项目进阶

stm32

```
DMA -
flash - spi-外接    内部
freeRTOS - RTOS 实时操作系统
cubeIDE / cubeMX - 软件 hal库 ll库
图像处理库 - 屏幕 - 显示
```

云平台

```
物联网lot  -  阿里云-设备开发平台 mqtt http
oneNET - 中国移动-设备开发平台 mqtt http tcp

百度云 - 人脸识别/车牌/物品/二维码/语音/文字....
天气
ai
```

QT

```
自己利用一些现有的库实现
    opencv

qt自己接入大模型
```

ARM接口技术

```
arm - 公司/芯片技术 - 芯片的学习
    三类: A-高性能    R-实时性高    M-低功耗

    exynos4412  -  cortex-A9
    从0开始 - 汇编, 寄存器编程, 点灯

接口技术
    GPIO
    ADC
    TIM
    NVIC
    UART
    RTC
    WDT
```

芯片 - 芯片的组成 - 硬件相关知识
    门电路: 由 与 或 非 异或 等门电路组成
            可以是单独的芯片: 74ls138 74HC594 .... (辅助芯片 半加器 选择器 38译码器 串转并 锁存器....)

    可编程的芯片: 计算机 加法
        门电路 - 运算器

    cpu: 芯片的大脑
    cpu的组成: 运算器 控制器 存储器(寄存器) 总线

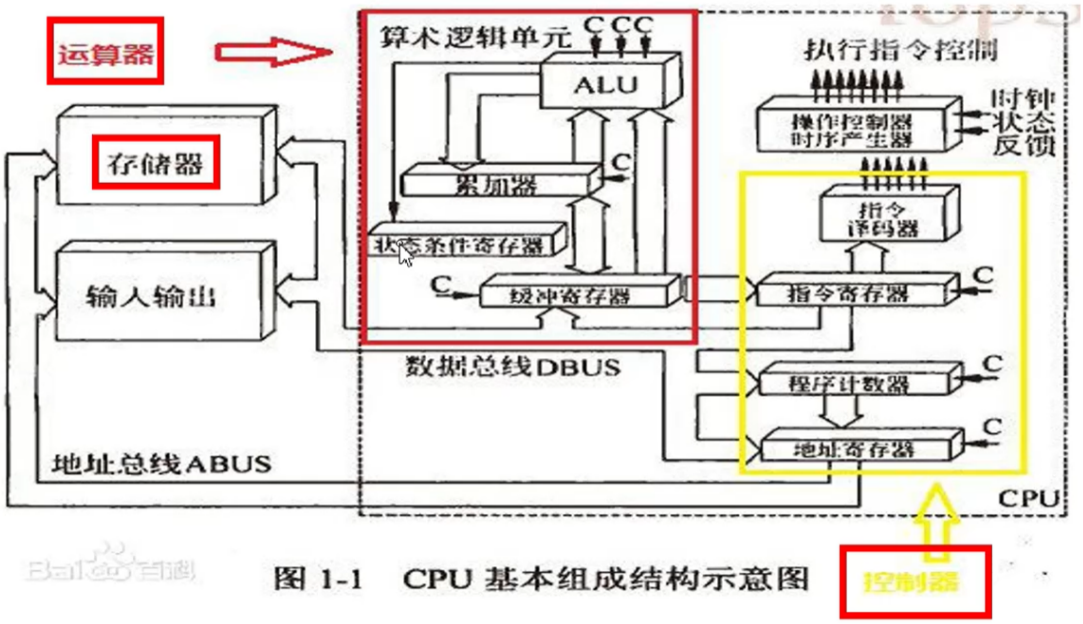


图 1-1 CPU 基本组成结构示意图

芯片: 有特定功能的模块单元

芯片的组成: (soc芯片 - system on chip)
    cpu + 存储器 + 专用控制器(GPIO TIM ADC DMA UART IIC SPI RTC WDT .....)
    电脑芯片: cpu + 存储器 + 专用控制器 + GPU + NPU + GPS .....

计算机的组成:
    核心芯片 + 输入 + 输出 + 存储器 + 总线 + 运算器 + 控制器

    输入输出: 键盘 鼠标 麦克风 摄像头   屏幕   喇叭
    存储器: register cache ram 硬盘 网盘

RISC: reduced instruction set computer (精简指令集计算机) arm
CISC: complex instruction set computer (复杂指令集计算机) x86

常见芯片架构: 芯片的图纸 (cpu架构)
    c51/52: 低功耗、价格低、成本低、简单
    x86: 性能高、速度快、兼容性好
    ARM: 成本低、低功耗 在这基础上最求高性能
    MIPS: 简洁、优化方便、高拓展性 天然64位
    PowerPc: 方便灵活、可伸缩性好、功耗极高
    RISC-V: 模块化、极简、可拓展、开源

了解arm芯片架构发展

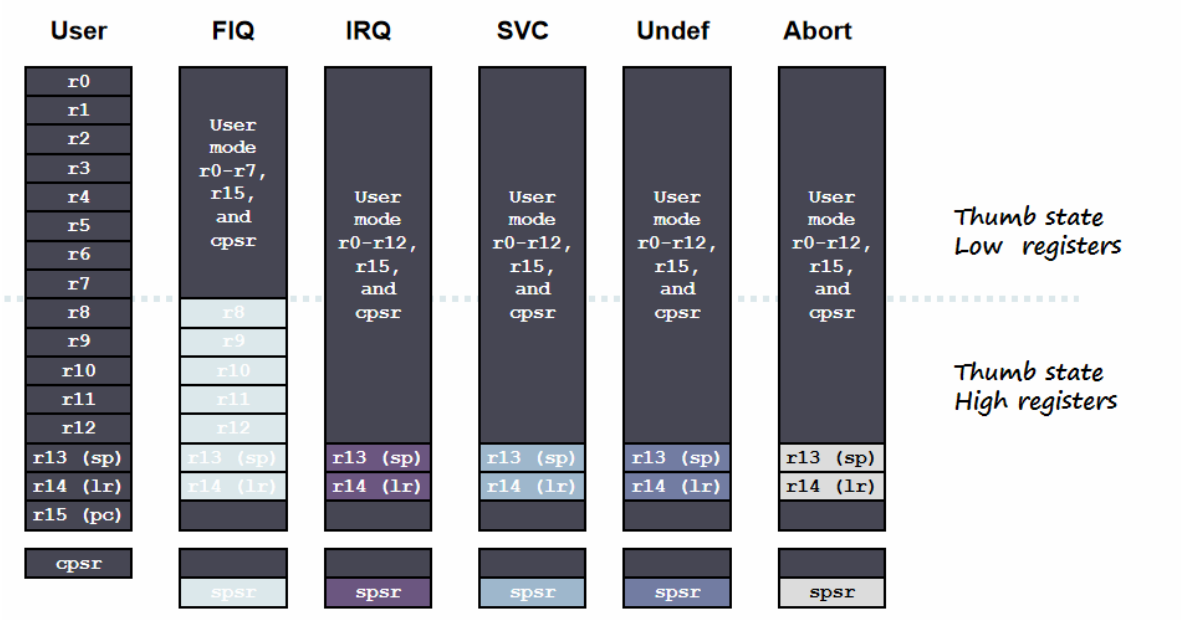
三级流水线：
最佳流水线：

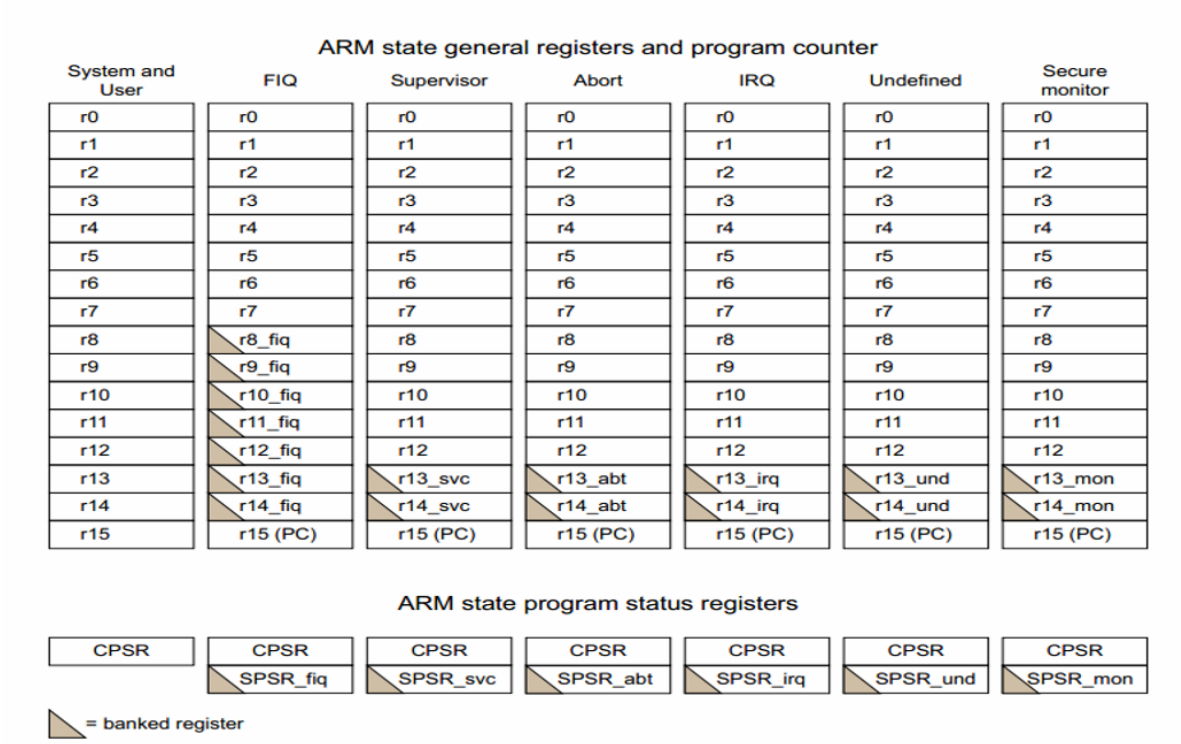
ARM 有8个基本工作模式：
    User ： 非特权模式，大部分任务执行在这种模式
    FIQ ： 当一个高优先级（fast）中断产生时将会进入这种模式
    IRQ ： 当一个低优先级（normal）中断产生时将会进入这种模式
    Supervisor ：当复位或软中断指令执行时将会进入这种模式(SVC)
    Abort ： 当存取异常时将会进入这种模式
    Undef ： 当执行未定义指令时会进入这种模式
    System ： 使用和User模式相同寄存器集的特权模式
    Cortex-A特有模式：
        Monitor ： 是为了安全而扩展出的用于执行安全监控代码的模式；也是一种特权模式

寄存器：存储数据 32位
    每种模式有自己的寄存器集

寄存器的名字： 40个
    基础寄存器：r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 r15
        r13(sp)：栈指针寄存器：保存(内存)栈地址
        r14(lr)：链接寄存器：保存程序的返回地址(函数的返回地址)
        r15(pc)：程序计数器寄存器：保存的是正被取指的指令地址，而非正在执行的指令

    当前程序状态寄存器(current program staius register)：cpsr (模式 状态 计算结果 中断禁止 大小端)
    保存程序状态寄存器(save program staius register)：spsr (进行模式切换时，保存上一个模式的状态)





操作寄存器：写代码

C语言：register int i = 0; x

汇编：指令

搬移指令：

```
MOV  R0, #1
MOV  R1, #2
```

看效果：

模拟开发板 - Ubuntu内部 gdb调试，查看寄存器的值

模拟开发板 - windows上 keil 4 软件，

环境搭建

1.交叉编译工具安装

ubuntu 输入： arm-linux-gcc -v 查看交叉编译工具是否安装好了

自己安装：

方法一：

先安装32位的兼容包

1、解压 gcc-4.6.4.tar.xz 到 主目录（家目录：/home/hqyj/ 根据自己Ubuntu命名）

```
tar -xvf  gcc-4.6.4.tar.xz
```

2、确认解压完成，可以进入~/gcc-4.6.4/bin 下面看到 arm-linux-gcc 命令

```
cd   ~/gcc-4.6.4/bin
pwd 获取绝对路径  /home/hqyj/gcc-4.6.4/bin
```

3、修改配置文件~/.bashrc，添加环境变量

(1).vi ~/.bashrc ,在最后一行添加 路径消息

```
export PATH=$PATH:/home/hqyj/gcc-4.6.4/bin
```

(2).保存退出后:wq，执行 source ~/.bashrc

(3).关闭所有终端

方法二：

安装32位的兼容包

在线安装： 自己查一下 Ubuntu 20.04 安装arm-linux-gcc交叉编译工具

2.模拟开发板的安装

```
sudo apt-get update
sudo apt-get install qemu-system
```

3.测试

```
【 写代码 汇编文件 .S的后缀 】
新建一个目录:    vim test.S
.global    _start
_start:
    mov r1, #3
    nop
    nop

【 程序编译 】
    arm-linux-gcc   test.S   -o   test.o    -c    -g
    arm-linux-ld    test.o   -o   test.elf  -Ttext 0x00000000

    -Ttext 0x00000000     程序运行的起始地址，模拟开发板的地址是0x0

【 开启虚拟目标板 】
在第一个终端执行下面命令:   可封装成脚本
    qemu-system-arm  -machine  xilinx-zynq-a9  -m  256M  -serial  stdio  -kernel
test.elf  -S  -s

【 调试端 - 看执行结果】
再开一个终端，在另外一个终端   执行下面命令:
    export LC_ALL=C
    arm-none-linux-gnueabi-gdb      test.elf
进入GDB后，执行
(gdb) target  remote  127.0.0.1:1234
(gdb) s

命令解释:
l:  查看代码
s: 单步调试，会进入函数体
n: 单步调试，不进入函数体
b  11:  在11行设置断点
c:  继续运行，运行到断点位置停止
p:  显示变量值    p  $r1 十进制打印;    p/x   val    十六进制打印
x:  显示内存值    x/4   0x10    显示内存地址0x10 开始的4块连续地址
info  locals: 查看当前栈中所有局部变量的值
q:  退出
```