项目操作篇

第 4 章

裸机开发

4.1 概述

对于功能比较简单的嵌入式产品，出于成本等原因的考虑，一般不需要安装嵌入式操作系统，在这种情况下进行应用程序的开发，即为裸机编程。裸机编程需直接面对硬件，对硬件进行操作。

裸机编程必须掌握以下知识和技能。

（1）电子电路基本知识，包括数字电路和模拟电路。

（2）Linux 操作系统知识及基本操作。

（3）C 语言编程技能。

（4）汇编语言编程技能。

（5）嵌入式开发环境搭建，包括硬件环境、交叉编译器安装以及串口、网络等的工具使用。

（6）ARM 体系结构及特殊功能寄存器。

（7）ARM 指令系统。

本章先学习嵌入式开发环境搭建、ARM 体系结构及特殊功能寄存器、ARM 指令系统，在此基础上，通过几个范例学习裸机编程方法。

其中，特殊功能寄存器的使用是学习裸机程序开发的关键。

4.2 建立 Linux 开发环境

开发裸机程序，一般都选用 ADS1.2 或者 MDK，但这些工具都是针对 ARM9 平台的，对于 Cortex-A8 就不支持了，所以选择在 Linux 下开发。

嵌入式软件开发的一个显著特点就是需要交叉开发环境（Cross Development Env）的支持，交叉编译器只是交叉开发环境的一部分。交叉开发环境是指编译、链接和调试嵌入式应用软件的环境，它与运行嵌入式应用软件的环境有所不同。

嵌入式系统通常是一个资源受限的系统，因此直接在嵌入式系统的硬件平台上编写软件比较困难，有时甚至是不可能的，解决办法通常都是采取交叉编译模式。

在 2.7 节介绍过交叉编译器，它能实现在 PC 平台（X86 CPU）上编译出能运行在 ARM 平台上的程序，编译得到的程序在 X86 CPU 平台上是不能运行的，在 ARM 平台上才能运行。例如，arm-linux-gcc，表示基于 Linux 和 ARM 平台的交叉编译器。

为实现上述开发模式，需要搭建宿主机－目标机硬件平台，如图 4.1 所示。在宿主机（一般为 PC）上完成代码编写和编译后，通过串口线进行命令或控制，目标代码通过网线下载到目标机上运行。在教学中，目标机就是各种开发平台或实验箱。在本书中用到的实验箱是外购的 ARM CortexTM-A8 的实验箱。

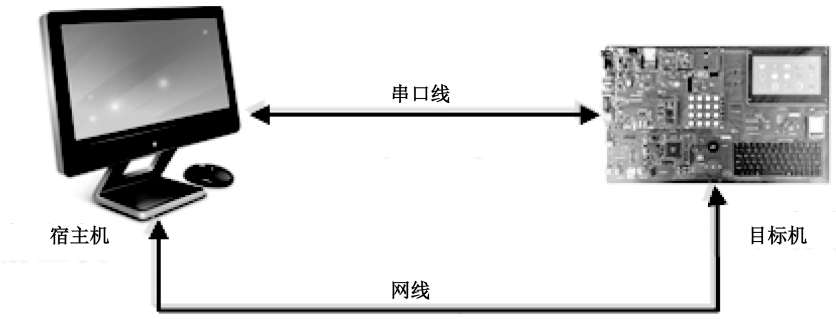

图 4.1　嵌入式软件开发的硬件环境

交叉编译环境所需工具主要包括：交叉编译器，如 arm-linux-gcc；交叉汇编器，如 arm-linux-as；交叉链接器，如 arm-linux-ld；各种操作所依赖的库；用于处理可执行程序和库的一些基本工具，如 arm-linux-strip，如图 4.2 所示。

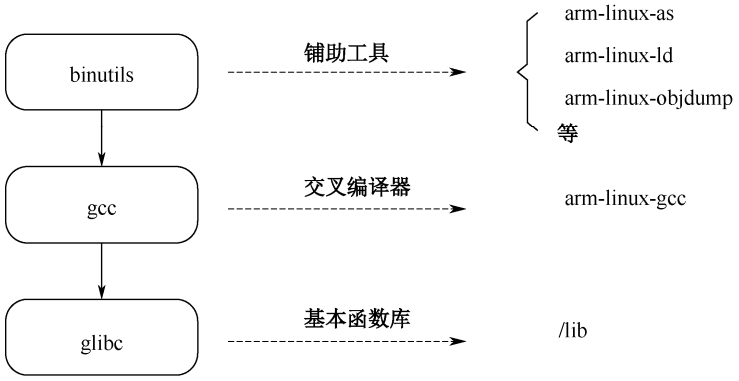

图 4.2　交叉环境工具

交叉环境工具的名称及作用，如表 4.1 所示。

表 4.1　交叉环境工具一览表

名称	归属	作用
arm-linux-as	binutils	编译 ARM 汇编程序
arm-linux-ar	binutils	把多个.o 合并成一个.o 或静态库（.a）
arm-linux-ranlib	binutils	为库文件建立索引，相当于 arm-linux-ar-s
arm-linux-ld	binutils	连接器（Linker），把多个.o 或库文件连接成一个可执行文件
arm-linux-objdump	binutils	查看目标文件（.o）和库（.a）的信息
arm-linux-objcopy	binutils	转换可执行文件的格式
arm-linux-strip	binutils	去掉 elf 可执行文件的信息，使可执行文件变小
arm-linux-readelf	binutils	读 elf 可执行文件的信息
arm-linux-gcc	gcc	编译.c 或.S 开头的 C 程序或汇编程序
arm-linux-g++	gcc	编译 C++程序

在使用交叉编译工具过程中，如出现工具使用异常（如提示未找到命令），可在操作系统中输入“ $PATH ”命令查询编译器路径配置情况，并根据需要配置交叉编译器的环境变量，参考 2.6 节。

根据需要，可对交叉编译工具如 arm-linux-gcc、arm-linux-objcopy、arm-linux-ld、arm-linux-objdump 等作链接处理，方法如下：

```
#cd /usr/local/arm/4.5.1/bin                                          //进入编译器路径
#ln -s arm-none-linux-gnueabi-objcopy arm-linux-objcopy  //作链接
#ln -s arm-none-linux-gnueabi-ld arm-linux-ld
#ln -s arm-none-linux-gnueabi-objdump arm-linux-objdump
#ls -la                                                                  //显示指向信息
```

4.3　S5PV210 介绍

4.3.1　S5PV210 简介

ARM 微处理器有 ARM7、ARM9、ARM10E、SecurCore、ARM11、Cortex-A8、Cortex-A9、Cortex-A15 等系列。本书采用的操作是基于 ARM CortexTM-A8 内核，微处理器芯片型号是 S5PV210。

S5PV210 芯片又名“蜂鸟”（Hummingbird），是三星推出的一款适用于智能手机和平板电脑等多媒体设备的应用处理器，主频可达 1GHz，64/32 位内部总线结构，32/32KB 的数据/指令一级缓存，512KB 的二级缓存，可以实现 2000DMIPS（每秒运算 2 亿条指令集）的高性能运算能力。包含很多强大的硬件编解码功能，内建 MFC（Multi Format Codec），支持 MPEG-1/2/4、H.263、H.264 等格式视频的编解码，支持模拟/数字 TV 输出。JPEG 硬件编解码，最大支持 8000×8000 分辨率。内建高性能 PowerVR SGX540 3D 图形引擎和 2D 图形引擎，支持 2D/3D 图形加速，是第五代 PowerVR 产品，其多边形生成率为 2800 万多边形/秒，像素填充率可达 2.5 亿/秒，在 3D 和多媒体方面比以往大幅提升，能够支持 DX9、SM3.0、OpenGL2.0 等 PC 级别显示技术。具备 IVA3 硬件加速器，具备出色的图形解码性能，可以支持全高清、多标准的

视频编码，流畅播放和录制 30 帧/秒的 1920×1080 像素（1080p）的视频文件，可以更快解码更高质量的图像和视频，同时，内建的 HDMI v1.3，可以将高清视频输出到外部显示器上。

S5PV210的存储控制器支持LPDDR1、LPDDR2和DDR2类型的RAM，Flash支持NandFlash、NorFlash、OneNand等，外围接口丰富，包括：4个UART接口；4-Timers with PWM；2路SPI；1路USB HOST；1路USB OTG；触摸屏液晶接口；数字视频输出接口；数字音频输出接口；VGA 接口；以太网接口；SD卡接口；Audio接口；HDMI高清数字接口。

S5PV210 芯片是 584 引脚的 FCFBGA 封装，引脚间距 0.65mm，体积为 17×17mm，其系统架构如图 4.3 所示。

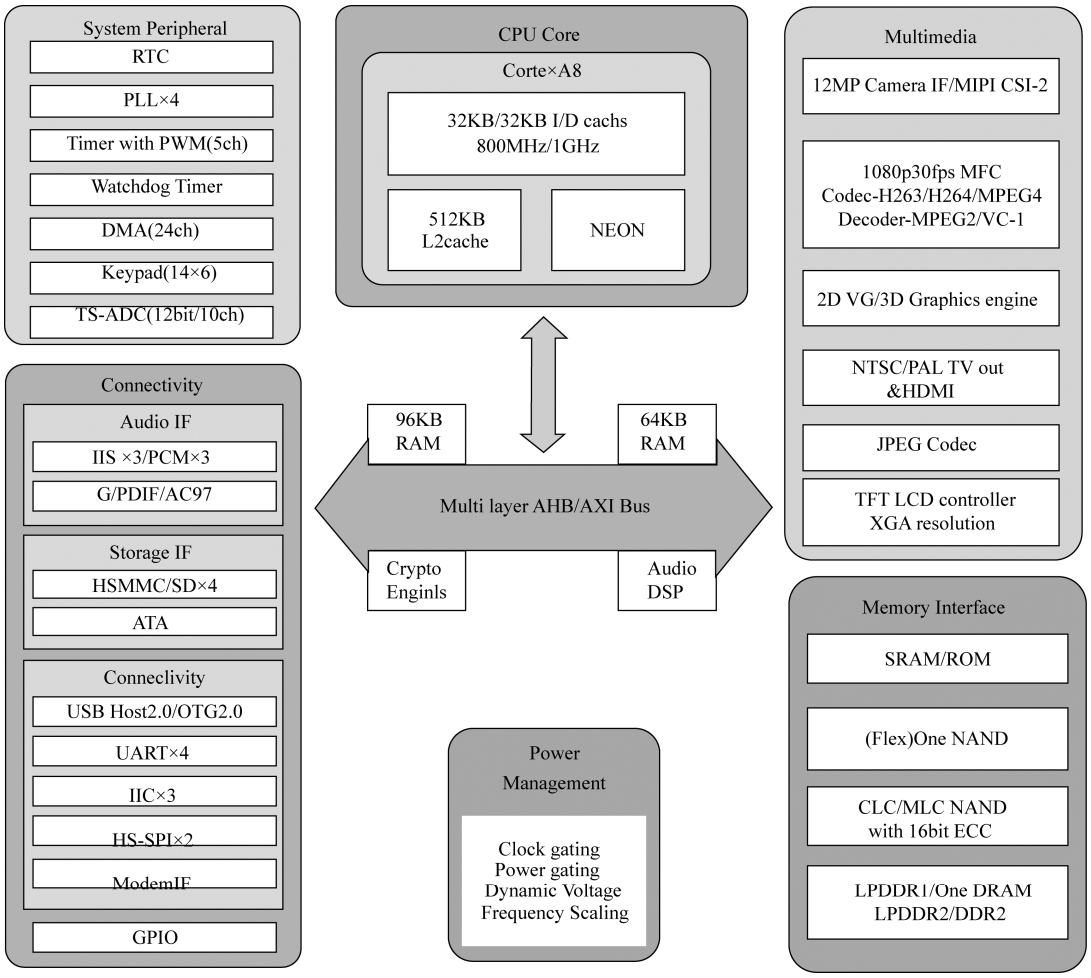

图 4.3　S5PV210 系统架构图

图 4.3 中主要模块英文备注：CPU Core（CPU 内核）、Power Management（电源管理）、Multimedia（多媒体）、System Peripheral（系统外设）、Connectivity（连通性）、Memory Interface（内存接口）、Multi layer AHB/AXI Bus（多层 AHB/AXI 总线）。

4.3.2　S5PV210 内存空间

S5PV210内存空间使用有特别规定，如表4.2所示，进一步了解可查阅该芯片的数据手册。

表 4.2　S5PV210 内存空间分布

地址范围（起止）		大小	用途描述
0x0000_0000	0x1FFF_FFFF	512MB	Boot area
0x2000_0000	0x3FFF_FFFF	512MB	DRAM 0
0x4000_0000	0x7FFF_FFFF	1024MB	DRAM 1
0x8000_0000	0x87FF_FFFF	128MB	SROM Bank 0
0x8800_0000	0x8FFF_FFFF	128MB	SROM Bank 1
0x9000_0000	0x97FF_FFFF	128MB	SROM Bank 2
0x9800_0000	0x9FFF_FFFF	128MB	SROM Bank 3
0xA000_0000	0xA7FF_FFFF	128MB	SROM Bank 4
0xA800_0000	0xAFFF_FFFF	128MB	SROM Bank 5
0xB000_0000	0xBFFF_FFFF	256MB	OneNAND/NAND Controller and SFR
0xC000_0000	0xCFFF_FFFF	256MB	MP3_SRAM output buffer
0xD000_0000	0xD000_FFFF	64KB	IROM
0xD001_0000	0xD001_FFFF	64KB	Reserved
0xD002_0000	0xD003_7FFF	96KB	IRAM
0xD800_0000	0xDFFF_FFFF	128MB	DMZ ROM
0xE000_0000	0xFFFF_FFFF	512MB	SFR region

4.3.3　S5PV210 特殊功能寄存器

正确使用芯片的特殊功能寄存器（SFR）是裸机程序开发的关键，S5PV210 I/O的特殊功能寄存器地址范围为0x0E000000～0xFFFFFFFF，通过对寄存器进行操作（读/写），实现对相关硬件功能进行配置及控制，具体功能及使用定义可查阅该芯片的数据手册。

S5PV210有多个通用I/O口（GPIO），每个I/O端口可定义为不同功能使用，通常可以用做输入口（input）、输出口（output）以及特殊功能口（如串口、中断信号）等，下面以端口GPA0为例。

端口GPA0有8个引脚，每个引脚的功能可以单独定义、单独使用。

控制寄存器GPA0CON（可读/写，地址=0xE020_0000）用于定义引脚的功能，可写入32位数据，写入不同数据，即定义引脚的不同功能。如表4.3所示，GPA0CON[n]中的n代表GPA0的第n个引脚，每4位数据定义一个引脚的功能，写入不同的数据，可定义为输入、输出、串口、中断等不同的功能。初始化状态32位数据都为0，各引脚均定义为输入功能。

表 4.3　GPA0CON 功能描述表

GPA0CON	数据位	功能描述	初始状态
GPA0CON[7]	[31:28]	0000=输入 0001=输出 0010=UART_1_RTSn 0011～1110=保留 1111=GPA0_INT[7]	0000
GPA0CON[6]	[27:24]	0000=输入 0001=输出 0010=UART_1_CTSn 0011～1110=保留 1111=GPA0_INT[6]	0000

续表

GPA0CON	数据位	功能描述	初始状态
GPA0CON[5]	[23:20]	0000=输入 0001=输出 0010=UART_1_TXD 0011～1110=保留 1111=GPA0_INT[5]	0000
GPA0CON[4]	[19:16]	0000=输入 0001=输出 0010=UART_1_RXD 0011～1110=保留 1111=GPA0_INT[4]	0000
GPA0CON[3]	[15:12]	0000=输入 0001=输出 0010=UART_0_RTSn 0011～1110=保留 1111=GPA0_INT[3]	0000
GPA0CON[2]	[11:8]	0000=输入 0001=输出 0010=UART_0_CTSn 0011～1110=保留 1111=GPA0_INT[2]	0000
GPA0CON[1]	[7:4]	0000=输入 0001=输出 0010=UART_0_TXD 0011～1110=保留 1111=GPA0_INT[1]	0000
GPA0CON[0]	[3:0]	0000=输入 0001=输出 0010=UART_0_RXD 0011～1110=保留 1111=GPA0_INT[0]	0000

数据寄存器 GPA0DAT（可读/写，地址=0xE020_0004）用于端口数据输入/输出，8 位数据与 GPA0 端口的 8 个引脚对应。如读取该寄存器，可获得端口 8 个引脚的数据；如向该寄存器写 8 位数据，这 8 位数据将出现在端口的 8 个引脚上。

上/下拉电阻寄存器 GPA0PUD（可读/写，地址=0xE020_0008）用于配置端口 8 个引脚上/下拉电阻，实现具体应用中外部电路与芯片引脚的匹配。如表 4.4 所示，GPA0PUD[n]中的 n 代表 GPA0 的第 n 个引脚，每 2 位数据配置一个引脚的上/下拉电阻，写入不同的数据，可配置为禁用、下拉、上拉等不同的功能。初始化状态 16 位数据为 0x5555，各引脚均配置下拉电阻。

表 4.4　GPA0PUD 功能描述表

GPA0PUD	数据位	功能描述	初始状态
GPA0PUD[n]	[2n+1:2n] n=0～7	00 =上拉/下拉禁止 01 =下拉允许 10 =上拉允许 11 = 保留	0x5555

4.4 ARM 常用指令集

4.4.1 ARM 寻址方式

1．寄存器寻址

操作数的值在寄存器中，指令中的地址码字段指出的是寄存器编号，指令执行时直接取出寄存器值操作。例如：

```
MOV  R1, R2        ;R2->R1
SUB  R0, R1, R2    ;R1-R2 -> R0
```

2．立即寻址

立即寻址指令中的操作码字段后面的地址码部分就是操作数本身，也就是说，数据就包含在指令当中，取出指令就取出了可以立即使用的操作数。例如：

```
SUBS R0, R0, #1      ;R0-1 -> R0
MOV  R0, #0xff00     ;0xff00 -> R0
```

注意：立即数要以“#”为前缀，表示十六进制数值时以“0x”表示。

3．寄存器偏移寻址

寄存器偏移寻址是 ARM 指令集特有的寻址方式，当第二个操作数是寄存器偏移方式时，第二个寄存器操作数在与第一个操作数结合之前选择进行移位操作。例如：

```
MOV  R0, R2, LSL #3        ;R2的值左移3位，结果存入R0，即R0 = R2 * 8。
ANDS R1, R1, R2, LSL R3    ;R2的值左移R3位，然后和R1相与操作，结果放入R1
```

寄存器偏移寻址可采用的移位操作如下。

LSL（Logical Shift Left）逻辑左移，寄存器中字的低端空出补0。

LSR（Logical Shift Right）逻辑右移，寄存器中字的高端空出补0。

ASR（Arthmetic Shift Right）算术右移，移位中保持符号位不变，即如果源操作数为正数，字高端空出补0，否则补1。

ROR（Rotate Right）循环右移，由字的低端移出的位填入高端空出的位。

RRX（Rotate Right eXtended by 1 place），操作数右移一位，左侧空位由CPSR的C填充。

4．寄存器间接寻址

寄存器间接寻址指令中的地址码给出的是一个通用寄存器的编号，所需要的操作数保存在寄存器指定地址的存储单元中，即寄存器为操作数的地址指针。例如：

```
LDR R1, [R2]          ;将R2中的数值作为地址，取出此地址中的数据保存在R1中
SWP R1, R1, [R2]      ;将R2中的数值作为地址，取出此地址中的数值与R1中的值交换
```

5．基址寻址

将基址寄存器的内容与指令中给出的偏移量相加，形成操作数的有效地址，基址寻址用于访问基址附近的存储单元，常用于查表、数组操作、功能部件寄存器访问等。例如：

```
LDR R2, [R3, #0x0F]       ;R3的数值加0x0F作为地址，取此地址的数据存入R2
STR R1, [R0, #-2]         ;R0中的数值减2作为地址，把R1中的数据存入此地址
```

6．多寄存器寻址

一次可以传送几个寄存器值，允许一条指令传送 16 个寄存器的任何子集或所有寄存器。

例如：

```
LDMIA R1!, {R2-R7, R12}  ；R1指向地址的数据读出到R2-R7, R12, R1自动更新
STMIA R0!, {R3-R6, R10}  ；R3-R6,R10中的数值保存到R0指向的地址, R0自动更新
```

7．堆栈寻址

堆栈是特定顺序进行存取的存储区，堆栈寻址时隐含地使用一个专门的寄存器（堆栈指针），指向一块存储区域（堆栈）。存储器堆栈可分为以下两种。

（1）向上生长：向高地址方向生长，称为递增堆栈。

（2）向下生长：向低地址方向生长，称为递减堆栈。

如此可结合出四种情况。

① 满递增：堆栈通过增大存储器的地址向上增长，堆栈指针指向内含有效数据项的最高地址，指令如 LDMFA、STMFA。

② 空递增：堆栈通过增大存储器的地址向上增长，堆栈指针指向堆栈上的第一个空位置，指令如 LDMEA、STMEA。

③ 满递减：堆栈通过减小存储器的地址向下增长，堆栈指针指向内含有效数据项的最低地址，指令如 LDMFD、STMFD。

④ 空递减：堆栈通过减小存储器的地址向下增长，堆栈指针指向堆栈下的第一个空位置，指令如 LDMED、STMED。

例如：

```
STMFD SP!,{R1-R7,LR}  ；将R1-R7, LR入栈, 满递减堆栈
LDMFD SP!,{R1-R7,LR}  ；数据出栈, 放入R1-R7,LR寄存器, 满递减堆栈
```

8．块复制寻址

用于一块数据从存储器的某一位置复制到另一位置。例如：

```
STMIA R0!,{R1-R7}
；将R1-R7的数据保存到存储器中，存储器指针在保存第一个值之后增加，方向为向上增长
STMIB R0!,{R1-R7}
；将R1-R7的数据保存到存储器中，存储器指针在保存第一个值之前增加，方向为向上增长
SIMDA R0!,{R1-R7}
；将R1-R7的数据保存到存储器中，存储器指针在保存第一个值之后增加，方向为向下增长
STMDB R0!,{R1-R7}
；将R1-R7的数据保存到存储器中，存储器指针在保存第一个值之前增加，方向为向下增长
```

不论是向上还是向下递增，存储时高编号的寄存器放在高地址的内存，出来时，高地址的内容给编号高的寄存器。

9．相对寻址

相对寻址是基址寻址的一种变通，由程序计数器PC提供基准地址，指令中的地址码字段作为偏移量，两者相加后得到的地址即为操作数的有效地址。例如：

```
BL ROUTE1     ；调用到 ROUTE1 子程序
BEQ LOOP      ；条件跳转到 LOOP 标号处
```

4.4.2 ARM 指令集

ARM 指令集可以分为数据处理指令、数据加载和存储指令、分支指令、程序状态寄存器

（PSR）处理指令、协处理器指令和异常产生指令六大类。

指令格式为：

```
<opcode> {<cond>}{S}<Rd>, <Rn>, {<operand2>}
```

其中：

<>内的项是必需的，{}内的项是可选的。

opcode：指令助记符，如 LDR、STR 等。

cond：执行条件，如 EQ、NE 等。

S：影响 CPSR 寄存器的值，有 S 影响 CPSR，否则不影响。

Rd：目标寄存器。

Rn：第一个操作数的寄存器。

operand2：第二个操作数。

指令格式举例如下：

```
LDR R0, [R1]            ;读取R1地址上的存储器单元内容，执行条件AL(无条件执行)
BEQ DATAEVEN            ;跳转指令，执行条件EQ，即相等跳转到DATAEVEN
ADDS R1, R1, #1         ;加法指令，R1+1 => R1 影响CPSR寄存器，带有S
SUBNES R1, R1, #0xD     ;条件执行减法运算（NE），R1-0xD => R1，影响CPSR
```

ARM 汇编条件码列表如表 4.5 所示。

表 4.5　ARM 汇编条件码列表

条件码助记符	标志	含义
EQ	Z=1	相等
NE	Z=0	不相等
CS/HS	C=1	无符号数大于或等于
CC/LO	C=0	无符号数小于
MI	N=1	负数
PL	N=0	正数
VS	V=1	溢出
VC	V=0	没有溢出
HI	C=1,Z=0	无符号数大于
LS	C=0,Z=1	无符号数小于或等于
GE	N=V	带符号数大于或等于
LT	N!=V	带符号数小于
GT	Z=0,N=V	带符号数大于
LE	Z=1,N!=V	带符号数小于或等于
AL		任何无条件执行（指令默认条件）

条件码应用举例如下。

（1）比较两个值大小，C 代码如下：

```
if(a>b) a++;
else b++;
```

写出相应的 ARM 指令，代码如下。

设 R0 为 a，R1 为 b，则：

```
CMP R0,  R1                ;R0与R1比较
```

```
ADDHI R0R0, #1      ;若R0>R1，则R0=R0+1
ADDLS R1, R1, #1    ;若R0<=R1，则R1=R1+1
```

（2）若两个条件均成立，则将这两个数值相加，C 代码为：

```
if((a!=10)&&(b!=20)) a=a+b;
```

对应的 ARM 指令为：

```
CMP R0, #10         ;比较R0是否为10
CMPNE R1, #20       ;若R0不为10，则比较R1是否为20
ADDNE R0, R0, R1    ;若R0不为10且R1不为20，则执行 R0 = R0+R1
```

（3）若两个条件有一个成立，则将这两个数值相加，C 代码为：

```
 if((a!=10)||(b!=20)) a=a+b;
```

对应的 ARM 指令为：

```
CMP R0, #10
CMPEQ R1, #20
ADDNE R0, R0, R1
```

1．数据加载和存储指令

数据加载和存储指令包含 LDR、STR、LDM、STM、SWP 指令。

（1）LDR/STR：加载/存储字和无符号字节指令，从寻址方式的地址计算方法分，加载/存储指令有以下 4 种形式。

零偏移：LDR Rd，[Rn]。

前索引偏移：LDR Rd,[Rn,#0x04]!，LDR Rd,[Rn,#-0x04] Rn 不允许为 R15。

程序相对偏移：LDR Rd,label，label 为程序标号，该形式不能使用后缀。

后索引偏移：LDR Rd,[Rn],#0x04，Rn 不允许是 R15。

指令举例如下：

```
LDR R2,[R5]         ;加载R5指定地址上的数据（字），放入R2中
STR R1,[R0,#0x04]
;将R1的数据存储到 R0+0x04存储单元，R0的值不变（若有!，则R0就要更新）
LDRB R3,[R2],#1     ;读取R2地址上的一字节数据并保存到R3中，R2=R2+1
STRH R1,[R0,#2]!    ;将R1的数据存入R0+2的地址中，只存储低2字节数据，R0=R0+2
```

（2）LDM 和 STM 是批量加载/存储指令，LDM 为加载多个寄存器，STM 为存储多个寄存器，主要用途是现场保护、数据复制、参数传递等，其模式有 8 种，前 4 种用于数据块的传输，后 4 种用于堆栈操作 。

IA：每次传送后地址加 4。

IB：每次传送前地址加 4。

DA：每次传送后地址减 4。

DB：每次传送前地址减 4。

FD：满递减堆栈。

ED：空递减堆栈。

FA：满递增堆栈。

EA：空递增堆栈。

批量加载/存储指令举例如下：

```
LDMIA R0!,{R3-R9}           ;加载R0指向的地址上的多字数据，保存到R3-R9中，R0值更新
STMIA R1!,{R3-49}           ;将R3-R9的数据存储到R1指向的地址上，R1值更新
```

```
STMFD SP!,{R0-R7,LR}        ;现场保存，将R0～R7、LR入栈
LDMFD SP!,{R0-R7,PC}^       ;恢复现场，异常处理返回
```

使用 LDM/STM 进行数据复制：

```
LDR R0,=SrcData             ;设置源数据地址，LDR此时作为伪指令加载地址要加 =
LDR R1,=DstData             ;设置目标地址
LDMIA R0,{R2-R9}            ;加载8字数据到寄存器R2 ～ R9
STMIA R1,{R2-R9}            ;存储寄存器R2-R9到目标地址上
```

使用 LDM/STM 进行现场保护，常用在子程序或异常处理中：

```
STMFD SP!,{R0-R7,LR}        ;寄存器入栈
.....
BL DELAY                    ;调用DELAY子程序
.....
LDMFD SP!,{R0-R7,PC}        ;恢复寄存器，并返回
```

（3）SWP 是寄存器和存储器交换指令，可使用 SWP 实现信号量操作。

```
12C_SEM EQU 0x40003000      ;EQU定义一个常量
12C_SEM_WAIT                ;标签
MOV R1,#0
LDR R0,=12C_SEM
SWP R1,R1,[R0]              ;取出信号量，并设置为0
CMP R1,#0                   ;判断是否有信号
BEQ 12C_SEM_WAIT            ;若没有信号，则等待
```

2．数据处理指令

ARM 数据处理指令包含数据传送指令、算术逻辑运算指令、比较指令、乘法指令。

（1）数据传送指令：MOV　MVN。

```
MOV  R1, R0                 ;将寄存器R0的值传送到寄存器R1
MOV  PC, R14                ;将寄存器R14的值传送到PC，常用于子程序返回
MOV  R1, R0, LSL #3         ;将寄存器R0的值左移3位后传送到R1
MOV  R0, #5                 ;将立即数5传送到寄存器R0
MVN  R0, #0                 ;将立即数0按位取反后传送到寄存器R0中，完成后R0 = -1
MVN  R1, R2                 ;将R2按位取反后，结果存到R1
```

（2）ADC 指令：带进位加法指令，将操作数 2 的数据与 Rn 的值相加，再加上 CPSR 中 C 条件标志位，结果保存到 Rd 中。使用 ADC 指令实现 64 位加法。

```
ADDS R0,R0,R2               ;R0+R2 => R0，影响CPSR中的值
ADC  R1,R1,R3               ;(R1、R0) = (R1、R0)+(R3、R2)
```

（3）SBC 指令：带借位减法指令，用寄存器 Rn 减去操作数 2，再减去 CPSR 中的 C 条件标志位的非（即若 C 标志清零，则结果减去 1），结果保存在 Rd 中。使用 SBC 实现 64 位减法。

```
SUBS R0,R0,R2
SBC  R1,R1,R3               ;使用SBC实现64位减法，(R1,R0) - (R3,R2)
```

（4）AND 指令：按位与操作。

```
ANDS R0,R0,#0x01            ;取出最低位数据
```

（5）ORR 指令：按位或操作。

```
ORR R0,R0,#0x0F             ;将R0的低4位置1
```

EOR 指令是进行异或操作，BIC 指令是位清除指令（遇 1 清 0）。

（6）比较指令：CMP、CMN、TST、TEQ。

```
CMP R1, #10          ;将寄存器R1的值与10相减，并设置CPSR标志位
ADDGT R0, R0, #5     ;如果R1>10，则执行ADDGT指令，将R0加5
CMN  R0, R1          ;R0 - (-R1)，反值比较，影响CPSR标志位
CMN  R0, #10         ;R0 - (-10)，反值比较，影响CPSR标志位
TST  R1, #3          ;位测试指令，检查R1中第0位和第1位是否为1，更新条件标志位
TEQ  R1, R2          ;相等测试指令，比较R0与R1是否相等，也可看做相减，相等则为0，Z=1
```

（7）MUL 指令：乘法指令。

```
MUL R1,R2,R3         ;R1=R2*R3。
MULS R0,R3,R7        ;R0=R3*R7，同时设置CPSR中的N位和Z位
```

（8）MLA 是乘加指令，将操作数 1 和操作数 2 相乘再加上第 3 个操作数，结果的低 32 位存入到 Rd 中。

UMULL 是 64 位无符号乘法指令。

```
UMULL R0,R1,R5,R8          ;(R1、R0) = R5 * R8
```

3．分支指令

（1）B 指令：跳转指令。

（2）BL 指令：带链接的跳转指令，指令将下一条指令复制到 R14（即 LR）链接寄存器中（方便跳转后的返回），然后跳转到指定地址运行。BL 指令用于子程序调用，例如：

```
BL  DELAY
```

（3）BX 指令：带状态切换的跳转指令。例如：

```
BX R0 ;跳转到R0指定的地址，并根据R0的最低位来切换处理器的状态
```

4．协处理器指令

（1）MCR：ARM 寄存器到协处理器寄存器的数据传送指令。

（2）MRC：协处理器寄存器到 ARM 寄存器的数据传送指令。

指令格式：

```
MRC/MCR {cond} coproc, opcode1, Rd, CRn, CRm{, opcode2}
```

coproc：指令操作的协处理器名，标准名为 pn，n 为 0～15。

opcode1：协处理器的特定操作码。

Rd：MRC 操作时，作为目标寄存器的协处理器寄存器，MCR 操作时，作为 ARM 处理器的寄存器。

CRn：存放第一个操作数的协处理器寄存器。

CRm：存放第二个操作数的协处理器寄存器。

opcode2：可选的协处理器特定操作码。

MRC/MCR 指令举例如下：

```
mcr/mrc p15,0,r0,c1,c0,0
```

5．异常产生及程序状态寄存器（PSR）处理指令

（1）中断指令 SWI：SWI 指令用于产生中断，从而实现用户模式变换到管理模式，CPSR 保存到管理模式的 SPSR 中，执行转移到 SWI 向量。

```
SWI 0x123456；软中断，中断立即数 0x123456
```

在 SWI 异常中断处理程序中，取出 SWI 立即数的步骤为：首先确定引起软中断的 SWI 指令是 ARM 指令还是 THUMB 指令，这可通过对 SPSR 访问得到，然后要取得该 SWI 指令的地

址，这可通过访问 LR 寄存器得到，接着读出指令，分解出立即数。程序代码如下：

```
T_bit EQU 0x20                ;0010 0000
SWI_Hander
STMFD SP!,{R0-R3,R12,LR}      ;现场保护
MRS R0,SPSR                   ;读取SPSR
STMFD SP!,{R0}                ;保存SPSR
TST R0, #T_bit                ;测试T标志位，0为ARM，1为THUMB
LDRNEH  R0,[LR,#-2]           ;若是THUMB指令，读出产生中断的指令码(16位)
BICNE R0,R0,#0xFF00           ;取得THUMB指令的8位立即数
LDREQ R0,[LR,#-4]             ;若是ARM指令，读取产生中断的指令码(32位)
BICEQ R0,R0,#0xFF000000       ;取得ARM指令的24位立即数
BL  C_SWI_Handler
LDMFD SP!,{R0-R3,R12,PC}^     ;SWI异常中断返回
```

（2）MRS 指令：读状态寄存器指令，在 ARM 处理器中，只有 MRS 指令可以从状态寄存器 CPSR 或 SPSR 读出到通用寄存器。

```
MRS R1,CPSR                   ;将CPSR状态寄存器读取，保存到R1
MRS R2,SPSR                   ;将SPSR状态寄存器读取，保存到R2
```

MRS 应用如下。

① 使能 IRQ 中断：

```
ENABLE_IRQ
MRS R0,CPSR
BIC R0,R0,#0x80               ;1000 0000
MSR CPSR,R0
MOV PC,LR
```

② 禁止 IRQ 中断：

```
DISABLE_IRQ
MRS R0,CPSR
ORR R0,R0,#0x80
MSR CPSR,R0
MOV PC,LR
```

（3）MSR：写状态寄存器指令，在 ARM 处理器中，只有 MSR 指令可以直接设置状态寄存器 CPSR 或 SPSR。

6．伪指令

ARM 伪指令不是 ARM 指令集中的指令，只是为了编程方便编译器定义了伪指令。

ARM 地址读取伪指令有四条，分别是 ADR、ADRL 、LDR、NOP 伪指令。

（1）ADR、ADRL 指令将基于 PC 相对偏移的地址读取到存储器中，例如：

```
ADR  R0 , DISP_TAB            ;加载转换表地址
LDR  R1, [R0,R2]              ;使用R2作为参数，进行查表
DISP_TAB
DCB  0xc0,0xf9,0xa4,0x99,0x92,0x82,0xf8,0x80
```

（2）LDR 伪指令用于加载 32 位的立即数或一个地址值到指定寄存器，前加“ = ”。

```
LDR R0,=0x123456              ;加载32位立即数0x123456
LDR R0,=DATA_BUF+60           ;加载DATA_BUF地址+60
```

（3）NOP 是空操作伪指令。

宏是一段独立的程序代码，它是通过伪指令定义的，在程序中使用宏指令即可调用宏，当程序被汇编时，汇编程序将对每个调用进行展开，用宏定义取代源程序中的宏指令。

（1）符号定义伪指令

全局变量声明：GBLA、GBLL 和 GBLS，其中最后一个字符 A 代表算术变量，初始化为 0；L 代表逻辑变量，初始化为 FALSE，S 代表字符串，初始化为空。

局部变量声明：LCLA、LCLL 和 LCLS，A、L、S 含义同上。

变量赋值：SETA、SETL、和 SETS。

应用举例如下：

```
    MACRO                          ;声明一个宏
    SENDDAT $dat                   ;宏的原型 $表示后面是变量
    LCLA bitno                     ;声明一个局部算术变量
    bitno SETA 8                   ;设置变量值为8
    MEND                           ;结束
```

① 为一个通用寄存器列表定义名称“ RLIST ”。RLIST 指令格式：

```
    name RLIST {reglist}
```

例如：

```
    LoReg RLIST {R0-R7}            ;定义寄存器列表LoReg
```

② 为一个协处理器的寄存器定义名称“ CN ”。指令格式：

```
    name CN expr
```

其中 name 是要定义的协处理器的寄存器名称，expr 对应协处理器的寄存器编号，数值范围为 0～15，

例如：

```
    MemSet CN 1                    ;将协处理器的寄存器1名称定义为 MemSet
```

③ 为一个协处理器定义名称“ CP ”。举例如下：

```
    DivRun CP 5                    ;将协处理器5名称定义为DivRun
```

（2）数据定义伪指令

① LTORG。LTORG 用于声明一个文字池，在使用 LDR 伪指令时，要有适当的地址加入。LTORG 声明文字池，这样就会把要加载的数据保存在文字池内，再用 ARM 的加载指令读出数据（若没有使用 LTORG 声明文字池，则汇编器会在程序末尾自动声明）。LTORG 伪指令应用举例如下：

```
    LDR R0,=0x12345678
    ADD R1,R1,R0
    MOV PC,LR
    LTORG                          ;声明文字池
    DCD 0x333
    DCD 0x555
```

② MAP 或 ^。MAP 用于定义一个结构化的内存表的首地址，^与 MAP 同义，例如：

```
    MAP  0x00,  R9   ;定义内存表的首地址为R9。
```

③ FIELD 或 #。FIELD 用于定义一个结构化内存表的数据域，例如：

```
    ^ _ISR_STARTADDRESS            ; ^ is synonym for MAP
    HandleReset  #  4              ;定义数据域 HandleReset，长度为4字节
```

④ SPACE 或 %。SPACE 用于分配一块内存单元，并用 0 初始化，例如：

```
    AREA  DataRAM, DATA, READWROTE     ;声明一数据段，名为DataRAM
```

```
DataBuf  SPACE  1000              ;分配1000字节空间
```

⑤ DCB。DCB 分配一段字节内存单元，并用指定的数据初始化，DCB 伪指令格式：

```
{label} DCB expr{,expr} ...
```

⑥ DCD 和 DCDU：分配一段字的内存单元，并用指令的数据初始化。

⑦ DCQ 和 DCQU：分配一段双字的内存单元，并用 64 位整数数据初始化。

⑧ DCW 和 DCWU：分配一段半字的内存单元，并用指定的数据初始化。

⑨ ASSERT 为断言错误伪指令，编译器对汇编程序的第二遍扫描中，若其中 ASSERT 条件不成立，ASSERT 伪指令将报告该错误信息，例如：

```
ASSERT  Top <>Temp    ;断言Top 不等于 Temp
ASSERT :DEF:ENDIAN_CHANGE
```

（3）汇编控制伪指令

① 条件汇编：IF、ELSE 和 ENDIF。

IF、ELSE 和 ENDIF 伪指令能够根据条件把一段代码包括在汇编程序内或将其排除在程序之外，[与 IF 同义，|与 ELSE 同义，]与 ENDIF 同义。应用举例如下：

```
[ {CONFIG} = 16     ; [ 代表 IF
BL __rt_udiv_1
|                   ; | 代表 ELSE
BL __rt_div0
]                   ; ] 代表 ENDIF
```

② MACRO 和 MEND。

MACRO 和 MEND 伪指令用于宏定义，MACRO 表示宏定义的开始，MEND 表示宏定义的结束，用 MACRO 和 MEND 定义的一段代码，称为宏定义体。应用举例如下：

```
MACRO
CSI_SETB            ;宏名为CSI_SETB，无参数
LDR R0,=rPDATG      ;读取GPG0 口的值
LDR R1,[R0]
ORR R1,R1,#0x01     ;CSI置位操作
STR R1,[R0]         ;输出控制
MEND
```

③ WHILE 和 WEND。

WHILE 和 WEND 伪指令用于根据条件重复汇编相同的或几乎相同的一段源程序，应用举例如下：

```
WHILE no< 5
no SETA no+1
WEND
```

（4）杂项伪指令

在汇编程序设计较为常用，如段定义伪指令、入口点设置伪指令、包含文件伪指令、标号导出或引入声明。

边界对齐：ALIGN。

段定义：AREA。

指令集定义：CODE16 和 CODE32。

汇编结束：END。

程序入口：ENTRY。

常量定义：EQU。

声明一个符号可以被其他文件引用：EXPORT 和 GLOBAL。

声明一个外部符号：IMPORT 和 EXTERN。

包含文件：GET 和 INCLUDE。

给特定的寄存器命名：RN。

对于 GNU 系统，伪指令前面统一加上“ . ”。

4.5 裸机程序编程步骤

裸机程序编程步骤可分为以下几步。

（1）查看电路原理图，知道硬件电路工作原理。

（2）找到硬件电路图中使用的 CPU 相应引脚，查看 CPU 手册，查阅对应引脚的相关控制寄存器的功能描述。

（3）编写启动程序 start.S。

（4）编写头文件.h 和源文件.c。

（5）编写 makefile 文件，执行 make 生成二进制可执行文件（bin 文件）。

（6）烧写程序到目标机并运行。

烧写程序到目标机的相关步骤如下。

（1）安装 TFTP 服务器，打开 tftp32.exe 工具。Windows 系统中 TFTP 是服务器，目标机中有 uboot 的 TFTP 客户端，工作时，TFTP 客户端发送请求，TFTP 服务器发送数据文件。这里在 Windows 系统中选用 TFTP（tftp32.exe）工具，实现裸机程序放在目标机中运行。打开 tftp32.exe 工具，选择下载文件所在的文件夹，并把裸机程序编译生成二进制可执行文件（bin 文件）放在该文件夹中。

（2）网络配置（同一网段）。设置 Windows IP，如“ 192.168.0.103 ;255.255.255.0 ;192.168.0.1 ”。

（3）在目标机中（通过串口工具如超级终端）操作：

目标机启动 3 秒内按 Enter 键，进入 uboot 菜单，选择菜单[e]，进入命令行，如图 4.4 所示。注意，系统进入命令行模式后，不能使用 Linux 命令。

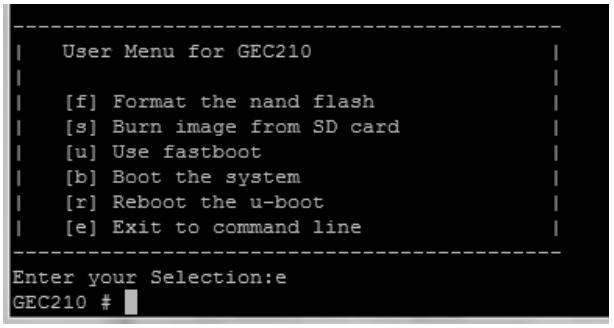

图 4.4　命令行模式

系统进入命令行后，设置服务器（Windows 系统）IP 地址、本地（目标机）IP 地址和网关 IP 地址。如下面命令：

```
# setenv  serverip 192.168.1.100      //设置服务器IP，与Windows系统一致
#setenv ipaddr 192.168.1.101         //设置目标机IP
#setenv gatewayip 192.168.1.1        //设置网关IP
#saveenv                             //保存环境设置
```

通过uboot的TFTP下载功能下载.bin文件到内存的0x30000000，然后使用“go 0x30000000”命令就可以运行该裸机程序。如下面命令：

```
#tftp 0x30000000  ***.bin            //TFTP传输
#go 0x30000000                       //执行运行命令
```

4.6 编程实现点亮 LED

任务目的：初步掌握使用汇编语言设置 S5PV210 特殊功能寄存器的方法。

所需知识与技能：ARM 汇编语言、指令系统、S5PV210 特殊功能寄存器使用（GPIO）方法、Linux 基本操作、嵌入式系统开发板（实验箱）。

所需设备：PC、嵌入式系统开发板（实验箱）。

实操方法：要求学生先按给定的程序完成操作，然后要求学生现场修改程序，实现 LED 灯闪烁时间间隔的变化。

1．电路原理图

目标机上提供了 4 个可编程用户 LED，原理图如图 4.5 所示。

注意：不同的实验箱上的 LED 原理图会有不同，主要是 LED 的接口不同，读者应查看自身的开发板或实验箱平台的 LED 原理图，查看 LED 与 I/O 口的连接关系，对下面的代码进行修改。

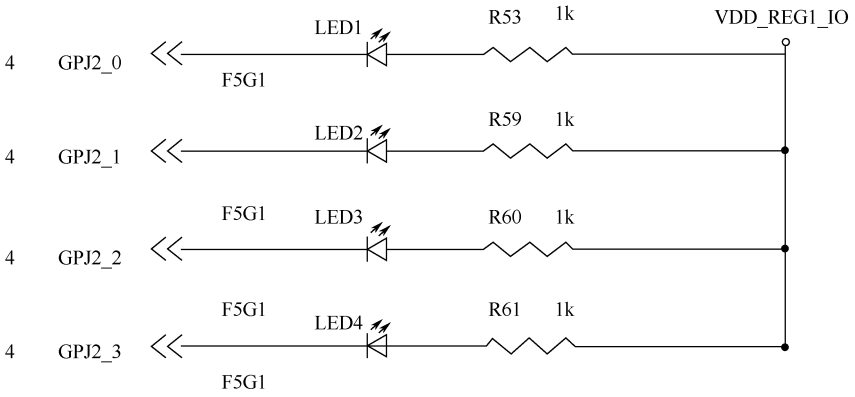

图 4.5　LED 控制电路原理图

可见，LED1、LED2、LED3、LED4 分别使用的 CPU 端口资源为 GPJ2_0、GPJ2_1、GPJ2_2、GPJ2_3。通过对 CPU 的 GPIO 编程，I/O 输出低电平，则 LED 灯亮；输出高电平，则 LED 灯灭。

2．程序相关讲解

（1）汇编代码 start.S

由图4.5可知，点亮 GEC210 的 4 个 LED 需如下两个步骤。

第一步：设置控制寄存器 GPJ2CON，使 GPJ2_0/1/2/3 四个引脚为输出功能；

第二步：往数据寄存器 GPJ2DAT 写 0，使 GPJ2_0/1/2/3 四个引脚输出低电平，4 个 LED 会亮；相反，往数据寄存器 GPJ2DAT 写 1，使 GPJ2_0/1/2/3 四个引脚输出高电平，4 个 LED 会灭。

以上两个步骤即为 start.S 中的核心内容，start.S 里面涉及的汇编指令请查看 4.4 节的内容。GPJ2CON、GPJ2DAT、GPJ2PUD 功能描述如表 4.6～表 4.8 所示。

表 4.6　GPJ2 端口配置寄存器

GPJ2CON	数据位	功能描述	初始状态
GPJ2CON[7]	[31:28]	0000=输入 0001=输出 0010= MSM_DATA[7] 0011= KP_COL[8] 0100=CF_DATA[7] 0101=MHL_D14 0110～1110=保留 1111=GPJ2_INT[7]	0000
GPJ2CON[6]	[27:24]	0000=输入 0001=输出 0010= MSM_DATA[6] 0011= KP_COL[7] 0100=CF_DATA[6] 0101=MHL_D13 0110～1110=保留 1111=GPJ2_INT[6]	0000
GPJ2CON[5]	[23:20]	0000=输入 0001=输出 0010= MSM_DATA[5] 0011= KP_COL[6] 0100=CF_DATA[5] 0101=MHL_D12 0110～1110=保留 1111=GPJ2_INT[5]	0000
GPJ2CON[4]	[19:16]	0000=输入 0001=输出 0010= MSM_DATA[4] 0011= KP_COL[5] 0100=CF_DATA[4] 0101=MHL_D11 0110～1110=保留 1111=GPJ2_INT[4]	0000
GPJ2CON[3]	[15:12]	0000=输入 0001=输出 0010= MSM_DATA[3] 0011= KP_COL[4] 0100=CF_DATA[3] 0101=MHL_D10 0110～1110=保留 1111=GPJ2_INT[3]	0000