEMU?

What tools we already have?

- Variables,data structure
- Algorithm

Devices C variables, arrays, link list, etc.

Logic functions Algorithmic functions

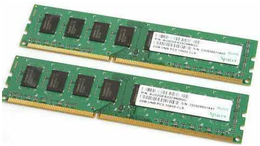
Runtime support Loop

E.g. cpu

```
struct CPU_State
{
    rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
    vaddr_t eip;
    EFLAGS;
    CRs;
    ......
}
```

E.g. memory

```
uint8_t pmem[PMEM_SIZE];
```



The simplest computer- Turing machine

Architecture of the simplest computer

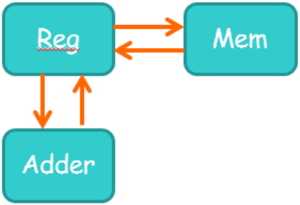
To place programs Memory

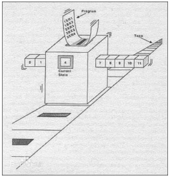
To process data Adder

To store temporary results efficiently Reg

Working mode of the simplest computer

- Fetch instruction from Mem using PC.
- Execute instruction.
- Update PC.





TRM in NEMU

Architecture of the simplest computer

Memory	**pmem array**(128MB nemu/src/memory/memory.c)
Adder(ALU)	**functions**
Reg	**CPU_state struct** (nemu/include/cpu/reg.h)

Working mode of the simplest computer

Function cpu_exec() (nemu/src/monitor/cpu-exec.c)

- Fetch instruction from Mem using PC.
- Execute instruction.
- Update PC.

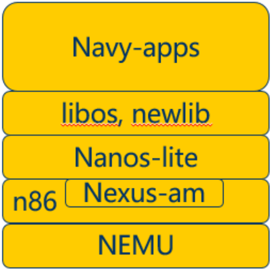
Programming Assignment Lecture I

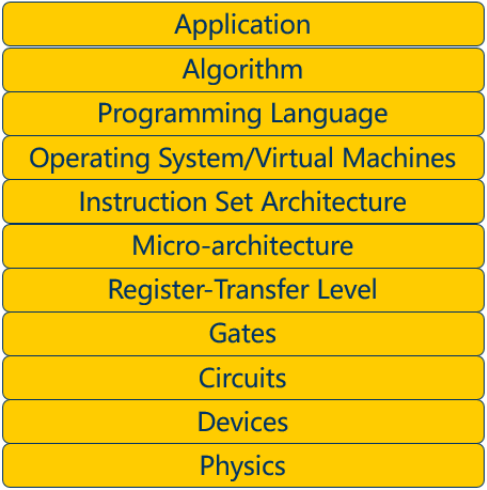
1. From Computer System, to ICS, to PA
2. How we emulate a computer? $\longrightarrow$ The story of computer
3. Help you do it! $\longrightarrow$ Brand new PA based on AM

Structure of PA

```
ics2017
|---nanos-lite    # mini operating system kernel
|---navy-apps     # apps
|---nemu          # NEMU
|---nexus-am      # abstract machine
```

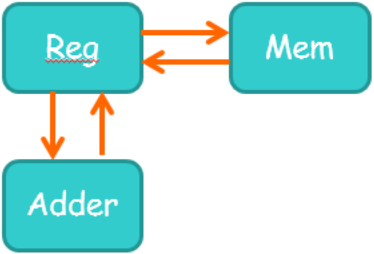
Structure of PA





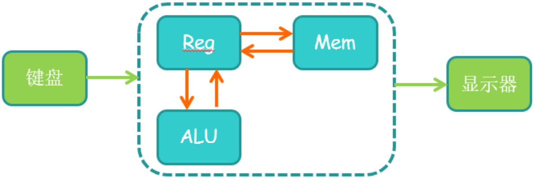
PA0 and PA1

- PA0 - Environment configuration
- PA1 - Simple monitor



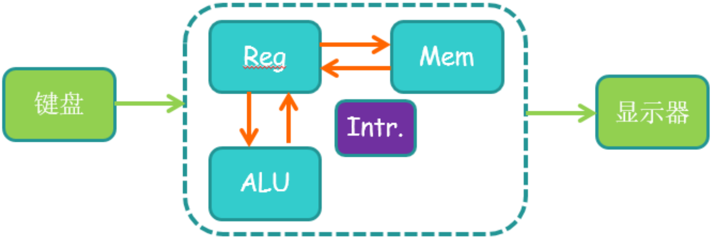
PA2-Von Neumann Computer System (TRM+IOE)

- Instructions system + I/O
- Run interesting apps on NEMU



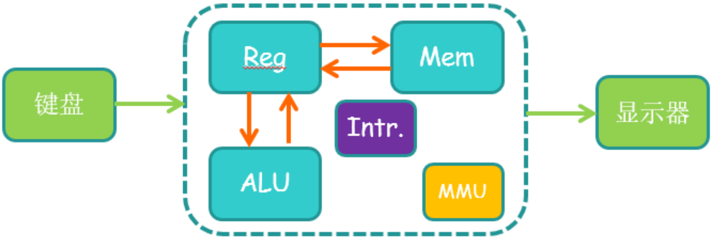
PA3- Asynchronous Extension

- Run tiny operating system Nanos-lite on NEMU.
  - ramdisk, fs, raw img loader
- Implement some library functions which packing the system call .
- App calls library functions and run PAL.



PA4- Multitask(Protection Extension)

- Virtual storage.
- Interruption.
- Run pal,typing game,clock at the same time on Nanos-lite.



PA5- Performance

- Elf32 loader
- cache
- TLB
- profiler, performance optimization
- JIT *(i.e just-in-time)* compilation

实验内容(括号中为新方案)	持续时间/周	预计耗时/小时	代码量/行
PA0 - 开发环境配置(不变)	1	10	无
PA1 - 简易调试器(不变)	3	30	400
PA2 - 指令系统 (PA2 - 冯诺依曼计算机系统)	6 (4)	60 (30)	800 (300)
课时不足可选择完成到PA2 [小计]	(8)	(70)	(700)
PA3 - 存储管理 (PA3 - 异常控制流)	4 (3)	50 (30)	500 (200)
PA4 - 中断与I/O (PA4 - 分时多任务)	3 (3)	30 (30)	300 (200)
无 (PA5 - 程序性能)[可设置为选做]	无 -	无 -	无 -
总计	17 (14)	180 (130)	2000 (1100)

The End