**Stage1\_2014.10.26 初级班试卷**

|  |  |  |  |
| --- | --- | --- | --- |
| 班别: | Stage1\_2014.10.26 初级班 | 日期: |  |
| 类别: | 软件: U-boot and Diagnostic code | 级别: | 初级班 |
| 工号: |  | 姓名: |  |

**试卷作答 (60%)**

|  |
| --- |
| 1.  ARM寄存器: R0 = 0x00000000, R1 = 0x00001100, R2 = 0x00220000, R3 = 0x33000000, CPSR = 0x0000\_0010;  当执行完下面汇编程序码  MOV R0, R3  SUBS R1, R2, #0x00220000  ADD R3, R1, R3 LSR#4  求R0, R1, R2, R3, CPSR (10%) |
| Answer:  R0 = 0x3300\_0000; R1 = 0x0000\_0000; R2 = 0x0022\_0000; R3 = 0x0330\_0000; CPSR = 0x6000\_0010;  R0:0x3300\_0000 2%; R1:0x0000\_0000 2%; R2: 0x0020\_0000 2%; R3: 0x0330\_0000 2%; CPSR: 0x6000\_0010 2% |
| 2.  U-boot 1.14起始代码名称, 在那个连接描述文件所定义, 在cheetach板上定义TEXT\_BASE多少, 在那档案所定义(10%) |
| Answer:  U-boot 1.14 在档案cpu/arm926ejs/start.s (/board/cheetah/u-boot.lds)定义起始代码为\_start  TEXT\_BASE在档案 /cheetach/config.mk 宣告, 其值为0x4040\_0000  u-boot.lds(start.s): 2%; \_start: 3%; config.mk: 2%; 0x40400000: 3% |
| 3.  叙述在start.s(U-boot 1.14)以下代码功用(10%)  relocate: /\* relocate U-Boot to RAM \*/  adr r0, \_start /\* r0 <- current position of code \*/  ldr r1, \_TEXT\_BASE /\* test if we run from flash or RAM \*/  cmp r0, r1 /\* don't reloc during debug \*/  beq stack\_setup  ldr r2, \_armboot\_start  ldr r3, \_bss\_start  sub r2, r3, r2 /\* r2 <- size of armboot \*/  add r2, r0, r2 /\* r2 <- source end address \*/  copy\_loop:  ldmia r0!, {r3-r10} /\* copy from source address [r0] \*/  stmia r1!, {r3-r10} /\* copy to target address [r1] \*/  cmp r0, r2 /\* until source end addreee [r2] \*/  ble copy\_loop |
| Answer:  relocate:  当前代码的地址\_start地址是否与TEXT\_BASE一致，若一致则跳到堆栈设定程序stack\_setup，若不一致则继续执行，将\_armboot\_start存入R2，将\_bss\_start存入R3，再计算出u-boot映像的大小  copy\_loop:  将代码由Nor Flash复制到内存TEXT\_BASE指定的内存空间，再执行堆栈设定程序stack\_setup  \_start == TEXT\_BASE 1%; 判断ram或rom: 4%; 在ram执行执行stack\_steup 1%; 在rom执行复制到ram: 4% |
| 4.  U-boot 1.14透过哪个命令引导Linux kernel, 描述由命令到引导Linux kernel 的过程(10%) |
| Answer:  bootcmd=bootm 16120000(内核IMAGE地址) 16300000( file system image address) 命令引导Linux kernel  这个命令实际调用do\_bootm()函数, 读取header, 把内核copy到指定的地方, 把参数给内核准备好,并告诉内核参数的首地址, 调用do\_bootm\_linux()引导内核  bootcmd(bootm): 3%; do\_boot: 2%; 过程: 4%; do\_boot\_linux: 1% |
| 5.  在Diagnostic测试时, 当EXT0\_INT中断触发时, 描述ARM CPU触发中断到调用相对应的函数过程  (Ref: Cheetah\_Connectivity\_Specification\_v.B.0.3-128.pdf) (10%) |
| Answer:  当EXT0\_INT中断触发时,ARM CPU会通过vector table找到\_irq,跳到irq去执行”bl irq\_dispatch”调用irq\_dispatch()函数,读取interrupt controller的状态寄存器, 根据对应的IRQ number 找到 对应的中断处理函数,判断发生了EXT0\_INT中断,判断是否有ISR处理函数,没有则直接return;有中断处理函数,则执行该函数,然后清掉EXT0\_INT中断标志  vector table: 4%; irq\_dispatch: 3%; EXT0\_INT中断: 3% |
| 6.  描述DMA功用, S/W Trigger DMA Mode 和 H/W Trigger DMA Mode 的差别 (Ref: Cheetah\_Connectivity\_Specification\_v.B.0.3-128.pdf) (10%) |
| Answer:  DMA的功用:  替CPU处理外部设备与内存之间的数据传输,以便CPU在数据传输过程中可以进行其他的工作,提高CPU的工作效率  SW trigger DMA mode与HW trigger DMA mode区别:  SW通过配置register触发DMA,HW是外部设备发请求信号  DMA作用: 5%; S/W, H/W内容, 差别: 5% |

**上机实操 (40%)**

|  |
| --- |
| 1. 在CDK Diagnostic code增加测式函数名称为ass\_test(), 文件名称为ass.c, 为改写使用汇编语言, 由内存位为址0x42000000，写入0x12003400的pattern，大小为0x20 byte的资料;再把内存地址为0x42000000的数据，大小为0x10 byte数据，搬到内存地址0x42000020，并在搬移过程中，把来源地址的数据清除为0 (40%) |
| (在此栏附上汇编代码, 再使用RVD写入0x20 byte数据的pattern后图片一张, 最后执行结果图片一张, 共两张图片)程0x42000000写入0x20bytes后:  01.PNG  Copy到0x42000020 0x10bytes后:  02.PNG  "ldr r0, =0x12003400 \n"  "mov r1, #0x42000000 \n"  "mov r2, #0x20 \n"  "loop: \n"  "str r0, [r1], #0x04 \n"  "subs r2, r2, #0x04 \n"  "bne loop \n"  "ldr r0, =0x42000000 \n"  "ldr r1, =0x42000020 \n"  "mov r2, #0x10 \n"  "copy\_loop: \n"  "ldr r3, [r0], #0x04 \n"  "str r3, [r1], #0x04 \n"  "mov r3, #0x0 \n"  "str r3, [r0, #-0x04] \n"  "subs r2, r2, #0x04 \n"  "bne copy\_loop \n"  上机结果: 20%; 源码架构效率: 20%; 没上传源码, 或未贴上图片: -2% |

git路径：git clone git@10.193.200.199:2014/工号/Stage1/exam/U-boot

上传档案: 1. U-boot\_exam.docx, 2. ass.c