1.1 什么是嵌入式系统？

答：嵌入式系统是以应用为中心，以计算机技术为基础，软/硬件可裁减，功能。可靠性，成本，体积，功耗要求严格的专用计算机系统。

1.2 与通用计算机相比，嵌入式系统有哪些特点？

答：（1）嵌入式系统通常是面向特定应用的；（2）嵌入式系统是将计算机技术，半导体技术和电子技术与各行各业的具体应用相结合的后的产物，是一门综合技术学科；（3）嵌入式系统和具体应用有机的结合在一起，它的升级换代也是和具体产品同步进行的，因此嵌入式产品一旦进入市场，就有较长的生命周期；（4）为了提高执行速度和可靠性，嵌入式系统中的软件一般都固化在存储器芯片或单片机本身中，而不是存储于磁盘等载体中；（5）.嵌入式系统本身不具有自主开发能力,即使设计完成以后用户通常也不能对其中的程序功能进行修改，必须有一套开发工具和环境才能进行开发。

1.3 ARM 处理器有几种寻址方式，说明各种寻址的方式。

答：

立即寻址：操作数直接放在指令中。例如：ADD　R0，R0，＃0x3f　；R0←R0＋0x3f

寄存器寻址：操作数放在寄存器中。例如：ADD　R0，R1，R2　　；R0←R1＋R2

寄存器间接寻址：操作数在内存，以寄存器中的值作为操作数的地址。

例如：LDR　R0，[R1]　　；R0←[R1]

基址加偏移量寻址（基址变址寻址）：基址寄存器的内容与指令中的偏移量相加形成操作数的有效地址

如：LDR R0，[R1，＃4]　；R0←[R1＋4]　　LDR R0，[R1，R2]　；R0←[R1＋R2]

多寄存器寻址：一条指令可以完成多个寄存器值的传送。例如：

LDMIA R0，{R1，R2，R3，R4}　；R1←[R0]；R2←[R0＋4]；R3←[R0＋8]；

R4←[R0＋12]

堆栈寻址：堆栈是一种数据结构，按先进后出（First In Last Out，FILO）的方式工作，使用一个称作堆栈指针的专用寄存器指示当前的操作位置，堆栈指针总是指向栈顶。

例如：STMFD　R13!，{R0，R4-R12，LR}

LDMFD　R13!，{R0，R4-R12，PC}

1.4 举例介绍嵌入式处理器有哪几类？

答：1.嵌入式微处理器（Embedded Microprocessor Unit,EMPU）；2.嵌入式微控制器；（Embedded Microcontroller Unit,EMCU）

3.嵌入式 DSP 处理器（Embedded Digital Signal Processor,EDSP）；　4.嵌入式片上系统（Embedded System on Chip,EsoC）；

3.1 编写 1+2+3+***+100 的汇编程序。

```
    AREA EXAMPLE1, CODE,READONLY      ;定义一个代码段，名称为 EXAMPLE1
    ENTRY                             ;入口
        MOV     R0,#0                 ;给 R0 赋值为 0
        MOV     R1,#0                 ;用 R1 来存放 1~100 的总和，初始化为 0
START   ADD     R0,R0,#1              ;用来判断终止的，每次加 1
        ADD     R1,R1,R0              ;从 1 加到 100
        CMP     R0,#100               ;执行 R0-100 的操作，但不保存，只影响 CPSR 的值
    BLT         START                 ;带符号数小于时跳转到 START 处执行
    STOP    B       STOP              ;死循环
        END
```

3.2 如何实现 128 位数的减法，请举例说明。

```
AREA        EXAMPLE2,CODE,READONLY
            ENTRY
START
            SUBS        R5,R5,R9
            SBCS        R4,R4,R8
            SBCS        R3,R3,R7
            SBCS        R2,R2,R6
STOP
            B       STOP
END
```

3.3 将存储器中起始地址地址 0X10 处的 4 个字数据移动到地址 0X20 处。

```
    AREA        EXAMPLE2,CODE,READONLY
    ENTRY
    LDR         R0,=0X10            ;把地址 0X10 赋给 R0
    LDR         R5,=0X20            ;把地址 0X20 赋给 R5
START
    LDMIA       R0,{R1-R4}          ;把 R0 为首地址的内存单元中的值依次赋给，
                                    ; R1 到 R4，每次赋完一次值，R0 自动加 1
    STMIA       R5,{R1-R4}          ;把 R1 到 R4 的值依次赋给以 R5 为首地址的内存单元中，
R5 每次自动加 1
STOP
    B           STOP                ;死循环
END
```

3.4 参考 CPSR 寄存器中各标志位的含义，使处理器处于系统模式。

```
AREA        EXAMPLE2,CODE,READONLY
                ENTRY
START
 MOV        R0,#0X1F                    ;给 R0 赋值，2 进制为 11111
 MSR        CPSR_c,R0                   ;把 CPSR 的条件位置 1
STOP
B       STOP                            ;死循环
                END
```

3.5 用跳转指令实现两段程序间的来回切换。

```
    AREA        EXAMPLE5      ,CODE,    READONLY
    ENTRY
    BL   X                                  ;跳到 a 对 R0，R1，赋值
START
    CMP         R0,R1                       ;比较 R0，R1 的值
    BNE Y                                   ;不等跳转到 b
    BEQ STOP                                ;相等时跳转到 STOP
X
    MOV         R0,#3                       ;对 R0，R1 赋值
    MOV         R1,#2
    MOV         R15,R14                     ;返回
```

```
Y
    ADD         R1,R1,#1                        ;R1 自加 1
    B       START                               ;跳转到 START
STOP
    B       STOP                                ;死循环
        END
```

3.5 ARM 调用 Thumb 子程序例子

```
        AREA ThumbSub, CODE, READONLY     ; Name this block of code
        ENTRY                             ; Mark first instruction to execute
        CODE32                            ; Subsequent instructions are ARM
header
        ADR     r0, start + 1             ; Processor starts in ARM state,
        BX      r0                        ; so small ARM code header used
                                          ; to call Thumb main program.
        CODE16                            ; Subsequent instructions are Thumb.
start
        MOV     r0, #10                   ; Set up parameters
        MOV     r1, #3
        BL      doadd                     ; Call subroutine
stop
        MOV     r0, #0x18                 ; angel_SWIreason_ReportException
        LDR     r1, =0x20026              ; ADP_Stopped_ApplicationExit
        SWI     0xAB                      ; Thumb semihosting SWI
doadd
        ADD     r0, r0, r1                ; Subroutine code
        MOV     pc, lr                    ; Return from subroutine.
        END                               ; Mark end of file
```

3.6 字符串拷贝子程序，将 r1 指向的字符串拷贝到 r0 指向的地方，字符串以 0 作结束标志

```
        AREA    SCopy, CODE, READONLY
        EXPORT strcopy
strcopy
        ; r0 points to destination string
        ; r1 points to source string
        LDRB    r2, [r1],#1         ; load byte and update address
        STRB    r2, [r0],#1         ; store byte and update address;
        CMP     r2, #0              ; check for zero terminator
        BNE     strcopy             ; keep going if not
        MOV     pc,lr               ; Return
        END
```

3.7、求两个数的最大值，并将最大值放 R0 寄存器

```
    AREA MAX,CODE,READONLY
    ENTRY
START
    MOV R1,#0x16
```

```
    M0V    R2,#0x17
    CMP R1,R2
    MOVMI R0,R2
    MOVPL R0,R1
STOP
        B       STOP
END
```

3.8、循环实现数的递减，即每次减 1。最终结果为 0 时退出循环。

```
    AREA MAX,CODE,READONLY
    ENTRY
START
    MOV R0,#10
LOOP
    SUBS R0,R0,#1
    BNE LOOP
STOP
        B       STOP
END
```

3.9 、改下列 C 程序段代码为 ARM 汇编程序段代码。

```
void gcd( int a, int b)
{
    while(a!=b)
        if (a>b)
            a=a-b;
        else
            b=b-a;
}
AREA MAX,CODE,READONLY
    ENTRY
gcd
    CMP R0,R1
    BEQ STOP
    BLT Less
    SUB R0,R0,R1
    B STOP
Less
    SUB R1,R1,R0
STOP
        B       STOP
End
```

4.1 什么是伪指令和伪操作？在 ARM 汇编中有哪几种伪 指令？

答：在 ARM 汇编语言程序中有些特殊助记符，这些助记符与一般指令的助记符的不同之处在于没有相对应的操作码或者机器码，通常称这些特殊指令助记符卫伪指令，他们多完成的操作成为伪操作；

在 ARM 汇编中，有如下几种伪指令：(1)符号定义伪指令； (2)数据定义伪指令；(3)汇编控制伪指令；(4)信息报告伪指令；
(5)宏指令及其他伪指令。

4.2 如何定义寄存器列表，试举一个使用寄存器列表的例子，要求实现 4 个字的内存复制。

```
    AREA        EXAMPLE1,CODE,READONLY
        ENTRY
LDR             R0,=0XFF                ;把地址 0XFF 赋给 R0
    LDR             R5,=0X0F                ;把地址 0X0F 赋给 R5
START
PBLOCK      RLIST       {R1-R4}     ;把 R1-R4 定义为 PBLOCK
LDMIA       R0,PBLOCK               ;把 R0 为首地址的内存 4 个字单元装载至 R1 到 R4 中
STMIA       R5,PBLOCK               ;把 R1 到 R4 的值依次存至 R5 为首地址的内存字单元
STOP        B               STOP            ;死循环
                END
```

4.3 如何定义一个宏，宏与子程序的区别是什么？

答：宏的格式为：

MARCO 和 MEND

[$标号]　宏名　　［＄参数 1，＄参数 2…..］

指令序列

MEND

MARCO 表示一个宏定义的开始，MEND 表示一个宏的结束，MARCO 和 MEND 前呼后应可以将一段代码定义为一个整体，又称宏，然后在程序中就可以在程序中通过宏的名称及参数调用该段代码。

宏指令可以重复使用，这一点的使用方式与子程序有些相似，子程序可以节省存储空间，提供模块化的程序设计。但使用子程序机构时需要保存/恢复现场，从而增加了系统的开销，因此，在代码传递的参数较多并且比较短时，可以使用宏代替子程序，宏在被调用的地方展开。

4.4　ARM 汇编中如何定义一个段，段有几种属性？

答：AREA 用于定义一个代码段，数据段，或者特定属性的段。

段的几种属性如下：READONLY 表示只读属性；READWRITE 表示本段可读写；CODE 定义代码段；DATA 定义数据段；

ALIGN=表达式的对齐方式为 2 的表达式次方；；COMMON：定义一个通用段，这个段不包含用户代码和数据。

4.5　在一个汇编源文件中如何包含另一个文件中的内容？

答：通常可以使用 GET/INCLUDE 指令，在某源文件中定义一些宏指令，用 MAP 和 FIELD 定义结构化数据结构类型，用 EQU 定义常量的符号名称，然后用 GET/INCLUDE 将这个源文件包含到其他源文件中。

4.6 分别编写一个函数和一个宏，实现字符串的复制。

```
        MACRO
        COPY        $P1,$P2         ;定义一个宏
$P1   DCB         "HELLO"         ;分配一个字节的空间，并初始化为一个字符串
$P2   DCB         "WORLD!"        ;同上
        LDR       R0,=$P1         ;把 P1 的首地址加载到 R0
        LDR       R6,=$P2         ;把 P2 的首地址加载到 R1
```

```
        LDRIA     R0,[R1-R5]       ;
        STRIA     R6,[R1-R5]       ;
        MEND
         AREA        COPY,CODE,READONLY
        ENTRY
START
        COPY    STR1,STR2                                   ;调用宏
        B         START                                     ;死循环
        END
```

4.9 说明在高速缓存命中率低时对程序性能的影响。

答：高速缓存的命中率随缓冲区的增加而提高。高速缓存的命中减少了对磁盘的访问，并因此提高了系统的整体性能。如果高速缓存的命中率降低，高速缓存对 CPU 的有效访问就会减弱，系统运行就会变慢，程序的运行就会变慢。

```
MOV R4, #0x0          ;i = 0,1,2,3...
Next
 LDR R6, [R1, R4]              ;b[i]
 ADD R6, R6, R2               ;b[i]+c
 STR R6, [R0, R4]                    ;a[i]=b[i]+c
ADD R4, R4, #0x1
 CMP R4, R3
 BLS     Next       ;无符号数小于或等于
```

答：由于 0x4000 是大端组织，所以 R0 存进后，0x4000 中有数 0x56781234,从 0x4000 加载一个字节到 R2 后（由地位到高位加载），R2 的值为 0x34。

5.⑴.MMU：MMU 是存储器管理单元的缩写，是用来管理虚拟内存系统的器件。MMU 通常是 CPU 的一部分，本身有少量存储空间存放从虚拟地址到物理地址的匹配表。此表称作 TLB(转换旁置缓冲区)。所有数据请求都送往 MMU，由 MMU 决定数据是在 RAM 内还是在大容量存储器设备内。如果数据不在存储空间内，MMU 将产生页面错误中断。

MMU 的两个主要功能是:1.将虚地址转换成物理地址。 2.控制存储器存取允许。MMU 关掉时，虚地址直接输出到物理地址总线。

在实践中，使用 MMU 解决了如下几个问题：

①使用 DRAM 作为大容量存储器时，如果 DRAM 的物理地址不连续，这将给程序的编写调试造成极大不便，而适当配置 MMU 可将其转换成虚拟地址连续的空间。②ARM 内核的中断向量表要求放在 0 地址，对于 ROM 在 0 地址的情况，无法调试中断服务程序，所以在调试阶段有必要将可读写的存储器空间映射到 0 地址。③系统的某些地址段是不允许被访问的，否则会产生不可预料的后果,为了避免这类错误,可以通过 MMU 匹配表的设置将这些地址段设为用户不可存取类型。

启动程序中生成的匹配表中包含地址映射，存储页大小(1M,64K,或 4K)以及是否允许存取等信息。

例如:目标板上的 16 兆 DRAM 的物理地址区间为 0xc000，0000～0xc07f，ffff;0xc100，0000～0xc17f，ffff;16 兆 ROM 的虚拟地址区间为:0x0000，0000～0x00ff，ffff。匹配表配置如下:

可以看到左边是连续的虚拟地址空间，右边是不连续的物理地址空间，而且将 DRAM 映射到了 0 地址区间。MMU 通过虚拟地址和页面表位置信息，按照转换逻辑获得对应物理地址，输出到地址总线上。

应注意到的是使能 MMU 后，程序继续运行，但是对于程序员来说程序计数器的指针已经改变，指向了 ROM 所对应的虚拟地址。

MMU 的作用有两个：地址翻译和地址保护。 软件的职责是配置页表，硬件的职责是根据页表完成地址翻译和保护工作。 那三个函数是用来访问页表的。如果 cpu 没有硬件 MMU 那么这张表将毫无意义。 你必须从 cpu 的角度去理解内存映射这个概念。内存映射不是调用一个函数，然后读取返回值。而是 cpu 通过 MMU 把一条指令中要访问的地址转换为物理地址，然后发送到总线上的过程。

6.通用处理器属于复杂指令集计算机(CISC)体系结构，如 Intel 的 Pentium 和 AMD 的 Athlon 处理器。

嵌入式处理器都是精简指令集计算机（RISC）体系。其中 ARM 处理器占了很大一部分市场份额，主要包括如下几个系列：ARM7、ARM9、ARM10、ARM11、XScale。

CISC 和 RISC 是 CPU 指令集的两种架构。其中，RISC 充分发掘并运用了 80/20 法则(CISC 指令集中只有大约 20%的指令被反复使用)，要求指令规整、对称和简单，在并行处理性能上明显优于 CISC，可以使处理器流水线高效地执行，使编译器更易于生成优化代码。

从硬件角度来看 CISC 处理的是不等长指令集，它必须对不等长指令进行分割，因此在执行单一指令的时候需要进行较多的处理工作。而 RISC 执行的是等长精简指令集，CPU 在执行指令的时候速度较快且性能稳定。因此在并行处理方面 RISC 明显优于 CISC，RISC 可同时执行多条指令，它可将一条指令分割成若干个进程或线程，交由多个处理器同时执行。由于 RISC 执行的是精简指令集，所以它的制造工艺简单且成本低廉。

从软件角度来看，CISC 运行的则是我们所熟识的 DOS、Windows 操作系统。而且它拥有大量的应用程序。因为全世界有 65%以上的软件厂商都理为基于 CISC 体系结构的 PC 及其兼容机服务的，象赫赫有名的 Microsoft 就是其中的一家。而 RISC 在此方面却显得有些势单力薄。虽然在 RISC 上也可运行 DOS、Windows，但是需要一个翻译过程，所以运行速度要慢许多。