第 1 章　嵌入式系统基础（共 59 分）

一、什么是嵌入式系统？（5 分）

通常的定义：嵌入式系统是以应用为中心，以计算机技术为基础，采用可剪裁软硬件，适用于对功能、可靠性、成本、体积、功耗等有严格要求的专用计算机系统。（5 分）

通俗的定义：嵌入到对象体系中的专用计算机系统。（通常的定义和通俗的定义任意写出一个就可以得 5 分。）

嵌入性、专用性与计算机系统是嵌入式系统的三个基本要素。

二、嵌入式微处理器的体系结构有哪两种？简单两种体系结构各自的特点。（8 分）

嵌入式微处理器的体系结构有冯.诺依曼结构和哈佛结构两种。（2 分）

冯.诺依曼结构也称为普林斯顿结构，是一种将程序指令存储器和数据存储器合并在一起的存储器结构。单次取指令长度和取数据的长度相同。（3 分）

哈佛结构：是一种将 程序指令存储和数据存储分开的存储器结构。程序指令存储和数据存储分开，单次取指令长度和取数据的长度可以不相同。（3 分）

三、简单描述嵌入式实时操作系统中的任务的四个状态。（8 分）

实时操作系统中的任务有四个状态：运行、就绪、挂起、冬眠。

运行：获得 CPU 的控制权。（2 分，只给出状态名称只得 1 分）

就绪：进入任务等待队列，等待通过调度转为运行状态。（2 分，只给出状态名称只得 1 分）

挂起：任务发生阻塞，移出任务就绪队列，等待系统实时事件的发生而唤醒，从而转为就绪 或 运行。（2 分，只给出状态名称只得 1 分）

冬眠：任务完成 或 错误 等原因被清除的任务，也可以认为是系统中不存在的任务。（2 分，只给出状态名称只得 1 分）

四、简述精简指令集 RISC 和复杂指令集 CISC 的主要区别（8 分）

RISC：一个周期执行一条指令，通过简单指令的组合实现复杂操作，指令长度固定。CISC: 指令长度不固定，执行需要多个周期。

RISC: 流水线每周期前进一步 CISC: 指令的执行需要调用一段微程序

RISC: 更多通用寄存器 CISC: 用于特定目的的专用寄存器

RISC: 独立 Load/Store 指令完成数据在寄存器和外部存储器之间的传输 CISC: 处理器能够直接处理存储器中的数据

五、通过网上查找资料，列出一些嵌入式操作系统，并对其做简单介绍(提供 7 种以上的嵌入式操作系统，至少其中 3 种是 PPT 中未提到过的嵌入式操作系统)。（14 分）

每种操作系统给出得 2 分，7 种共 14 分。

六、多选题 （10 分）

1、以下关于 RISC 和 CISC 说法正确的是(A、C)。（3 分）
A、RISC 一个周期执行一条指令。
B、RISC 相对于 CISC，其指令集实现的功能更多。
C、RISC 相对于 CISC 具有更多的寄存器
D、RISC 能直接处理存储器中的数据
2、嵌入式系统的三个基本要素是（ A、B、C ）（2 分）
A：嵌入性 B：专用性
C：计算机系统 D：通用性
3、在以下分类中，属于按实时性划分的是(B、D)。（2 分）
A、8 位嵌入式系统。
B、硬实时系统。
C、前后台系统
D、软实时系统
4、属于硬件调试工具是（ A、C、D ）。（3 分）
A、实时在线仿真器
B、电路开发板
C、逻辑分析仪
D、ROM 仿真器

七、单选：（6 分）

1、在下列嵌入式处理器类型中，集成度最高的是(D)。（3 分）
A、嵌入式微处理器
B、嵌入式微控制器
C、嵌入式 DSP 处理器
D、SOC 片上系统
2、实时操作系统中的任务的就绪状态是指(B)。（3 分）
A、获得 CPU 的控制权。
B、进入任务等待队列，等待通过调度转为运行状态。
C、任务发生阻塞，移出任务就绪队列。
D、任务完成或错误 等原因被清除的任务。

第 2 章　ARM 处理器及系统结构（共 96 分）

一、ARM 处理器有几种工作状态？分别是什么？（3 分）

ARM 处理器有两种工作状态（1 分）：ARM 状态和 Thumb 状态。（两种状态各 1 分）

二、ARM 处理器有几种运行模式？分别是什么？（8 分）

ARM 处理器有 7 种运行模式（1 分）：中断、快中断、复位中断、软中断异常、预取指令中止异常、数据中止异常 和 未定义指令异常（7 种模式各 1 分）

三、简单描述 ARM 体系结构与纯粹的 RISC 体系结构的不同点。（10 分）

ARM 内核不是一个纯粹的 RISC 体系结构。ARM 指令集与纯粹的 RISC 的定义有以下几个不同。

（1）一些特定指令的周期数可变，并不是每条 ARM 指令都是单周期的。（2 分）

（2）内嵌的桶形移位器产生了更为复杂的指令，扩展了指令的功能，因此改善了内核的性能。（2 分）

（3）支持 16 位的 Thumb 指令集，提高了代码密度。（2 分）

（4）支持条件执行：每条指令都可以设置一个执行条件，只有条件满足时才执行。（2 分）

（5）增强指令：一些功能强大的数字信号处理指令被加入到 ARM 指令集中。（2 分）

四、简单描述 ARM9 的 5 级流水线（10 分）

取指：指令从存储器中取出，放入指令流水线；（2 分）

译码：指令译码；（2 分）

执行：把一个操作数移位，产生 ALU 的结果。如果指令是 Load 或 Store，在 ALU 中计算存储器的地址；（2 分）

缓存/数据：如果需要，则访问数据存储器；否则，ALU 的结果只是简单地缓冲一个时钟周期，以便使所有指令具有同样的流水线流程；（2 分）

回写：将指令产生的结果写回到寄存器堆，包括任何从寄存器读出的数据。（2 分）

五、简单描述 ARM 处理器进入异常所采取的操作。（10 分）

(1) 在适当的 LR 中保存断点的地址。（2 分）

(2) 把当前程序状态寄存器（CPSR）中的内容保存到模式私有寄存器 SPSR 中；（2 分）

(3) 将寄存器 CPSR 中的 MODE 域设置为中断（异常）应进入的运行模式；（2 分）

(4) 对 CPSR 的 I 位和 F 位进行相应的设置，以防止再次响应同一个中断请求。（2 分）

(5) 强制 PC 从相关的异常向量处取指，即到中断向量表中获取中断向量，转向用户所编写的中断（异常）服务程序。（2 分）

六、ARM 的快中断采用了几种措施来保证更快的响应速度？这些措施是什么？（6 分）

为减少延时，ARM 在快中断中采取了 两个措施：（1 分）

（1）专门为快中断配置了较多的私有寄存器，从而可使中断服务程序有足够的寄存器来使用，而不必与被中断服务程序使用同一组寄存器，这样就免去了因寄存器冲突而必需的保护及恢复现场工作。（3 分）

（2）ARM 把 FIQ 的中断向量放在了中断（异常）向量表末尾 0X0000001C 处，因此它后面没有其它中断向量，允许用户将中断服务程序直接放在这里。（2 分）

七、多选题（7 分）

1、为减少延时，提高中断处理速度，ARM 在快中断中采取了哪些措施（ B、D ）（2 分）

A. 配置了高速缓存。

B. 配置了较多的私有寄存器。

C. 更高的中断优先级。

D. 把 FIQ 的中断向量放在了中断（异常）向量表末尾 处。

2、下列说法正确的是（ A、B、C、D ）。（3 分）

A. 异常实质上也是一种中断，只不过它主要负责处理处理器内部事件

B. 中断控制器用于中断源和处理器之间，主要用于对处理器可以接收中断源的数目进行扩充及对中断进行必要的管理。

C. 处理器在现行指令执行结束后，才能响应中断。

D. 在处理器收到中断请求之后，它们都需要获得中断服务程序首地址——中断向量。

3、下列关于 ARM 的说法正确的是（ B、C、D ）：（2 分）

A. ARM 公司生产的处理器

B. ARM 是一个公司的名称

C. ARM 是一类微处理器的通称

D. ARM 是一种技术的名称

八、判断题 （20 分，每小题 2 分）

1. ARM9TDMI 核其工作模式有两种：ARM 模式和 Thumb 模式。（ X ）
2. 当异常发生，进入异常运行模式的时候，处理器会自动禁用中断和快中断，以确保异常处理程序安全运行。（ X ）
3. 当处理器处于 Thumb 状态时发生了异常，在异常向量地址装入 PC 时，会自动切换到 ARM 状态。（ √ ）
4. S3C2440 芯片可以通过软件来指定存储器格式，缺省为小端格式。（ √ ）
5. 在减法指令中，当运算中发生了借位，则 C 标志位=1。（ X ）

6. ARM 处理器在用户模式下，可以通过修改 CPSR 进入系统模式。（ X ）
7. 对于计算机系统来说，一个字的长度是 32 位的，半字的长度是 16 位的。（ X ）
8. ARM920T 处理器中，对是通过协处理器 P15 来实现对 MMU 的控制的。（ √ ）
9. ARM9 的 5 级流水线设计，相对于 ARM7 的 3 级流水线设计，减少了在每个时钟内必须完成的最大工作量，进而允许使用较高的时钟频率。（ √ ）
10. ARM 处理器采用的是 RISC 体系结构，因此所有的指令都在一个周期内执行完成。（ X ）

九、单选题（22 分，每小题 2 分）

1. 如果处理器采用大端存储器格式，假如一个字存放在地址为 A、A+1、A+2、A+3 四个连续的存储单元中，则该字的地址为（ A ）
   A．A　　B．A+1　　C．A+2　　D．A+3
2. ARM7EJ 名称中的 J 后缀，代表（ C ）
   A、支持增强型 DSP 指令　　B、支持 Thumb 指令集
   C、支持 Jazelle　　D、支持 Embedded ICE
3. 在通用的 CPU 上提供 DSP 能力，是从 ARM 指令集构体系结构版本（ B ）以后开始的。
   A、v4　　B、v5　　C、v6　　D、v7
4. ARM 推出的 Cortex 系列包括三个系列，其中实时操作系统而设计的是（ C ）
   A、Cortex-A　　B、Cortex-B　　C、Cortex-R　　D、Cortex-M
5. 在 ARM9 的 5 级流水线设计中，从寄存器中读取操作数，是在哪个阶段完成的？（ B ）
   A、取指　　B、译码　　C、执行　　D、缓存/数据　　E、回写
6. 在 ARM9 的 5 级流水线设计中，计算访存指令访问存储器的地址的操作，是在哪个阶段完成的？（ C ）
   A、取指　　B、译码　　C、执行　　D、缓存/数据　　E、回写
7. ARM 处理器的 7 种运行模式中，处理器复位之后进入（ C ）。
   A．用户模式　　B．系统模式　　C．管理模式　　D．终止模式
8. 下列 ARM 处理器的运行模式中，不属于异常模式的是（ B ）。
   A．未定义模式　　B．系统模式　　C．管理模式　　D．终止模式。
9. R13 通常用于存储（ D ）。
   A．程序计数器　　B．中断返回地址　　C．子程序返回地址　　D．堆栈指针
10. 程序计数器的值保存在（ D ）中。
    A．R12　　B．R13　　C．R14　　D．R15
11. 读程序计数器时，指令读出的 R15 的值是当前指令地址加上（ D ）字节。
    A．1　　B．2　　C．4　　D．8

第 3 章 ARM 指令集（共 50 分）

一、ARM 指令分为几类？分别是什么？（6 分）

ARM 指令有五类（1 分）：分支指令、数据处理指令、存储访问指令、协处理器指令和杂项指令五类。（5 类指令各 1 分）

二、ARM 指令有几种寻址方式？分别是什么？（9 分）

ARM 指令有八种寻址方式（1 分）：立即数寻址、寄存器寻址、寄存器移位寻址、寄存器间接寻址、基址变址寻址、多寄存器寻址、堆栈寻址、相对寻址。（8 类寻址各 1 分）

三、单选（35 分，每小题 5 分）

1. 下列指令合法的是（ B ）
   A．MOV R0,#0x132　　B．MOV R0,#0x264
   C．MOV R0,#0x266　　D．MOV R0,#0x4C8
2. 汇编指令 MOV R0,#0x4800, 经编译后，所得到的机器码的低 12 位是（ A ）
   A．0xB12　　B．0xB24　　C．0xA09　　D．0xC48
3. 假设 R1 寄存器中的值为 ADR，则在执行了指令：STMIA R1!,{R3-R5,R8} 指令后，R1 的内容为（ A ）。
   A、R1=ADR+16　　B、R1=ADR+12
   C、R1=ADR-16　　D、R1=ADR-12
4. 假设 R1 寄存器中的值为 ADR，则在执行了指令：STMDB R1!, {R3-R5,R8} 指令后，R5 寄存器中的内容将存放在地址为（ B ）的存储单元中：
   A、ADR-4　　B、ADR-8　　C、ADR-12　　D、ADR-16
5. 假设 SP 寄存器中的值为 ADR，则在执行了指令：STMED SP!,{R3-R5,LR} 指令后，SP 的内容为（ C ）。
   A、ADR+16　　B、ADR+12　　C、ADR-16　　D、ADR-12
6. 假设 SP 寄存器中的值为 ADR，则在执行了指令：STMED SP!,{R3-R5,LR} 指令后，LR 寄存器中的内容将存放在地址为（ A ）的存储单元中。
   A、ADR　　B、ADR-4　　C、ADR-12　　D、ADR-16
7. 假设 SP 寄存器中的值为 ADR，则在执行了指令：LDMEA SP!, {R3-R5,PC} 指令后，地址为（ B ）的存储单元的值，将出栈到 PC 寄存器中。
   A、ADR　　B、ADR-4　　C、ADR-12　　D、ADR-16

第 4 章 ARM 伪指令及编程基础（共 125 分）

一、简述指令与伪指令的本质区别是什么？ （4 分）

指令与伪指令的本质区别是：指令经编译后，会生成对应的机器码（2 分）。而伪指令经编译后没有指令代码。（2 分）

二、简述伪指令的作用是什么？（8 分）

（1）程序定位的作用；（2 分）
（2）为非指令代码进行定义；（2 分）
（3）为程序完整性做标注；（2 分）
（4）有条件的引导程序段。（2 分）

三、简述宏指令与伪指令的主要区别是什么？（4 分）

宏指令与伪指令的主要区别是：在汇编时，这些宏指令被替换成一条或两条真正的 ARM 或 Thumb 指令。而伪指令经编译后没有指令代码。

四、简述伪指令 LTORG 的主要作用和目的。（6 分）

伪指令 LTORG 用来说明某个存储区域为一个用来暂存数据的数据缓冲区，也叫文字池或数据缓冲池。大的代码段也可以使用多个数据缓冲池。（3 分）
其目的是，防止在程序中使用 LDR 之类的指令访问时，可能产生的越界。（3 分）

五、定义一个内存表，其首地址为固定地址 8192，该内存表中包含 5 个数据域：consta 长度为 4 字节，constb 长度为 4 字节，x 长度为 8 字节，y 长度为 8 字节，string 长度为 16 字节（12 分）

```
MAP       8192          （2 分）
consta    FIELD    4（2 分）
constb    FIELD    4（2 分）
x         FIELD    8（2 分）
y         FIELD    8（2 分）
string    FIELD    16（2 分）
```

六、在列表中查找指定的数据（23 分）

要求如下：
(1) 定义一个存储单元首地址名为 Start 的数据列表，该数据列表包含 5 个字存储单元。第一个单元是列表中数据的数量，其值为 4，即列表中包含 4 个数据。后面四个单元存储

的是列表中的 4 个数据：0x0138A, 0x0A21DC, 0x1F5376, 0x9018613。

(2) 定义一个名为 NewItem 的字存储单元，该单元中包含要查找的数据，该例要求是 0x1F5376

(3) 要求编写一个程序，在 Start 数据列表中查找是否包含 NewItem 单元中的数据。如果包含则把该数据在列表中的位置序号（1、2、3、4）存储到 Index 字存储单元中。否则在 Index 字存储单元中存储 0xFFFFFFFF。

(4) 要求 Start 的数据列表和 NewItem 的字存储单元定义在代码段中（只读的 ROM 区），而 Index 字存储单元要定义在一个名为 mydata 的数据段中（可读写的 RAM 区）。

程序示例一：

```
;第四章作业，在列表中查找指定的数
    AREA    myprog, CODE, READONLY        （2 分）
    ;myprog 有编程人员自己定义，其余的关键词保持不变
    ;关键词的大写和小写都可以
    ENTRY                       ;程序入口（1 分）
    ;以下实现代码，不同同学有不同实现，大致没错就可以，总分 10 分
Main
    LDR     R0,=Start           ;R0=要查找的数据地址
    LDR     R1,NewItem          ;
    LDR     R3,Start            ;
    CMP     R3,#0               ;
    BEQ     Missing             ;
    LDR     R4,[R0,#4]!         ;取第 1 个操作数
    MOV     R2,#1
Loop
    CMP     R1,R4               ;
    BEQ     Done                ;
    SUBS    R3,R3,#1            ;
    LDR     R4,[R0,#4]!         ;
    ADD     R2,#1
    BNE     Loop                ;
Missing
    MOV     R2,#0xFFFFFFFF      ;
Done
    LDR     R5,=Index           ;
    STR     R2,[R5]             ;
    SWI     0x11                ;
Start
    DCD     0x4, 0x0138A, 0x0A21DC, 0x1F5376, 0x9018613    （3 分）
    ;0x0138A 可以定义为 0x00138A、0x0000138A 等不同形式,0x 可以由&代替
    ;还可以每个单元单独定义形式如下：
    ;DCD    0x4
    ;DCD    0x0138A
    ;DCD    0x000A21DC
```

```
    ;DCD    0x1F5376
    ;DCD    0x9018613

NewItem
    DCD     0x1F5376    （1 分）

    AREA    mydata, NOINIT   （3 分）
    MAP     0x40000000   （1 分）
Index
    FIELD 4   （1 分）

    END    （1 分）
```

程序示例二：

```
    AREA    myprog, CODE, READONLY   （2 分）
    ;myprog 由编程人员自己定义，其余的关键词保持不变
    ;程序中所有关键词的大写和小写都可以
    ENTRY   （1 分）
;以下实现代码，不同同学有不同实现，大致没错就可以，总分 10 分
Main
    LDR     R0,=NewItem
    SUB     R0,R0,#4
    LDR     R1,NewItem
    LDR     R3,Start
    CMP     R3,#0
    BEQ     Missing
    LDR     R4,[R0],#-4
Loop
    CMP     R1,R4
    BEQ     Done
    SUBS    R3,R3,#1
    LDR     R4,[R0],#-4
    BNE     Loop
Missing
    MOV     R3,#0xFFFFFFFF
Done
    LDR     R5,=Index
    STR     R3,[R5]
    SWI     0x11
Start    （3 分）
    DCD     0x4
    DCD     0x0000138A
    DCD     0x000A21DC
    DCD     0x1F5376
```

```
        DCD     0x9018613
        ;Start 定义的位置可以在程序的其它位置，不必一定在这里
    NewItem
        DCD     0x1F5376    （1 分）

        AREA    mydata, NOINIT   （3 分）
        MAP     0x40000000      （1 分）
    Index
        FIELD 4     （1 分）

        END    （1 分）
```

七、降序冒泡排序程序（33 分）

(1) 编写一个降序冒泡排序程序，要求如下：（22 分）

a) 定义一个存储单元首地址名为 StartOR 的数据列表（在代码段只读 ROM 区），该数据列表包含若干个字节存储单元。第一个单元是列表中数据的数量，后续若干连续存储单元包含若干字节数据。字节数据的数量由编程人员自己确定。

b) 编程要求把 StartOR 数据列表中存放的数据按降序排序（冒泡排序法）。排序后的数据存放到名为 Start 的数据列表中，该数据列表定义在名为 mydata 的数据段中（可读写的 RAM 区）。Start 数据列表的存储结构与 StartOR 数据列表完全相同。

(2) 要求画出程序流程图。流程图要求黑框白底，分支需要标注转移满足的条件。（5 分）大致正确就可以给满分。

(3) 回答问题：为什么 StartOR 数据列表要存放在只读 ROM 区中？而 Start 数据列表需要存放在 RAM 区中？（6 分）

答：只有存放在 ROM 去的程序和数据，在掉电后才能保持其原值，在上电后 ARM 处理器才能正确读取存放存放于其中的数据。因此 StartOR 数据列表保存了数据的初始值，需要在 ROM 中存放。而排序后的结果需要存放于 Start 数据列表中，其对应的存储单元必须可写，因此 Start 数据列表需要存放在 RAM 区中。（大致意思正确就可以用给满分）

```
;第四章作业：降序冒泡排序程序示例程序
    AREA    myprog, CODE, READONLY      （2 分）
    ;myprog 有编程人员自己定义，其余的关键词保持不变
    ;关键词的大写和小写都可以
    ENTRY                       ;程序入口（1 分）
;以下实现代码，不同同学有不同实现，大致没错就可以，总分 12 分
    LDR     R0,=StartOR     ;获取 StartOR 的首地址
    LDR     R2,=Start           ;获取 Start 的地址
    MOV     R6,R2
    LDRB    R1,[R0]     ;获取列表中数据的数量
    STRB    R1,[R2]
    CMP     R1,#0
    BEQ     Done
CopyLoop
```

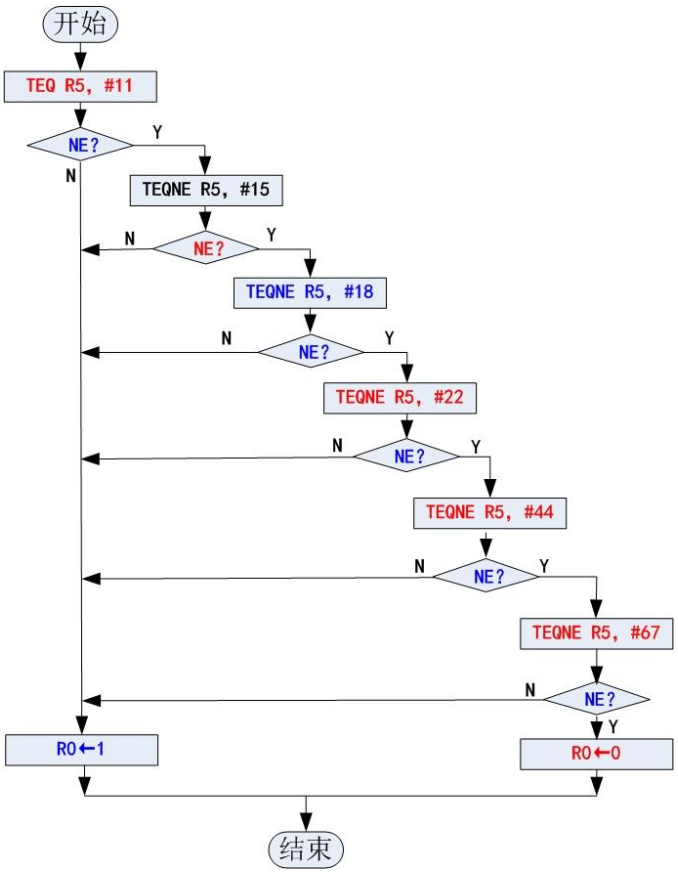
```
    LDRB    R3,[R0,#1]!
    STRB    R3,[R2,#1]!
    SUBS    R1,#1
    BNE     CopyLoop
Order
    MOV     R0,#0
    LDRB    R0,[R6]
    MOV     R8,R6
Sort
    ADD     R7,R6,R0
    MOV     R1,#0
    ADD     R8,R8,#1
Next
    LDRB    R2,[R7],#-1
    LDRB    R3,[R7]
    CMP     R2,R3
    BCC     NoSwitch
    STRB    R2,[R7],#1
    STRB    R3,[R7]
    ADD     R1,R1,#1
    SUB     R7,R7,#1
NoSwitch
    CMP     R7,R8
    BHI     Next
    CMP     R1,R0
    BNE     Sort
Done
    B       .           ;

StartOR DCB     8
        DCB     0x2A, 0x5B, 0x60, 0x3F, 0xD1, 0x19, 0x01, 0x78   （2 分）
    ;以上 2 行 StartOR 的第 1 个单元的值 8，需要和后面定义的数据的数量相对应。
    ;后面的数据的值由编程人员自己定义，不必和这里的示例完全相同
    ;如果后面有 6 个数据，则第 1 个单元的值为 6。
    ;上述 2 行可以合并为 1 行定义：
    ; DCB   8, 0x2A, 0x5B, 0x60, 0x3F, 0xD1, 0x19, 0x01, 0x78
    AREA    mydata, DATA   （2 分）
Start   DCB     0, 0, 0, 0, 0, 0, 0, 0, 0   （2 分）
    ;这里定义的单元数量必须和前面 StartOR 的一致，比如这里定义了 9 个 0，即有 9
个单元，和前面 StartOR 的一致。
    END   （1 分）
```

八、编写一个程序段，判断寄存器 R5 中的数据是否为 11、15、18、22、44、67，如果是，则将 R0 中的数据置为 1；否则将 R0 设置为 0，并把这个程序段定义为一个代码段，要求先画出程序流程图（流程图要求黑框白底，分支需要标注转移满足的条件。），再用 ARM 指令写详细代码，程序要完整。（17 分）

解：

流程图（7 分，大致正确即可）



程序如下：大致正确即可，程序结构正确 4 分，执行正确 6 分（10 分）

```
        AREA  TT, CODE, READONLY
        ENTRY
        TEQ     R5, #11
        TEQNE   R5, #15
        TEQNE   R5, #18
        TEQNE   R5, #22
        TEQNE   R5, #44
        TEQNE   R5, #67
        MOVEQ   R0, #1
        MOVNE   R0, #0
        END
```

九、试把如下 C 函数改写成 ARM 指令函数。（9 分）

```
int subxx(int x, int y)
{
    return  x-y;
}
```

解：

```
        AREA    TT,   CODE,   READONLY  （1 分）
        EXPORT     subxx     （2 分）
subxx     （1 分）
        SUBS R0, R0, R1     （2 分）
        MOV  PC,LR     （2 分）
        END    （1 分）
```

十、把下面的 ARM 指令函数改写成 C 语言函数。（9 分）

```
        AREA tt,CODE, READONLY
        EXPORT strcopy
strcopy
        LDRB R2,[R1],#1
        STRB R2,[R0],#1
        CMP R2,#0
        BNE strcopy
        MOV PC, LR
        END
```

解：

```
void strcopy( char *d, char *s)
  {
    while (*s !='\0')
     {
       *d= *s;
       d=d+1;
       s=s+1;
     }
     *d= *s;
  }
```

第 5 章 S3C2440 嵌入式系统（共 63 分）

一、简述启动代码存储在 NAND Flash 存储器上时，S3C2440 的启动过程。（6 分）

为了支持 NAND Flash 的 boot loader，S3C2440A 配备了一个内部的 SRAM 缓冲器名为 “Steppingstone”（垫脚石）。（2 分）

启动时，NAND Flash 上的前 4KByte 字节将被装载到 Steppingstone 中，并且装载到 Steppingstone 上的启动代码会被执行。（2 分）

一般情况下，启动代码会拷贝 NAND Flash 上的内容到 SDRAM 中，在引导代码执行完毕后就跳转到 SDRAM 执行。（2 分）

二、功率管理模块能够使系统工作在哪四种模式下？并分别对每种模式进行简单描述。（12 分）

（1）正常模式：功率管理模块向 CPU 和所有外部设备提供时钟。这种模式下，系统功率将达到最大。（3 分）

（2）低速模式：低速模式直接使用外部时钟（XTIpll 或者 EXTCLK)作为 FCLK，没有使用 PLL 产生的时钟。这种模式下，功率仅由外部时钟决定。（3 分）

（3）空闲模式：仅关掉 FCLK，停止为 CPU 提供时钟信号，而继续提供时钟给其他外设。（3 分）

（4）掉电模式：功率管理模块断开内部电源，因此 CPU 和除唤醒逻辑单元以外的外设都不会产生功耗。要执行掉电模式需要有两个独立的电源，其中一个给唤醒逻辑单元供电，另一个给包括 CPU 在内的其他模块供电。在掉电模式下，第二个电源将被关掉。（3 分）

三、DMA 的主要优点是什么？简单描述 S3C2440 的 DMA 不同源和目的设备的四种情况。（11 分）

DMA 的主要优点是：可以不通过 CPU 的中断来实现数据的传输，DMA 的运行可以通过软件或者通过外围设备的中断和请求来初始化。（3 分）

(1)源设备和目标都在系统总线 AHB 上；（2 分）

(2)源设备和目标都在外围总线 APB 上；（2 分）

(3)源设备在系统总线，而目标设备位于外围总线；（2 分）

(4)源设备在外围总线，而目标设备位于系统总线。（2 分）

四、简单描述 DMA 传输时，需求模式和握手模式的区别。（8 分）

主要区别如下：

在一次传输结束时，DMA 检查 **DMA 请求信号**的状态。（2 分）

在需求模式下：如果 DMA 请求信号仍然有效，则传输马上再次开始，否则等待；（3 分）

在握手模式下：如果一次传送结束，DMA 的请求信号还是有效的，那 DMA 控制器，不会进行下一次传送，而是要等待，直到 DMA 请求信号变得无效后，下一次请求到来时，才能进行新的一次传输。每请求一次传输一次。（3 分）

五、简单描述 S3C2440 的 PWM 模块的自动加载模式和双缓冲模式的主要作用。（6 分）

自动加载模式：自动加载模式**使能**时，当 TCNTn 的值到 0 时，自动加载操作**复制 TCNTBn 的值到 TCNTn** 中。但是如果自动加载模式没有使能， TCNTn 将不进行任何操作。（3 分）

双缓冲模式：脉宽调制定时器有一个双缓冲功能 ，在这种情况下，改变**下次加载值**的同时不影响当前**定时周期**。因此，尽管设置一个新的定时器值，当前定时器的操作将会继续完成而不受影响。（3 分）

六、假如 CPU 响应 TIMER1 中断进入中断服务程序，为了避免该中断请求信号再次引起中断响应，需要把 SRCPND 和 INTPND 寄存器中相关标志位清除（12 分）

(1) 请写出相关的汇编语言程序代码。（注：只需写出相关指令代码，不需要写出完整的汇编程序结构）（8 分）

(2) 请写出相关的 C 语言程序代码。假设寄存器名称与寄存器的地址已经相关联（在头文件或其他文件中定义），对 S3C2440 中相关接口的控制寄存器的访问，可以直接使用该寄存器的名称。 （注：只需写出相关代码，不需要写出完整的程序结构）（4 分）

解：

汇编程序代码如下：（每行代码 2 分，酌情扣分）

```
LDR R1, =0x4A000000
MOV R2, #0x800
STR R2,[R1]
STR R2,[R1, 0x10]
```

或者

```
LDR R1, =0x4A000000
MOV R2, #0x800
STR R2,[R1]
LDR R1, =0x4A000010
STR R2,[R1]
```

或者

```
LDR R0, =0x4A000000
LDR R1, =0x4A000010
LDR R2, [R1]
STR R2, [R0]
STR R2, [R1]
```

C 代码：（4 分）

```
SRCPND = (1 << 11);
INTPND = INTPND;
```

七、如果程序要屏蔽 INT_TICK 中断，请写出相关汇编语言程序代码（注：其余的屏蔽标志位要保持其原值不变。只需写出相关指令代码，不需要写出完整的汇编程序结构）（8 分）

解：（每行代码 2 分，酌情扣分）

```
LDR R3,=0x4A000008
LDR R4,[R3]
ORR R4, R4, #0x100
STR R2,[R1]
```