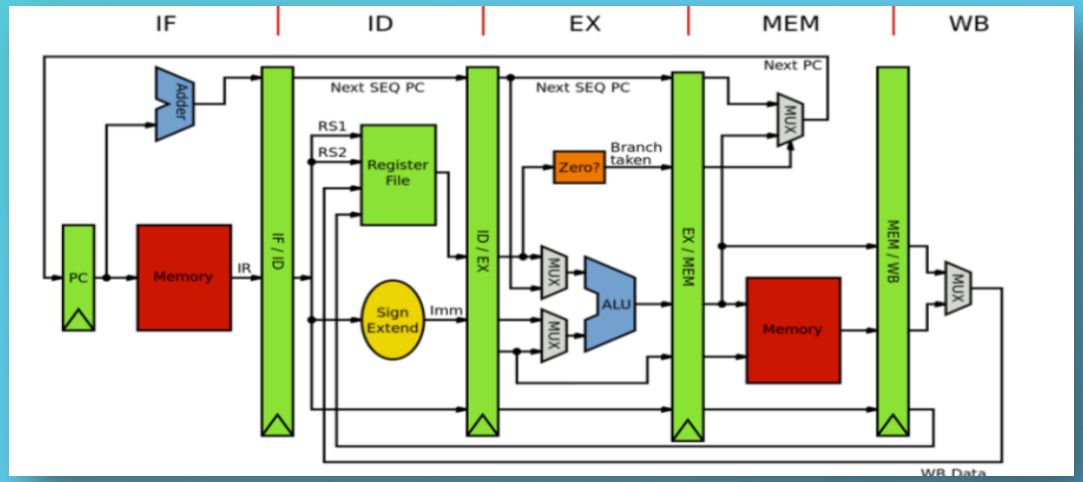
RISC-V CPU 模拟器

编程综合实践

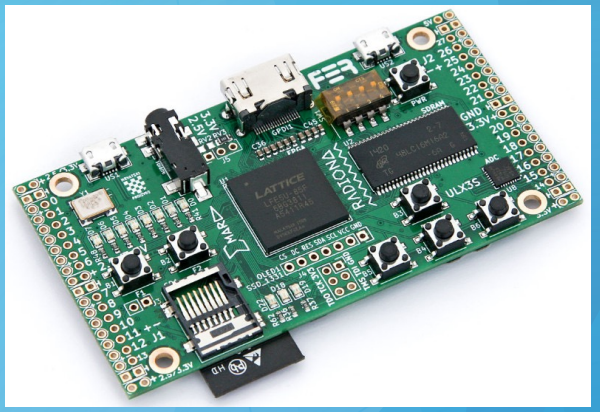
Principle and Practice of Computer Algorithms

2021级ACM班 & 致远工科荣誉 2022.6.20



RISC-V®

RISC-V: The Free and Open RISC Instruction Set Architecture



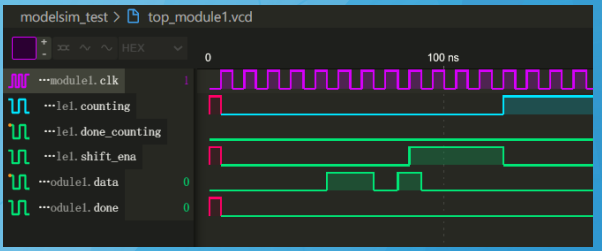
```
00000000 <.rom>:
   0:   00020137            lui sp,0x20
   4:   040010ef            jal ra,1044 <main>
   8:   0ff00513            li  a0,255
   c:   00030eb7            lui a3,0x30
  10:   00a68223            sb  a0,4(a3) # 30004 <__heap_start+0x2e004>
  14:   ff9ff06f            j   c <printInt-0xff4>

Disassembly of section .text:

00001000 <printInt>:
    1000:   00001737            lui a4,0x1
    1004:   06c72783            lw  a5,108(a4) # 106c <__bss_end>
    1008:   00f54533            xor a0,a0,a5
    100c:   0ad50513            addi    a0,a0,173
    1010:   06a72623            sw  a0,108(a4)
    1014:   00008067            ret

00001018 <printStr>:
    1018:   00054783            lbu a5,0(a0)
    101c:   02078263            beqz    a5,1040 <printStr+0x28>
    1020:   00001737            lui a4,0x1
    1024:   06c72683            lw  a3,108(a4) # 106c <__bss_end>
    1028:   00d7c7b3            xor a5,a5,a3
    102c:   2097879b            addiw   a5,a5,521
    1030:   06f72623            sw  a5,108(a4)
    1034:   00150513            addi    a0,a0,1
    1038:   00054783            lbu a5,0(a0)
    103c:   fe0794e3            bnez    a5,1024 <printStr+0xc>
    1040:   00008067            ret

00001044 <main>:
    1044:   ff010113            addi    sp,sp,-16 # 1fff0 <__heap_start+0x1dff0>
    1048:   00112623            sw  ra,12(sp)
    104c:   0b100513            li  a0,177
    1050:   fb1ff0ef            jal ra,1000 <printInt>
    1054:   000017b7            lui a5,0x1
    1058:   06c7a503            lw  a0,108(a5) # 106c <__bss_end>
    105c:   00c12083            lw  ra,12(sp)
    1060:   01010113            addi    sp,sp,16
    1064:   00008067            ret
```

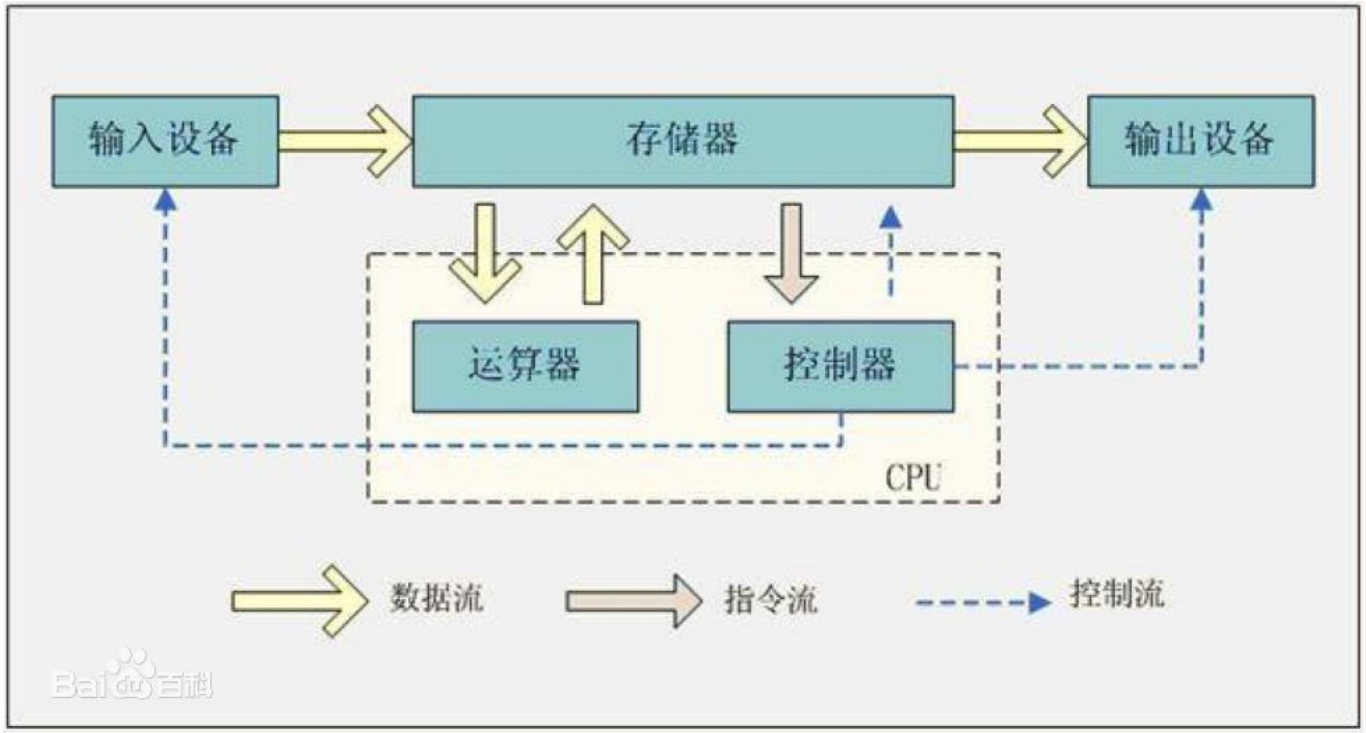


项目目标

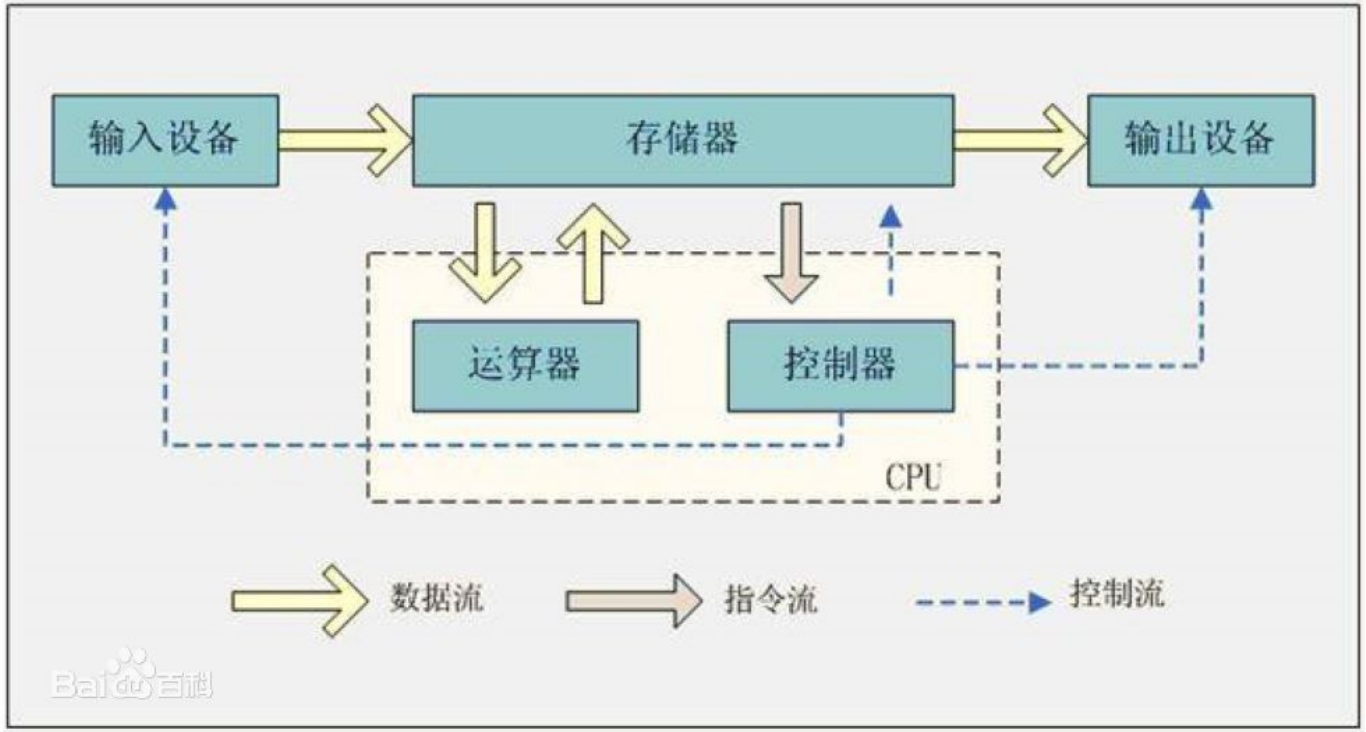
- 了解计算机架构、指令集架构
- 学习调度算法
- 了解硬件设计

01 Architecture

01 冯诺依曼架构

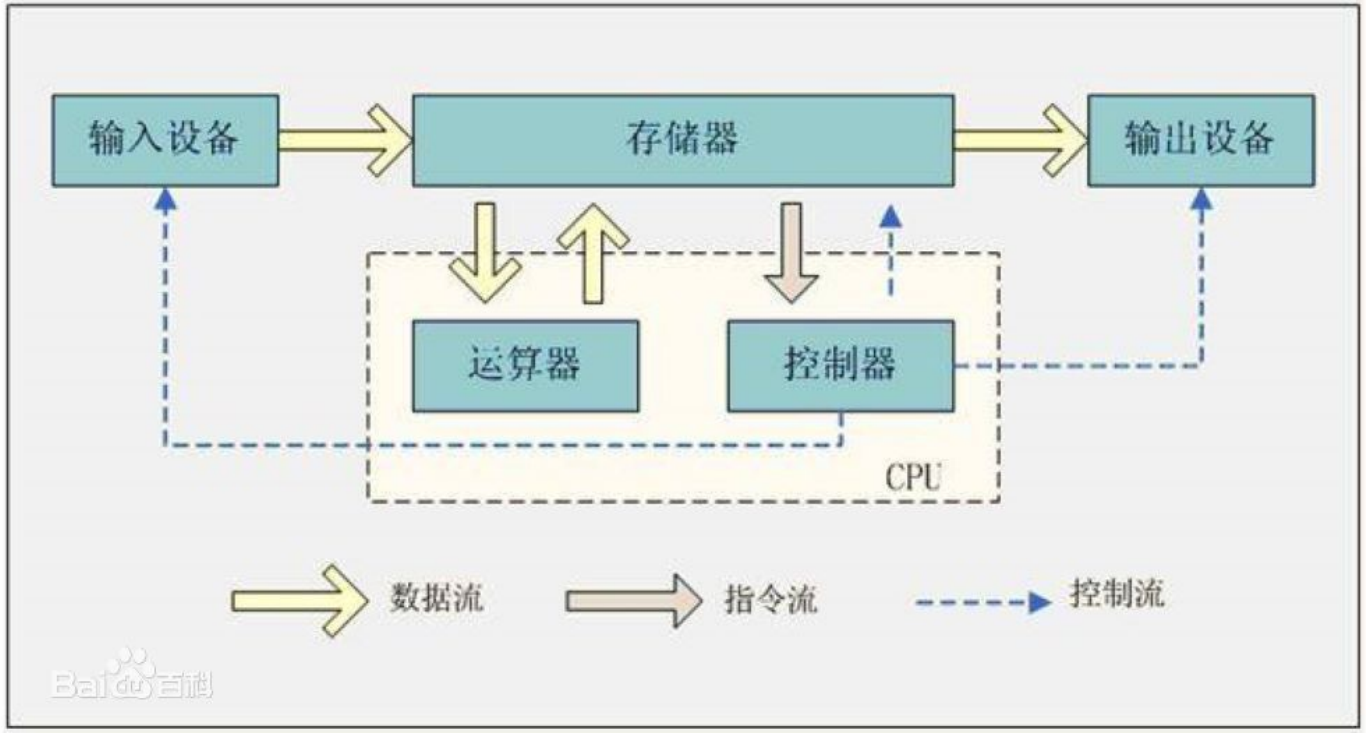


01 冯诺依曼架构


输入设备
存储器
输出设备
运算器
控制器
CPU
数据流
指令流
控制流


AMD Ryzen 9 5950X
AMD
RYZEN

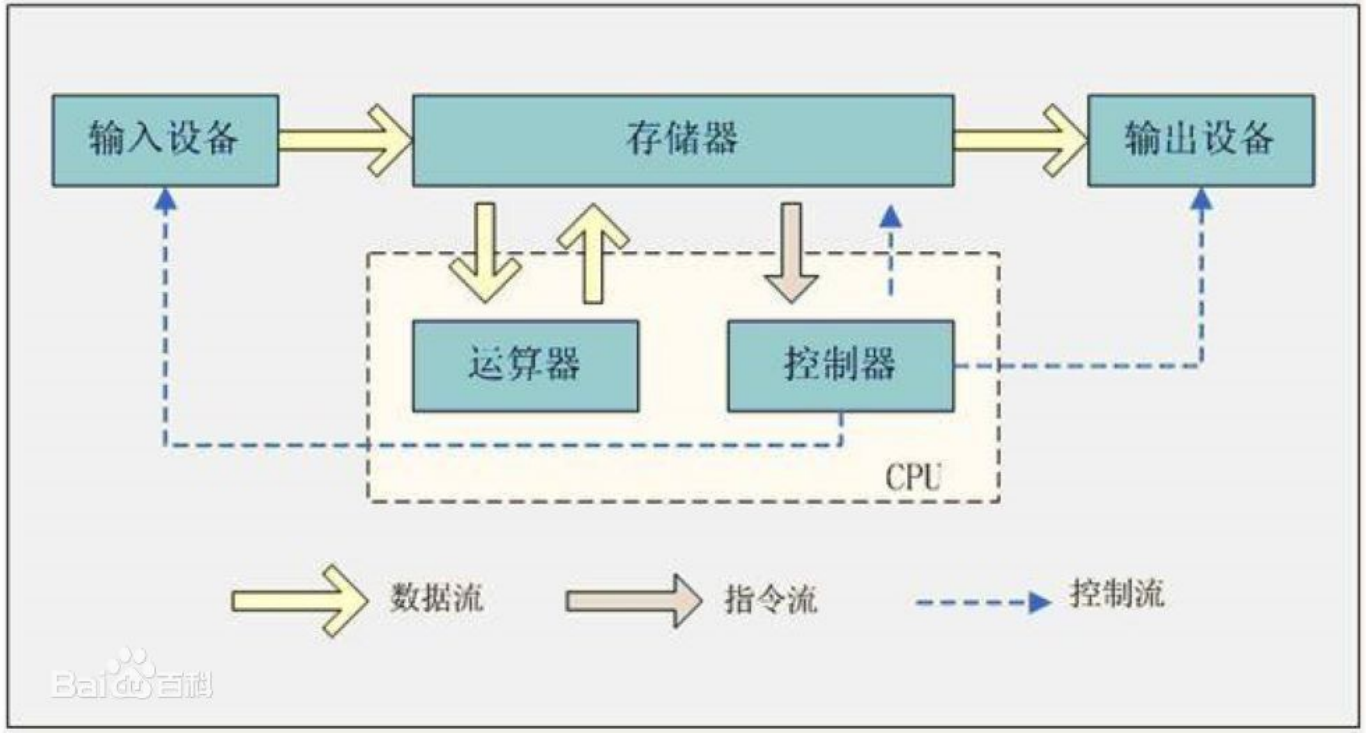
01 冯诺依曼架构



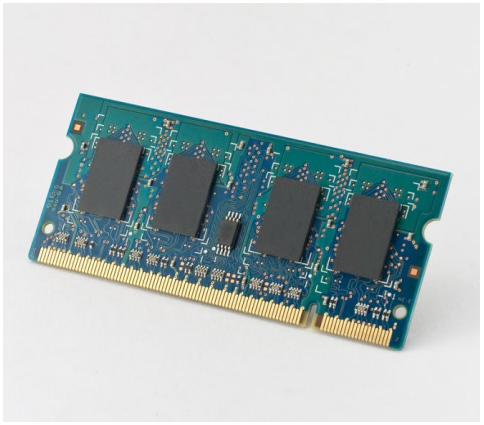
控制与运算单元

寄存器

01 冯诺依曼架构

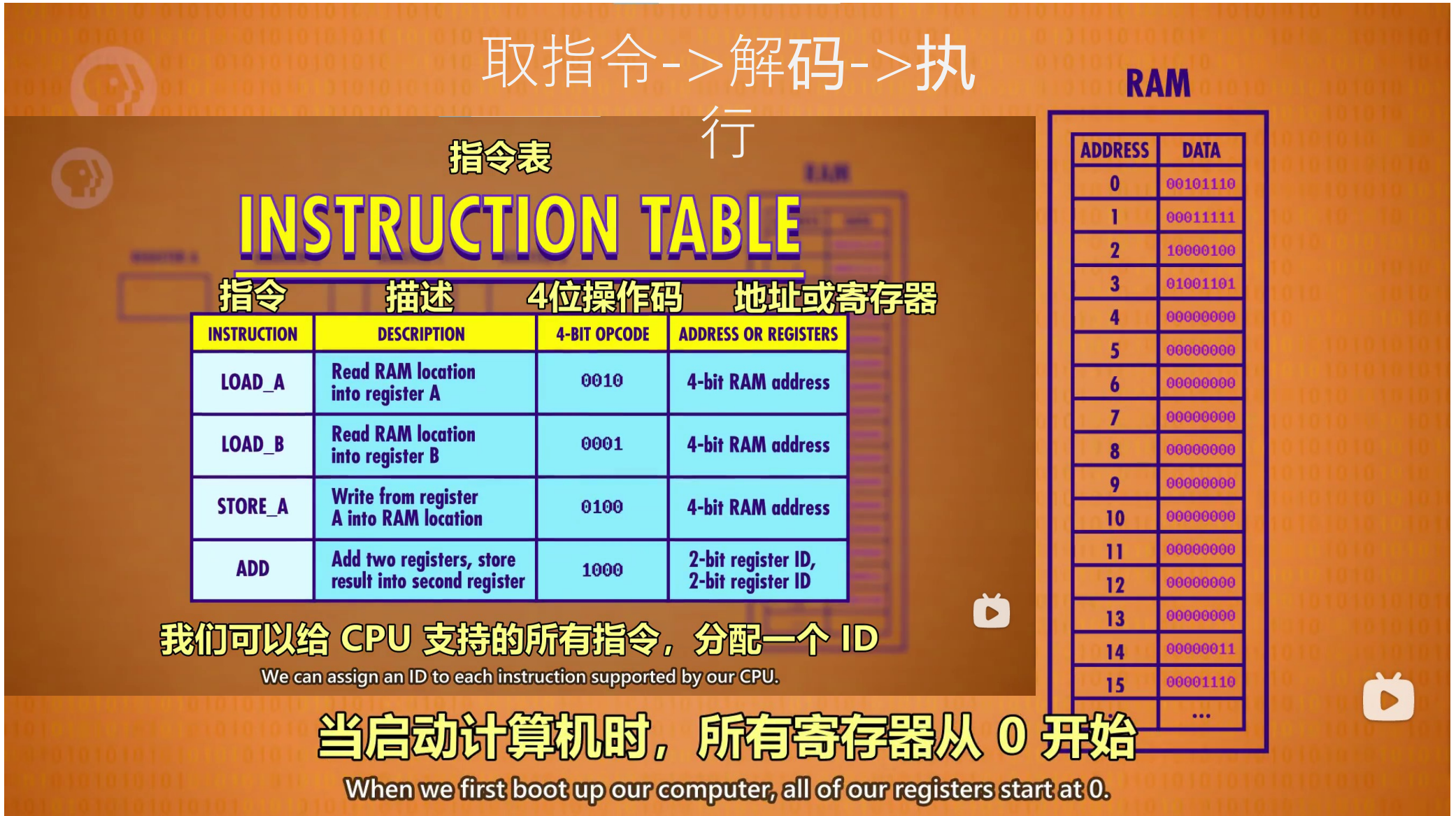


- 控制与运算**单**元
- 内存
- 寄存器



01 概念：CPU执行指令的流程

https://www.bilibili.com/video/BV1EW411u7th 5-9



02 ISA

02 Instruction Set Architecture

- In computer science, an instruction set architecture (ISA), also called computer architecture, is an abstract model of a computer. A device that executes instructions described by that ISA, such as a central processing unit (CPU), is called an implementation.
- Including：
  - Instructions
  - Register
  - Data type, address pattern, interruption, I/O…
- Classification
  - Complex instruction set computer (CISC)
  - Reduced instruction set computer (RISC)

Assembly vs. machine code

Machine code bytes	Assembly language statements
	foo:
B8 22 11 00 FF	movl $0xFF001122, %eax
01 CA	addl %ecx, %edx
31 F6	xorl %esi, %esi
53	pushl %ebx
8B 5C 24 04	movl 4(%esp), %ebx
8D 34 48	leal (%eax,%ecx,2), %esi
39 C3	cmpl %eax, %ebx
72 EB	jnae foo
C3	retl

Instruction stream

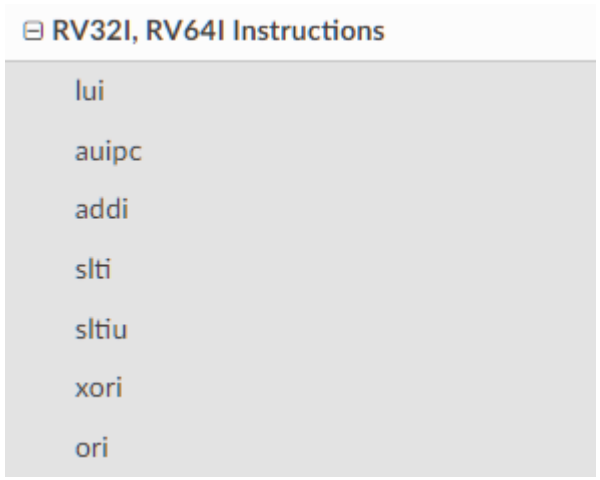
```
B8 22 11 00 FF 01 CA 31 F6 53 8B 5C 24
04 8D 34 48 39 C3 72 EB C3
```

RISC vs CISC

	RISC	CISC
Instruction width	Fixed(despite some 16bits compacted extensions)	Variable
Memory access style	Load, store with memory compute in registers	One single instruction can access both registers and memory
Code size and cycles	Low cycles per second, large code sizes	Small code sizes, high cycles per second
Memory access efficiency	Heavy use of RAM	More efficient of RAM
Instruction Number	Small	Large number

02 RISC-V

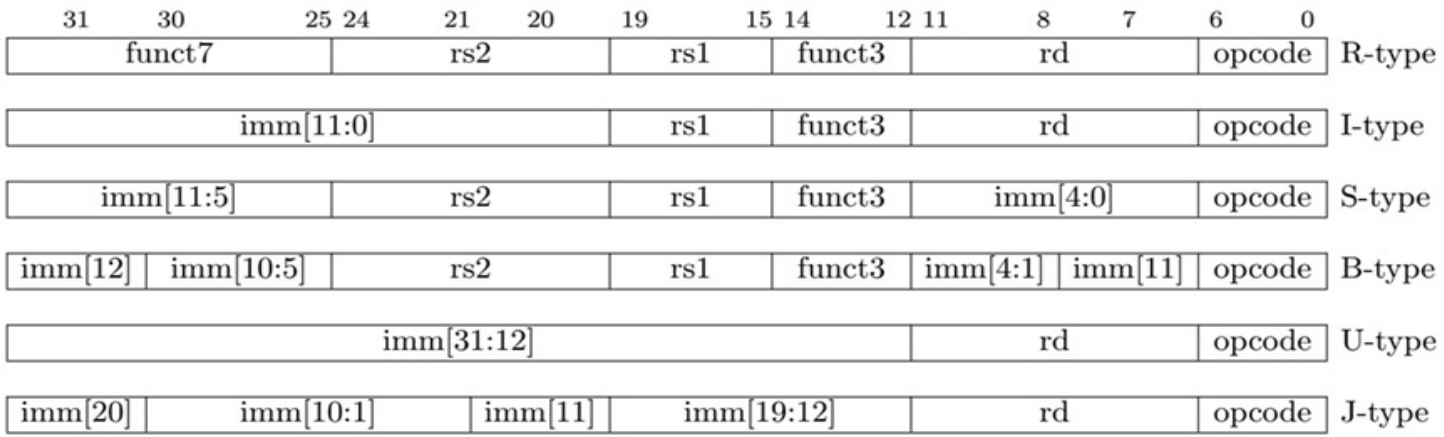
- RISC-V（发音为“risk-five”）是一个基于精简指令集（RISC）原则的开源指令集架构（ISA），简易解释为开源软体运动相对应的一种“开源硬体”。该项目2010年始于加州大学柏克莱分校，但许多贡献者是该大学以外的志愿者和行业工作者。
- 作业里涉及的指令是RV32I的一部分。32位基础整数指令集，它支持32位寻址空间，支持字节地址访问，寄存器也是32位整数寄存器。



x0	zero	Hard-wired zero，常数0	
x1	ra	Return address	caller，调用函数的指令pc
x2	sp	Stack pointer	callee，被调用的函数指令pc
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5	t0	Temporary/alternate link register	caller
x6	t1	Temporaries	caller
x7	t2	Temporaries	caller
x8	s0/fp	Saved register/frame pointer	caller
x9	s1	Saved register	caller
x10	a0	Function arguments/return values	caller
x11	a1	Function arguments/return values	caller
x12	a2	Function arguments	caller
x13	a3	Function arguments	caller
x14	a4	Function arguments	caller
x15	a5	Function arguments	caller

02 RISC-V Instruction

- Opcode
- Funct
- Rd
- Operand(2)
  - Rs1, Rs2
  - Immediate



02 RISC-V Instruction

```
@00000000
37 01 02 00 EF 10 00 04 13 05 F0 0F B7 06 03 00
23 82 A6 00 6F F0 9F FF
@00001000
37 17 00 00 83 27 C7 06 33 45 F5 00 13 05 D5 0A
23 26 A7 06 67 80 00 00 83 47 05 00 63 82 07 02
37 17 00 00 83 26 C7 06 B3 C7 D7 00 93 87 97 20
23 26 F7 06 13 05 15 00 83 47 05 00 E3 94 07 FE
67 80 00 00 13 01 01 FF 23 26 11 00 13 05 10 0B
EF F0 1F FB B7 17 00 00 03 A5 C7 06 83 20 C1 00
13 01 01 01 67 80 00 00
@00001068
FD 00 00 00
```

其后数据在内存中的起始位置

00001737
opcode = 0110111 = lui
imm = 0x1
rd = 01110 = 14 = a4
lui a4,0x1
a4 = 0x1 << 12

pc寄存器：读取的指令的位置
顺序+4，遇到jump等跳转指令则需要改变pc寄存器的值

Scheduling

运行的拆分

- 一条指令的运行会被拆分成若干的步骤，例如
  - Instruction Fetch
  - Instruction Decode
  - Execution
  - Memory Access
  - Write Back to rd register
- 空间上的互异，不同指令的不同阶段能够并行。

指令的并行调度

- 空间上的互异，不同指令的不同阶段能够并行。
- 可能遇到的问题
  - Hazards
    - Data hazard: dependence
    - Structural hazard
    - Control hazard: branch
- 并行调度算法
  - Pipeline（明天由何夏麟助教讲解）
  - 乱序执行
    - Scoreboard
    - Tomasulo algorithm（周三由洪熠佳助教讲解）

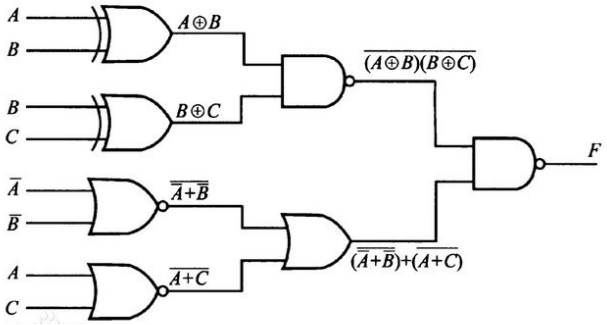
分支预测

- 静态预测：一律跳转/不跳转
- 动态预测：
  - 饱和计数器
  - 两级自适应预测器
  - ~~ML~~

04 Hardware Design

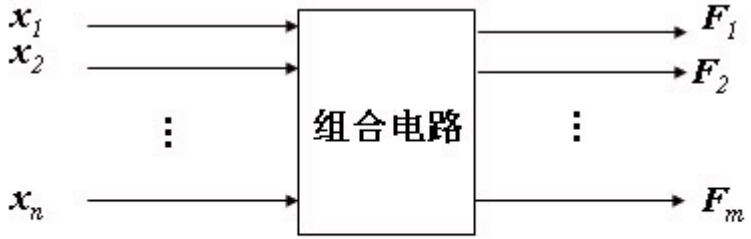
04 逻辑电路

- 逻辑电路是一种离散信号的传递和处理，以二进制为原理、实现数字信号逻辑运算和操作的电路。
- 逻辑电路的设计：
  - 物理搭线
  - 软件设计：
    - hardware design language(HDL), e.g. Verilog
    - 仿真模拟
    - 综合成电路
    - 烧录到设备上（芯片，*FPGA，域可编程逻辑阵列*）



04 组合逻辑与时序逻辑

- 组合逻辑
  - 它的任一时刻的稳态输出，仅仅与该时刻的输入变量的取值有关，而与该时刻以前的输入变量取值无关。进行逻辑代数运算。
  - 在电路设计中对应无记忆元件。
  - 组合逻辑输入输出间存在延迟。



$$F_i = f(x_1, x_2, \cdots, x_n) \qquad (i = 1, 2, \cdots, m)$$

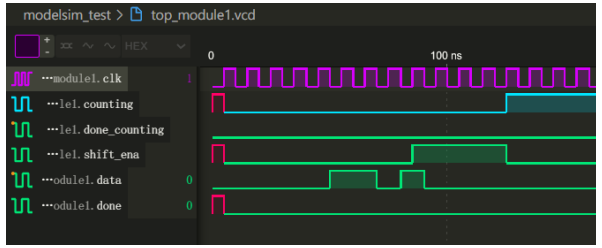
- 时序逻辑
  - 电路任何时刻的稳态输出不仅取决于当前的输入，还与前一时刻输入形成的状态有关。
  - 在电路设计中对应存在记忆原件。
  - 在时钟上升沿更新记忆原件的值。

04 组合逻辑与时序逻辑

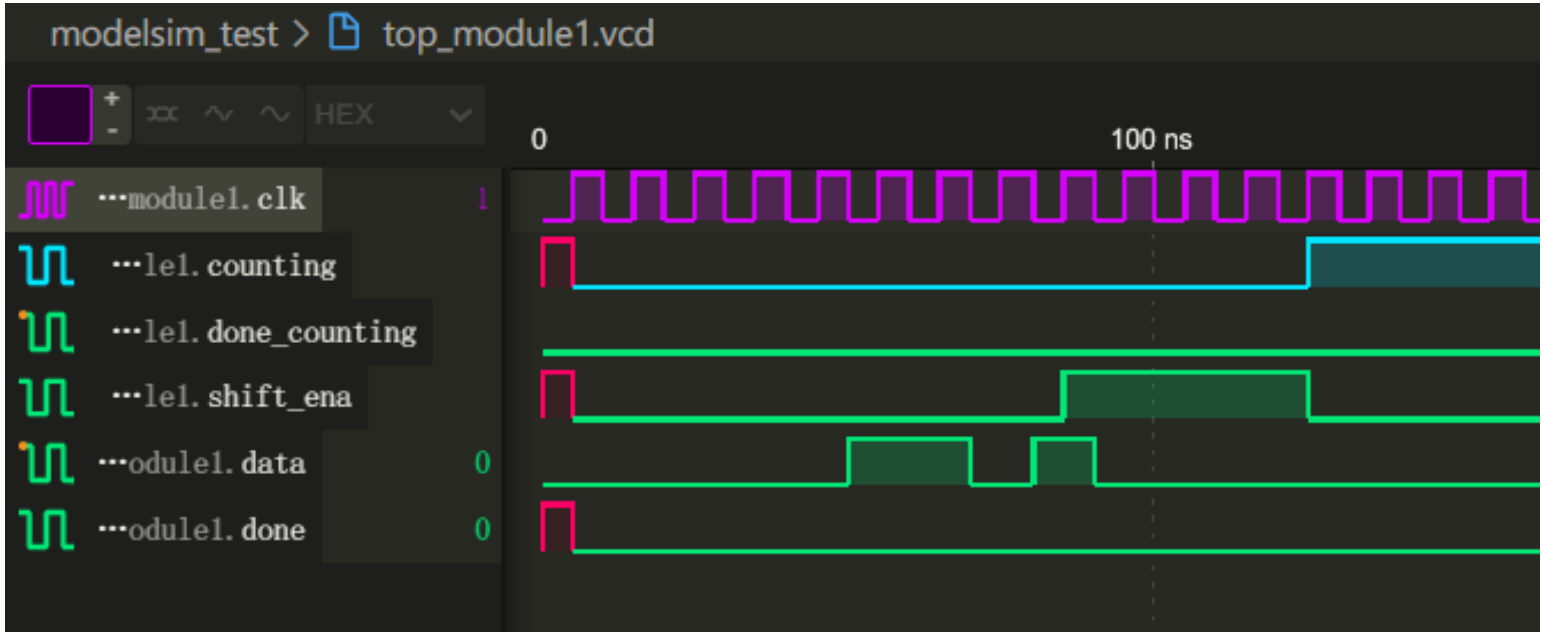
- 对组合逻辑与时序逻辑的通俗理解

记忆单元（寄存器、输入等）-> 组合逻辑电路 -> 输出 -> 更新记忆单元（时钟上升沿）

两个上升沿之间的间隔：保证组合逻辑能顺利得到输出



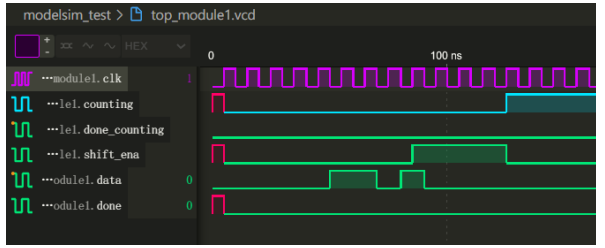
04 时钟clk


modelsim_test
top_module1.vcd
HEX
0
100 ns
···module1.clk
1
···le1.counting
···le1.done_counting
···le1.shift_ena
···odule1.data
0
···odule1.done
0

04 对Simulator的要求

- 模拟clk循环，clk数来反应调度的效率
- 模拟寄存器：两个态，in和out，通过上一周期的out计算出in
- 模拟上升沿对寄存器的update（在周期开始用in更新out）
- 支持各stage的乱序（模拟硬件上的并行）

```
clk=0;
while (true) {
        clk++;
        update();
        random_run{IF,ID,EX,MEM,WB};
        ……
}
```



05 Simulator

Your Task

Simulation

- 给出的数据中含有.c, .dump和.data文件
- Input (stdin)：.data 文件
- Output (stdout)：模拟器运行结果
- .c文件和.dump文件可供参考

05 概念：CPU执行指令的流程

- 我要做什么？取指令->解码->执行
  - 开一个小数组（如unsigned reg[32]）作为寄存器；
  - 开一个大数组（如unsigned mem[500000]）作为内存；
  - 程序开始运行时，指令（是一串01串）在内存里，根据一个当前正在运行第几条指令的计数器去找内存里的第几条指令，把指令拿出来放到寄存器（Fetch Phase）
  - 根据一定的指令规范，从指令中截取出：（Decode Phase）
    - 操作码（opcode），它决定了这条指令要做什么
    - 操作对象，可能是寄存器，也可能是一个内存单元。如果某指令要对某数据做访问、修改等操作，就会用到操作对象。
  - 识别出操作码与操作对象，该运算运算，该访问访问，该修改修改（Execute Phase）

05 RISCV里的真实寄存器

- 我要做什么？取指令->解码->执行
  - 开一个小数组（如unsigned reg[32]）作为**寄存器**
  - 开一个大数组（如unsigned mem[500000]）作为
  - 程序开始运行时，指令（是一串01串）在内存里，去找内存里的第几条指令，把指令拿出来放到寄存
  - 根据一定的指令规范，从指令中截取出：（Decoc
    - 操作码（opcode），它决定了这条指令要做什
    - 操作对象，可能是寄存器，也可能是一个内存等操作，就会用到操作对象。
  - 识别出操作码与操作对象，该运算运算，该访问访

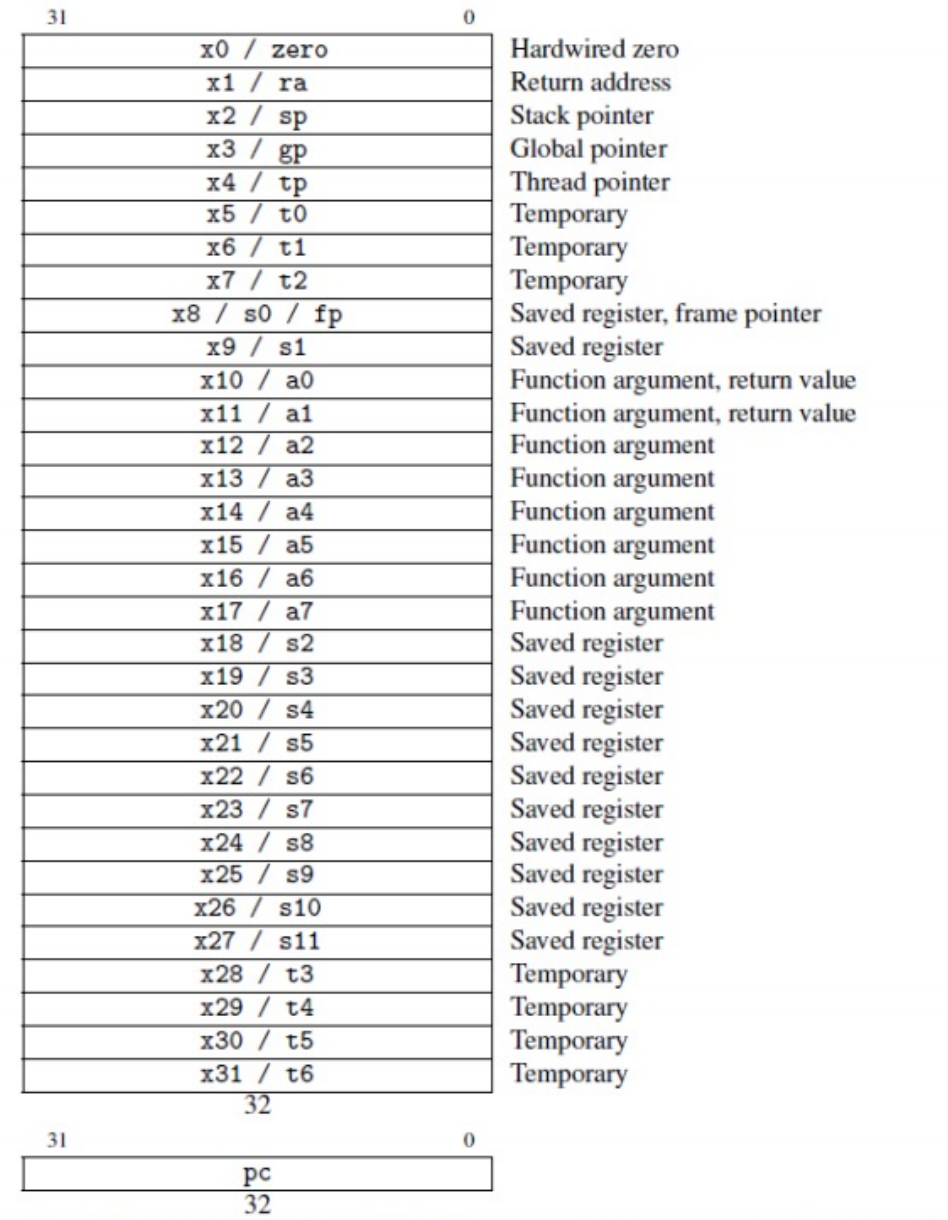


图 2.4：RV321 的寄存器。第 3 章解释了 RISC-V 调用约定，各种指针（sp，gp，tp，fp），保存寄存器（s0-s11）和临时寄存器（t0-t6）背后的基本原理（基于[Waterman and Asanovi'c 2017]的图 2.1 和表

05 RISCV里的真实指令

- 我要做什么？取指令->解码->执行
  - 开一个小数组（如unsigned reg[32]）作为寄
  - 开一个大数组（如unsigned mem[500000]）
  - 程序开始运行时，指令（是一串01串）在内存
    去找内存里的第几条指令，把指令拿出来放到
- 根据一定的**指令规范**，从指令中截取出：（D
  - 操作码（opcode），它决定了这条指令要
  - 操作对象，可能是寄存器，也可能是一个
    等操作，就会用到操作对象。
- 识别出操作码与操作对象，该运算运算，该访

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

05 RISCV里的真实指令

摘自The RISC-V Instruction Set Manual

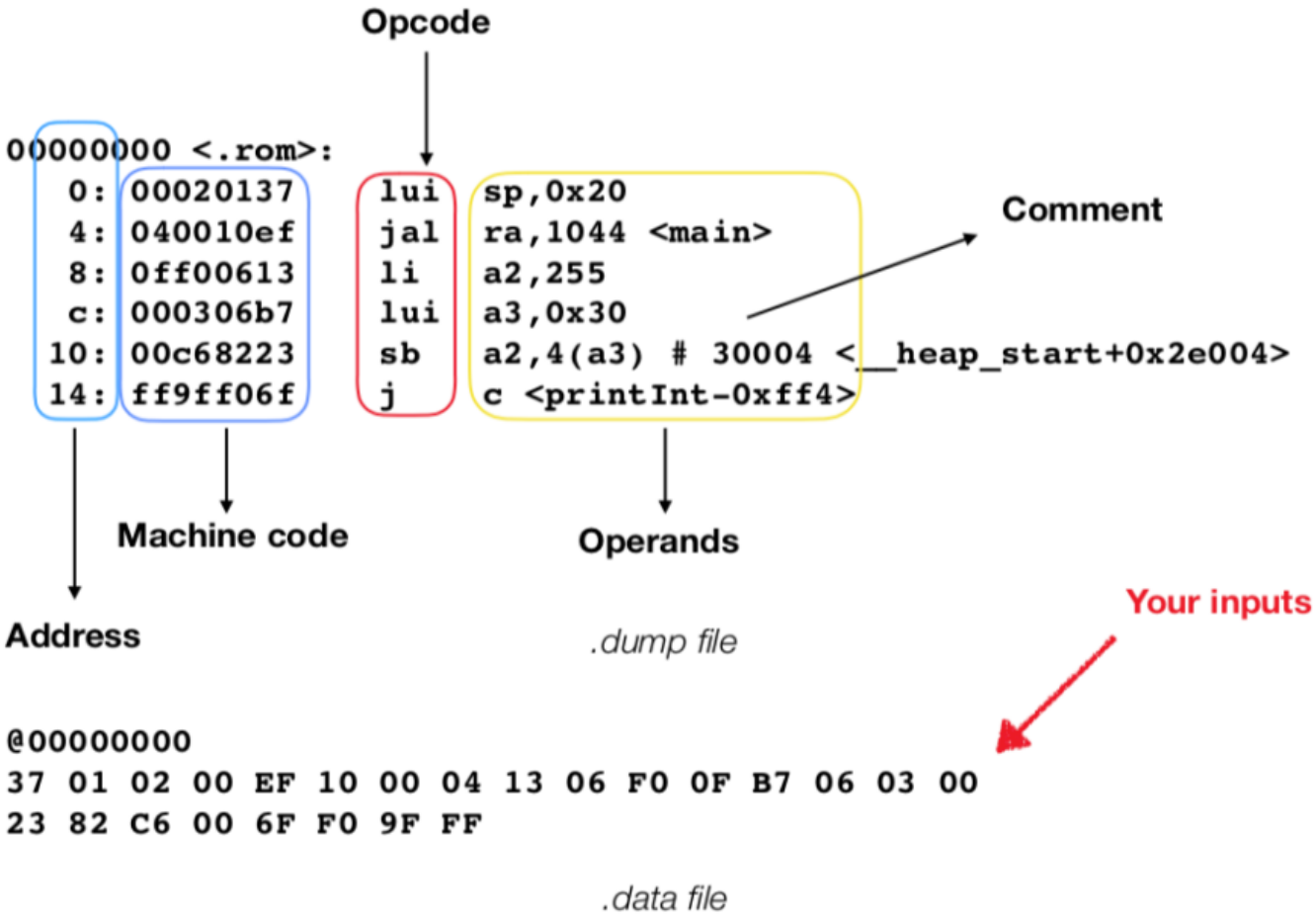
- 指令呈现方式为二进制，即使用一个32 位的01串来描述
- 每条指令的详细含义参考Chapter2
- 右图为本项目需要实现的全部指令

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

RISCV里的真实指令

- RISC-V Assembly
- 汇编语言 (assembly language) 的一种
- 是基于计算机基础结构的基础语言
- 每条指令语句都会是简单的，仅涉及三个元素的操作
- Reference: Chapter 2, The RISC-V Instruction Set Manual, VOL1（参考2.1-2.7即可）
- 使用RV32I基础整数指令集 (RV32I Base Integer ISA)

05 如何读dump


Opcode
00000000 <.rom>:
0: 00020137 lui sp,0x20
4: 040010ef jal ra,1044 <main>
8: 0ff00613 li a2,255
c: 000306b7 lui a3,0x30
10: 00c68223 sb a2,4(a3) # 30004 <__heap_start+0x2e004>
14: ff9ff06f j c <printInt-0xff4>
Comment
Machine code
Operands
Address
.dump file
Your inputs
@00000000
37 01 02 00 EF 10 00 04 13 06 F0 0F B7 06 03 00
23 82 C6 00 6F F0 9F FF
.data file

RISCV里的真实指令

- 执行流程：
  - 从标准输入读入机器指令
  - 从内存00000000处开始取指令执行，每次连续取4个2位十六进制数，组成一条指令（如取到“37 17 00 00”，拼成32位指令“00001737”）
  - load/store指令内存访问部分（第四级）请用**三个周期**模拟
  - 执行到指令0ff00513（ li a0,255 ）时，向标准输出 输出程序的返回值（一个0-255的非负整数），结束模拟。注意：
    - 程序的返回值存在a0寄存器里，但是寄存器是32位的，返回值是8位的，所以你应该输出a0的后八位。例如，你的a0寄存器是int数组reg中的reg[10]，你**应该输出**的是((unsigned int)reg[10]) & 255u。

Simulator的约定

- 数据内存读写和代码内存读写不冲突
- 代码内存读写一个周期可读出一条指令，且不需要等待3个周期
- Pc指针计算和预测可以是组合逻辑

一些建议

建议流程：

熟悉RISCV指令集，弄懂机器指令的执行流程

写一个不带流水的简易模拟器

加上五级流水

写写bonus，欢迎实现各种个性化的功能

项目评分

- 五级流水
  - 支持打乱stage执行（模拟出硬件并行）：60%测试点得分+10%理解得分
    - 仅支持顺序执行：30%
  - forwarding：12%
  - 二位饱和预测：8%
  - Bonus：10%
  - Cr：10%
- Tomasulo
  - 实现分支预测的tomasulo算法：75%测试点得分+15%理解得分
    - 若不支持分支预测：60%测试点得分+10%理解得分
  - Bonus：10%
  - Cr：10%
- 分数溢出部分忽略。
- 对电路设计的模拟也将在Cr中考察。
- bonus部分可以实现一些提高预测准确率的高级分支预测、模拟Cache、合理的多级流水、多发射。

可参考的资料

CAAQA, Computer Architecture: A Quantitative Approach

RV32I基础指令集https://www.cnblogs.com/mikewolf2002/p/11196680.html