厦門大學
XIAMEN UNIVERSITY

电子系统设计 Part4

RISC-V SOC系统设计(1)

信息与通信工程系 陈凌宇

chenly@xmu.edu.cn

导言

本部分概述了RISC-V处理器的基础知识，以一个样例工程为范本介绍了RISC-V C语言编程方法。在此样例的基础上添加必要外设，并编写程序实现一个音乐播放器。

目录

- RISC-V处理器
- 使用C语言编程
- 拓展实验（3）

背景

存在ARM，x86，MIPS，SPARC，Power等多种处理器架构

- 复杂且不开源：可能存在后门漏洞，无法安全可控（x86）
- 存在专利壁垒：昂贵的知识产权授权费用（ARM）
- 长期支持风险：商业公司可能被收购或者面临危机
- 技术标准推广普及存在问题: 缺乏完善的产业链生态支持（MIPS、Power）

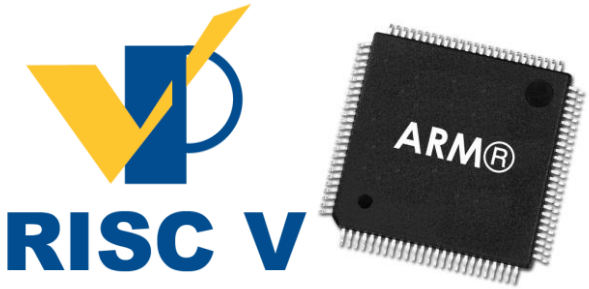
CPU架构	诞生时间/年
IBM 701	1953
CDC 6600	1963
IBM 360	1964
DEC PDP-8	1965
Intel 8008	1972
Motorola 6800	1974
DEC VAX	1977
Intel 8086	1978
Intel 80386	1985
ARM	1985
MIPS	1985
SPARC	1987
Power	1992
Alpha	1992
HP/Intel IA-64	2001
AMD64 (EMT64)	2003

RISC-V诞生

- 2010年发源于伯克利大学Krste教授项目
  - 全新的指令集架构，从RISC-I开始设计，不断迭代至RISC-V
- 特点
  - 指令集完全开源
  - 免费授权，不存在专利壁垒
  - 成立永久的非营利性基金会管理运营
  - 组织建立完整的上下游产业开发联盟
  - 每年召开两次workshop国际技术讨论大会

RISC体系结构

- RISC & CISC



RISC
(Reduced Instruction Set Computer)

CISC
(Complex Instruction Set Computer)

软硬件生态

工具链

GNU Toolchain，
LLVM，QEMU，
OpenOCD，JLINK，
….

操作系统

Linux Kernel，
Linux distributions，
FreeRTOS，uCOS-II，
RT-Thread，….

集成开发环境

Eclipse，IAR，
Embedded Studio，
MounRiver Studio，
….

开源内核实现

Rocket，BOOM，RI5CY，
Zero-riscy，PicoRV32，
Hummingbird E203 ….

商业内核提供公司

Nuclei，T-HEAD，
CloudBEAR，Syntacore
StarFive，SiFive ….

RISC-V指令集

- 基本整数指令集：
  - RV32I、RV64I、RV128I
- 扩展指令集指令集
  - 见右表
- 指令集简写惯例：
  - 如RV64IMAF

简称	描述
M	整数乘除法标准扩展
A	不可中断指令标准扩展
F	单精确度浮点运算标准扩展
D	双倍精确度浮点运算标准扩展
G	所有以上的扩展指令集以及基本指令集的总和的简称
Q	四倍精确度浮点运算标准扩展
L	十进制浮点运算标准扩展
C	压缩指令标准扩展
B	位操作标准扩展
J	动态指令翻译标准扩展
T	顺序存储器访问标准扩展
P	单指令多数据流运算标准扩充
V	向量运算标准扩充

RISC-V指令集

- 指令集分为以下几类:
  - R-type：寄存器-寄存器指令
  - I-type：寄存器-立即数指令
  - B-type：条件跳转指令
  - U-type：高位读入指令
  - S-type：访存指令
  - J-type：无条件跳转指令

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RISC-V指令集

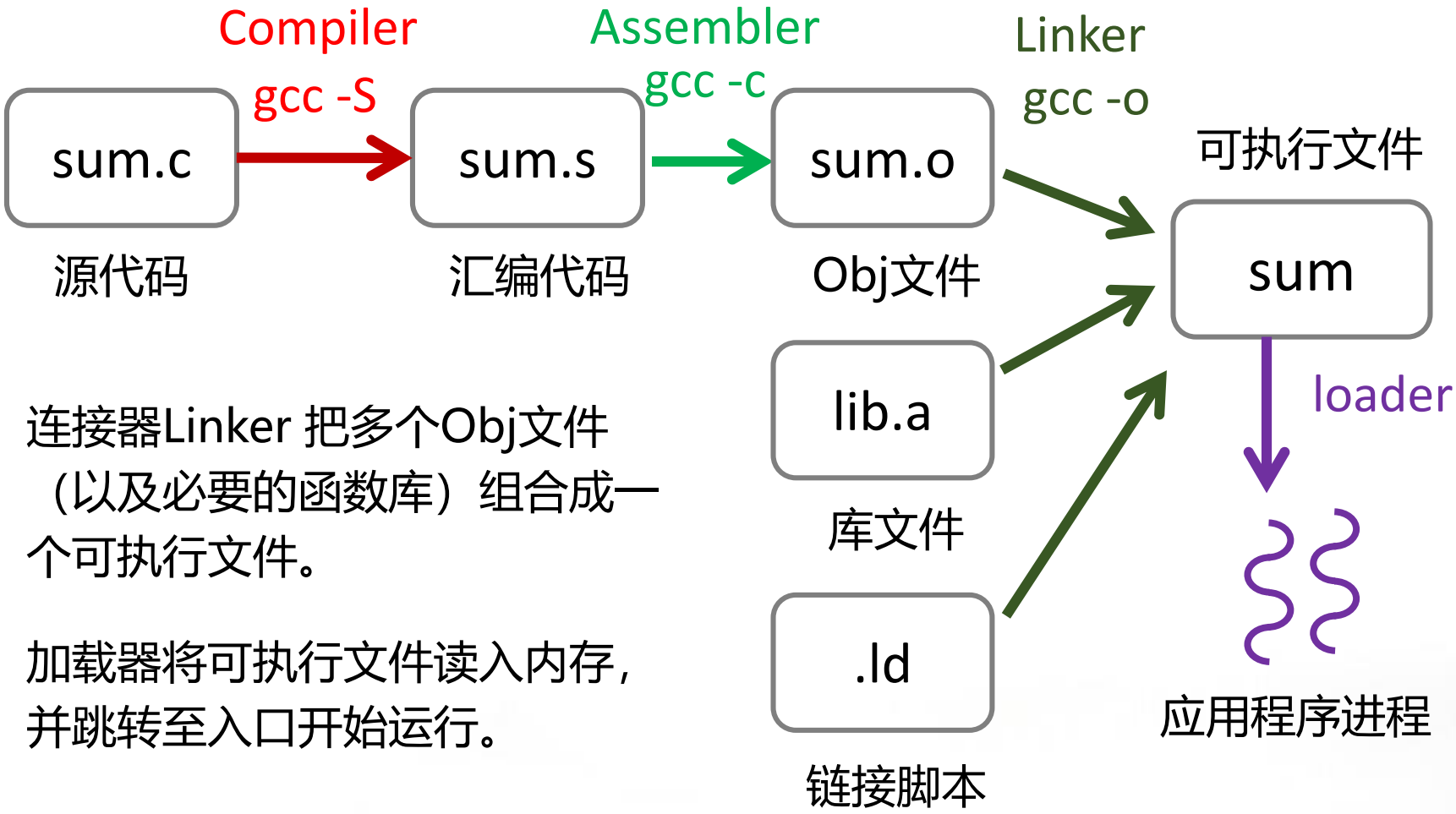
32个通用寄存器

寄存器	编程接口名称	描述	使用
x0	zero	Hard-wired zero	硬件零
x1	ra	Return address	常用于保存（函数的）返回地址
x2	sp	Stack pointer	栈顶指针
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5-7	t0-2	Temporary	临时寄存器
x8	s0/fp	Saved Register/ Frame pointer	（函数调用时）保存的寄存器和栈顶指针
x9	s1	Saved register	（函数调用时）保存的寄存器
x10-11	a0-1	Function argument/ return value	（函数调用时）的参数/函数的返回值
x12-17	a2-7	Function argument	（函数调用时）的参数
x18-27	s2-11	Saved register	（函数调用时）保存的寄存器
x28-31	t3-6	Temporary	临时寄存器

目录

- RISC-V处理器
- 使用C语言编程
- 拓展实验（3）

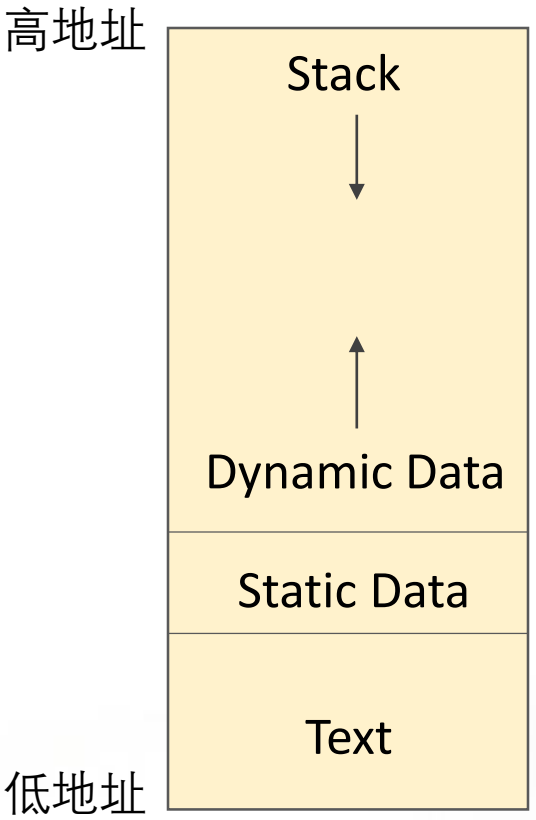
从编程到运行



- 连接器Linker 把多个Obj文件（以及必要的函数库）组合成一个可执行文件。
- 加载器将可执行文件读入内存，并跳转至入口开始运行。

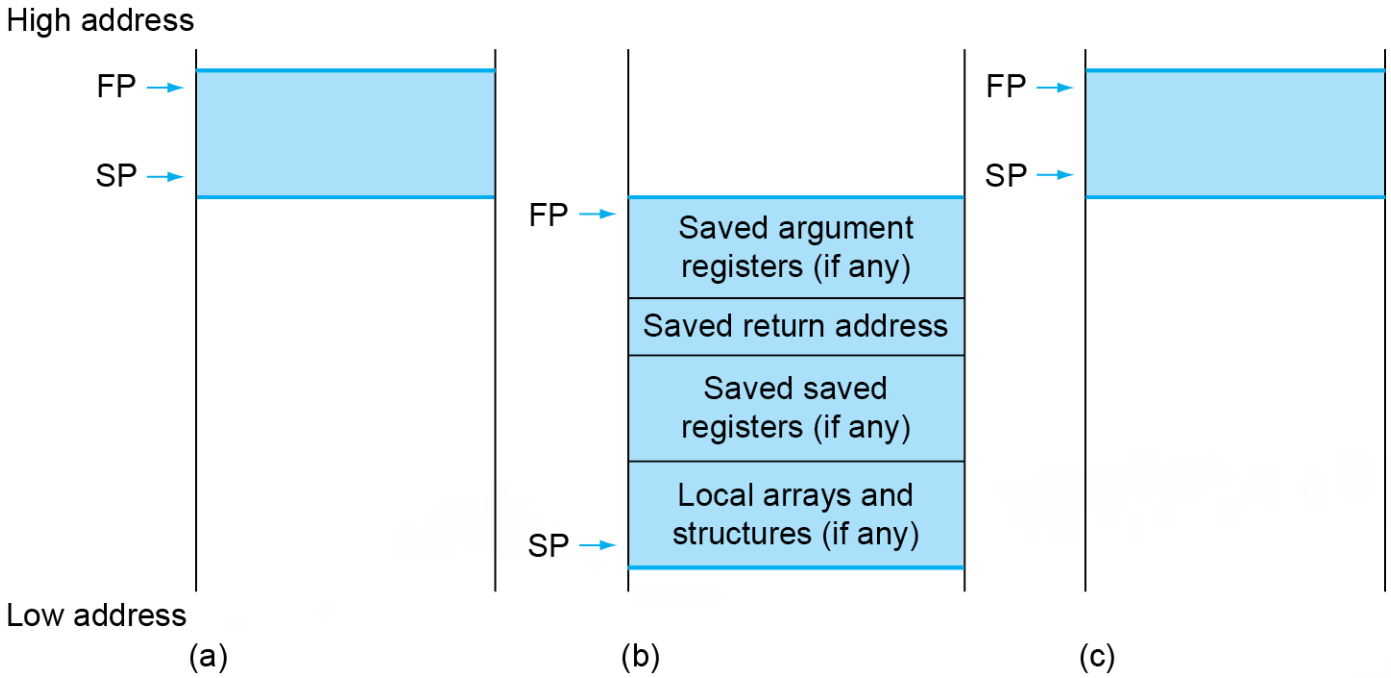
内存分配

- Text段：机器码
- 静态内存区：存放全局变量
  - RODATA 有初始值的全局变量/静态变量
  - BSS(Block Started by Symbol)未初始化的全局变量/静态变量
- 动态内存区：heap
  - 如C语言中的malloc操作
- 栈：局部变量，传参，返回值



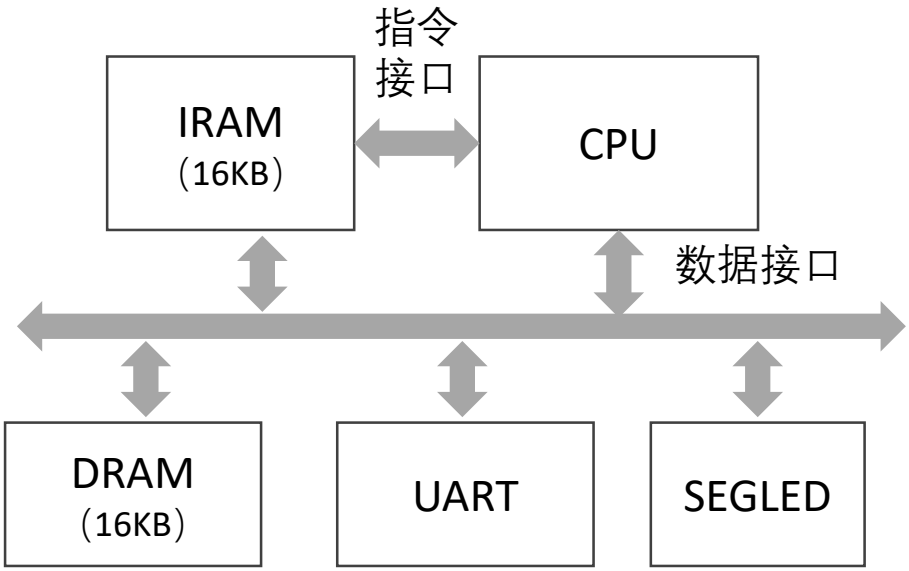
栈

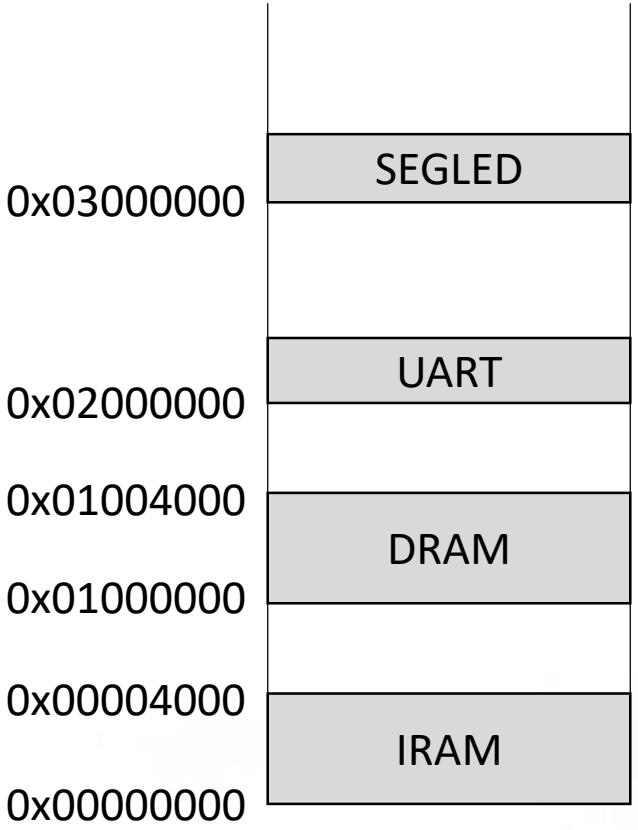
- 调用函数A时候，把参数压栈，然后跳转到函数A；
- 进入函数A，把需要保存的寄存器/返回地址压栈，同时开辟一段空间用于局部变量；
- 返回时，从栈中取出返回地址，从而返回调用函数。



系统结构

- 样例工程的内存映射





相关代码

- C语言的入口函数一般是main()
- 在进入main()函数之前，需要进行一系列初始化，该部分代码在start.s中
  - 完成BSS段/RODATA段的初始化
  - CPU关键寄存器的初始化，中断向量表初始化等
- 样例代码

外设的访问

- 定义访问地址，直接进行赋值操作

```
#define UART0_BASE      0x02000000
#define UART0_CTRL      0x00
#define UART0_STATUS    0x04
#define UART0_BAUD      0x08
#define UART0_TXDATA    0x0c
#define UART0_RXDATA    0x10

#define UART0_REG(addr) \
     (*((volatile uint32_t *)(addr+UART0_BASE)))


//.....
     while (UART0_REG(UART0_STATUS) & 0x1);
     UART0_REG(UART0_TXDATA) = c;
```

目录

- RISC-V处理器
- 使用C语言编程
- 拓展实验（3）

拓展项目

- 实现一个音乐播放器，要求如下
  - 使用串口终端输入一段乐谱，格式自定，如：

    乐谱用0-7的数字串代表，0代表休止，其他分别为do..xi，每个数字固定为1/4拍，按回车结束。

  - 利用蜂鸣器演奏，完成后等待新的一段乐谱输入。

拓展项目

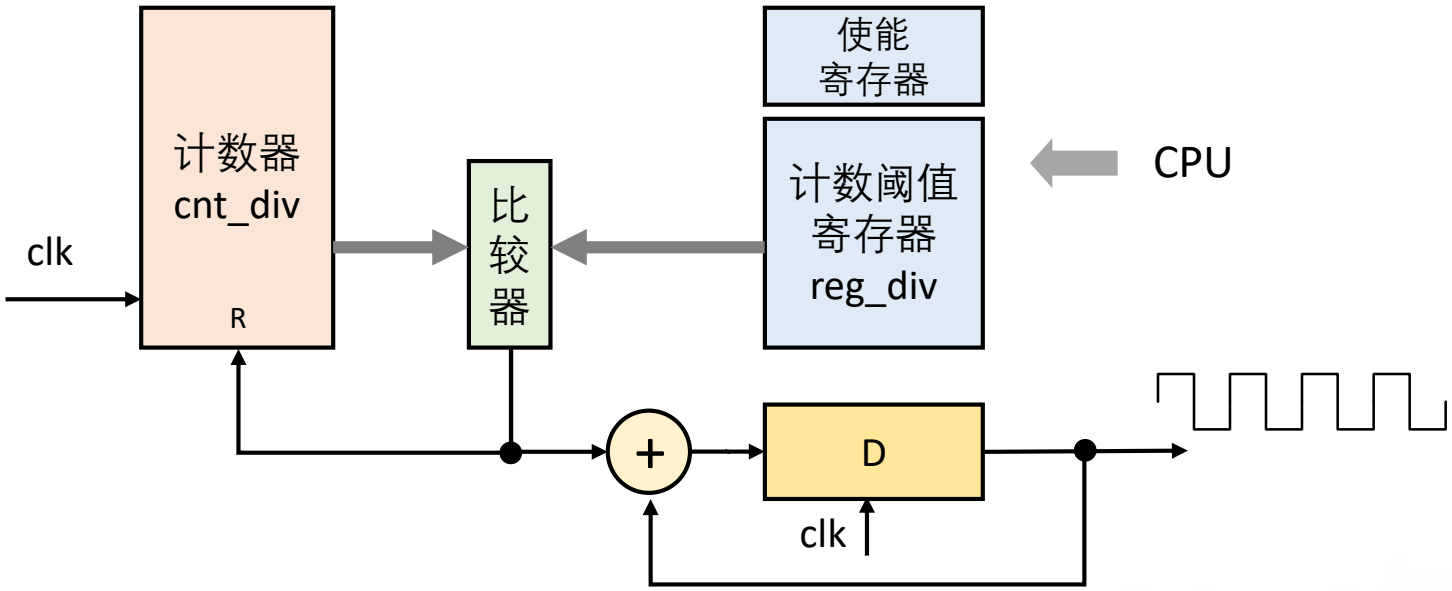
厦門大學 XIAMEN UNIVERSITY

- 音阶生成
  - 利用50MHz信号进行分频，获得不同频率的声音

音高	频率(Hz)	50M分频系数	实际频率(Hz)	音高	频率(Hz)	50M分频系数	实际频率(Hz)
C4	261.6256	95556	261.63	C5	523.25	47778	523.25
D4	293.6648	85131	293.67	D5	587.33	42566	587.32
E4	329.6276	75843	329.63	E5	659.26	37921	659.27
F4	349.2282	71586	349.23	F5	698.46	35793	698.46
G4	391.9954	63776	392	G5	783.99	31888	783.99
A4	440	56818	440	A5	880	28409	880
B4	493.8833	50619	493.89	B5	987.77	25310	987.75

拓展项目

- 不同频率声音的发生



Question ?