

drwxr-xr-x 2 root root 4096 11-04 17:10 test2

① 修改 a.c b.c 权限为 777, 777, 代码如下:

```
[root@JLU2H a2] # chmod 777 a.c b.c
```

```
[root@JLU2H a2] # ls -l
```

drwxr-xr-x 2 root root 4096 11-04 17:10 a

drwxr-xr-x 2 root root 4096 11-04 21:08 a.c

drwxr-xr-x 2 root root 4096 11-04 21:20 a.c

drwxr-xr-x 2 root root 4096 11-04 17:10 b.c

drwxr-xr-x 2 root root 4096 11-04 17:10 test1

drwxr-xr-x 2 root root 4096 11-04 17:10 test2

② 掌握建立软连接，硬连接及删除寻常命令。

① 使用 ln 建立软连接 link1 指向 a.c 硬连接 link2 指向 b.c 代码如下

```
[root@JLU2H a2] # -s link1 a.c
```

```
[root@JLU2H a2] # -s link2 b.c
```

② 使用 ls -l, 代码如下:

```
[root@JLU2H a2] # ls -l
```

总计 24

drwxr-xr-x 2 root root 4096 11-04 17:10 a

drwxr-xr-x 2 root root 4096 11-04 21:08 a.c



实验日期：_____年____月____日 实验名称：_____

drwxr-xr-x 2 root root 4096 11-04 17:10 test

drwxr-xr-x 2 root root 4096 11-04 17:10 test

② 使用 rm 删除 a.c，用 ls -l 查看，注意链接的信息代码如下

[root@JLUV2H ~] # rm -r a.c

rm: 是否进入目录 "a.c"? y

rm: 是否删除 符号链接 "a.c[links]"? y

rm: 是否删除 目录 "a.c"? y

[root@JLUV2H ~] # ls -l

④ 使用 rm 命令将 test 目录全部删除 例如如下:

[root@JLUZH test]# rm -r a2 b2

rm: 是否进入目录 "a2"? y

rm: 是否删除 目录 "a2/test2"? y

rm: 是否删除 目录 "a2/a1"? y

rm: 是否删除 目录 "a2/test1"? y

rm: 是否删除 目录 "a2/b2/c"? y

rm: 是否删除 目录 "a2/a"? y

rm: 是否删除 目录 "a2"? y b1, b2 同上

[root@JLUZH test]# ls

2. U 盘 + 打包 + 压缩 + 解压

① 插入 U 盘，先查看已经挂载的设备，例如如下：

[root@JLUZH ~]# fdisk -l

Disk /dev/sda: 139 GB, 13958643712 bytes

255 heads, 63 sectors/track, 1697 cylinders

Units = cylinders of 16065 * 172 = 8225180 bytes

Disk identifier: 0x0009124c

- Device Boot Start End Blocks Id System

/dev/sda1 * 1 13 102400 83 Linux

Partition 1 does not end on cylinder boundary

/dev/sda1 1 13 524288 83 Linux

Partition 2 does not end on cylinder boundary

ldev/sda3 79 1697 13004033 83 . Linux

Disk ldev/sdb1 1048 MB. 1048576000 bytes

128 heads , 32 sectors /track , 500 cylinders

Units = cylinders of $4096 \times 512 = 2097152$ bytes

Disk identifier : 0x00000000

Device	Boot	start	End	Block Id	System
ldev/sdb1	*	1	500	1023984	4 FAT16 <32m

Partition 1 has different physical / logical entit

phys = (502, 127, 32) logical = (497, 127, 32)

② 然后把 U 盘 挂在到 /mnt/usb 目录下，代码如下：

```
[root@JLUZH ~]# mkdir /mnt/usb
```

```
[root@JLUZH ~]# mount -t vfat /dev/sdb1 /mnt/usb
```

```
[root@JLUZH ~]# ls /mnt/usb
```

boot.log.txt

cn.bat

command.com

config.exe

[root@JL

③ 将 U 盘 中的

[root@JU

[root@JU

[root@JU

test 公共的

④ 卸载 U 盘 进入

[root@JL

[root@JL

[root@JL

bootlog.txt ghofiles.exe net setup.exe 新建文件夹
cn.bat ChosrvLoader.exe NetBoot setvar.bat
command.com ghost netfiles.exe shsuctx.exe
config.exe GhostSrv.exe netghost .soft

[root@JLV2H ~]#

③ 将U盘中的test文件夹拷贝到 /root 目录下，代码如下：

[root@JLV2H usb]# cp -a ./test /root/

[root@JLV2H usb]# cd

[root@JLV2H ~]# ls

test 公共的 桌板 视频 图片 文档 音乐 桌面

④ 卸载U盘，进入到 test 目录下将里面的文件用 tar 打包，代码如下：

[root@JLV2H ~]# umount /dev/sdb1

[root@JLV2H ~]# ls /mnt/usb

[root@JLV2H ~]# tar -cvf test.tar ./test

./test/a/

./test/b/

./test/a/

./test/b/

⑤ 查看文件大小，代码如下：

[root@JLV2H ~]# du -sh test.tar

实验日期: _____ 年 _____ 月 _____ 日 实验名称: _____

⑥ 再用 gzip 压缩，代码如下

```
[root@JLUZH ~] # gzip test.tar
```

```
[root@JLUZH ~] # ls
```

samba -3.4.3 test.tar.gz 模板、图片、音乐、桌面
test 公共的 视频、文档、音乐

⑦ 查看文件大小，代码如下

```
[root@JLUZH ~] # du -sh test.tar.gz
```

4.0K test.tar.gz

⑧ 使用 tar 解压和解包，代码如下：

```
[root@JLUZH ~] # tar -xvf test
```

实验日

二 实

1. 熟

2. 掌

3. 享

二 实

Tar

三 宝

建立 A

四

实验日期：2020年12月30日 实验名称：TQ2440裸机-LEDXJ

一、实验目的

1. 熟悉ADS开发环境

2. 掌握S3C2440的寄存器的配置

3. 懂得C语言编程

二、实验设备

TQ2440开发板 PC机+JTAG调试板

三、实验内容

建立ADS开发环境，编程实现对开发板上LED的跑马灯控制

四、实验原理

1. 相应寄存器介绍：

LED灯是接在某一个I/O上的，点亮或者熄灭LED灯其实就是对I/O寄存器的操作。对于ARM芯片来说，也有这样的问题，因为它其中的寄存器很多，为了能让ARM找到相应的寄存器，系统给每个寄存器都分配一个固定地址，地址就像一个人的身份证号一样是唯一的。那么使用该地址时，ARM就知道是哪个寄存器了，所以在编程时，首先要声明操作的寄存器的地址如下：

Register	Address	R/W	Description	Reset Value
GPBCON	0x56000010	R/W	Configures the pins of portB	0x0
GPBDAT	0x56000014	R/W	The data register for portB undefined	
GPBVUP	0x56000018	R/W	pull-up disable register for portB	0x0
Reserved	0x5600001C			

GPB8	[17:16]	00 = Input 10 = nxDREQ1	01 = Output 11 = Reserved
GPB7	[15:14]	00 = Input 10 = nxDACK1	01 = Output 11 = Reserved
GPB6	[13:12]	00 = Input 10 = nxBLER	01 = Output 11 = Reserved
GPB5	[11:10]	00 = Input 10 = nxBACK	01 = Output 11 = Reserved
GPB4	[9:8]	00 = Input 10 = TCLK[D]	01 = Output 11 = Reserved
GPB3	[7:6]	00 = Input 10 = TUV3	01 = Output 11 = Reserved
GPB2	[5:4]	00 = Input 10 = TUV2	01 = Output 11 = Reserved
GPB1	[3:2]	00 = Input 10 = TUV1	01 = Output 11 = Reserved
GPB0	[1:0]	00 = Input 10 = TUV0	01 = Output 11 = Reserved

ARM设置完I/O状态后，就准备读/写数据了。这个功能由W、通过设置数据寄存器 GPBDAT 来实现。下表为其功能描述。

GPBDAT	Bit	Description
GPB[10:0]	[10:0]	When the port is configured as input, the corresponding bit is the pin state. When the port is configured as output port: the pin state is the same as the port.

When the port is configured as functional pin, the undefined value will be read.

设置完前后两个寄存器然后，还要设置上拉电阻使能寄存器 GPBUP。上拉电阻下拉电阻作用在于当 I/O 引脚 P 处于第三态时，它的电平状态由下拉电阻、上拉电阻确定。下拉为它的功能描述，当为 0 时，上拉电阻是允许的，反之是禁止的。

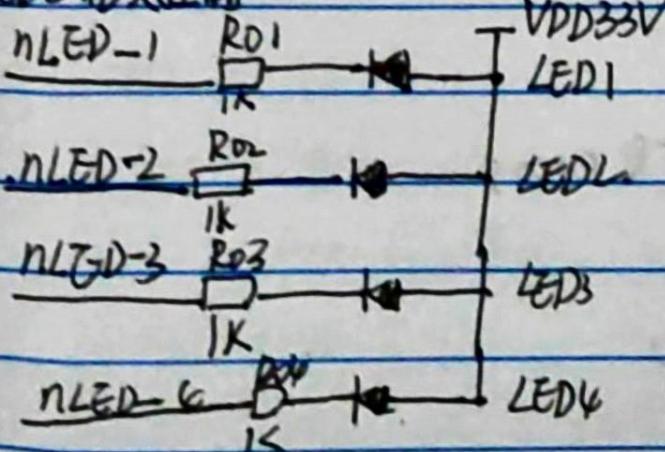
GPBUP	Bit	Description
GPB[10:0]	[10:0]	0: the pull up function attached to the corresponding port is enabled. 1: the pull up function is disabled.

五 实验电路图

这个电路采用灌电流方式驱动 LED；只需加一路流电阻即可，可算出限流电阻 $R = (V_1(\text{电源}) - V_{LED}(\text{端电压})) / I_D(\text{灌电流})$

通过控制 GPB5, GPB6, GPB8, GPB10, 实现对 LED0, LED1, LED2,

LED3 亮灭控制



- $nxDACKo \rightarrow 0 \cdot nxDACKo / GPB7$
- $nLED3 \rightarrow 0 \cdot nxDACKo / GPB7$
- $nxDREG0 \rightarrow 0 \cdot nxDREG0 / GPB10$
- $nLED4 \rightarrow 0 \cdot nxDREG0 / GPB8$
- $nLED1 \rightarrow 0 \cdot nxDREG0 / GPB8$
- $nLED2 \rightarrow 0 \cdot nxDREG0 / GPB6$

六、实验步骤

1. 点亮一个LED灯

inits 程序

AREA [DATA], CODE, READONLY

ENTRY

ldr r13, =0x1000

IMPORT Ledo

b Ledo

END

Ledo.c 主程序

#define GPBCON (*volatile unsigned *) 0x8000

#define GPBDAT (*volatile unsigned *) 0x8000

#define GPBUP (*volatile unsigned *) 0x8000

int Ledo (void)

{ GPBCON = 0x400;

GPBDAT = GPBDAT & 0x20;

GPBUP = 0xFF;

六 实验步骤

1. 点亮一个LED灯

inits 程序

AREA [DATA], CODE, READONLY

ENTRY

ldr r3, =0x1000

②

IMPORT R0

③

b-ledo

④

END

⑤

Ledo.c 主程序

```
#define GPBCON (* (volatile unsigned *) 0x80000010)
#define GPBDAT (* (volatile unsigned *) 0x80000010)
#define GPBUP (* (volatile unsigned *) 0x80000010)

int Ledo (void)
{
    GPBCON = 0x400;
    GPBDAT = GPBDAT & 0x40;
    GPBUP = 0xff;
    GPBDAT = ~ (1<<5);
    while (1);
    return 0;
}
```

实验日期：_____年____月____日 实验名称：_____

Debug Rel setting #ARM Linker # Layout -

· object / symbol 应添 init.o

· section 添 init

init.s:

AREA init, CODE, READONLY

ENTRY

IMPORT main

b main

END

Main.c

#define GPBICON (* (volatile unsigned *) 0x56000010)
#define GPBDAT (* (volatile unsigned *) 0x56000010)

#define GPBUP (* (volatile unsigned *) 0x56000018)

void delay()

{ int i, j; }

for (i=900; i>0; i--)

for (j=900; j>0; j--) ;

int Main()

GPBICON = 0x551FC;

GPBUP = 0xFFFF;

GDR·TDAAT = (1<<5) | (1<<6) | (K<7) | (K<8);

~~GPBDAT1 = (1<<7);~~

~~GPBDAT & = ~L1<<8>;~~

~~delay ();~~

~~GPBDAT1 = (1<<8);~~

~~}~~

~~}~~

实验日期： 2016年 7月 9 日 实验名称：

实验日期：

3. 为当前工程添加源程序文件

ARM汇编程序源文件后缀名为S大小写均可

4. 编辑源程序代码

参考程序 adds:

; armadd 源程序

N EQU 7

AREA Adding, CODE, READONLY

ENTRY

MOV R0, #0

MOV R1, #1

REPEAT ADD R0, R0, R1

ADD R1, R1, #1

CMP R1, #N

BLG REPEAT

LDR R2, =RESULT

STR R0, [R2]

HERE B HERE

AREA Dataspace, DATA, READWRITE

RESULT DCD 0

END

右击 add.s 文件，选择 Complie，如果没有成功会弹出错误和警告窗口

6. 编译整个工程

选择 Project → Make 进行整个工程的编译，可以在目录空间查看是否生成了映像文件 add.axf。

7. 确认调试目标设置

8. 运行映像文件

9. 调试准备

10. 调试映像文件

五. 实验结果

序号	执行指令	R0	R1	R2	PC	存储器(十六进制)
1	MOV R0, #0	0X00000000	0X00000008	✓	X	
2	MOV R1, #1	0X00000000	0X00000008	0X00009000	0X00008004	0X9000
3	REPEAT ADD R0, R1, #1	0X00000000	0X00000001	0X00009000	0X00008008	0X9000
4	ADD R1, R1, #1	0X00000001	0X00000001	0X00009000	0X0000800C	0X9000
5	CMP R1, #N	0X00000001	0X00000002	0X00009000	0X00008010	0X9000
6	BLE REPEAT	0X00000001	0X00000002	0X00009000	0X00008014	0X9000
7	ADD R0, R0, #1	0X00000000	0X00000001	0X00009000	0X00008008	0X9000
8	ADD R1, R1, #1	0X00000001	0X00000001	0X00009000	0X0000800C	0X9000

- 12 ADD R1,R1,#1 0x00000001 0x00000001 0x0009000 0x000800C 0x9000
- 13 CMP R1,#N 0x00000001 0x00000002 0x0009000 0x0008010 0x9000
- 14 BLE REPEAT 0x00000000 | 0x00000002 0x0009000 0x0008014 0x9000
- 15 ADD R0,R0,R1 0x00000000 0x00000001 0x0009000 0x0008008 0x9000
- 16 ADD R1,R1,#1 0x00000001 0x00000000 | 0x0009000 0x000800C 0x9000
- 17 CMP R1,#N 0x00000001 0x00000002 0x0009000 0x0008016 0x9000
- 18 LDR R2,=RESULT 0x0000001C 0x00000008 0x0009000 0x0008018 0x9000
- 19 STR R0,[R1] 0x0000001C 0x00000008 0x0009000 0x000801C 0x9000
- 20 B HERE 0x0000001C 0x00000008 0x0009000 0x0008020 0x91C

六 实验总结

通过这次学习，我学到了如何使用不同的指令，并熟悉了指令的格式
并通过上机知道了指令的执行过程

实验日期: 2020 年 10 月 16 日 实验名称: ARM 汇编与 C 语言混合程序设计

一. 实验目的

- 掌握 C 程序内嵌指令的使用方法
- 理解汇编程序调用 C 程序函数和变量的方法

二. 实验内容

- 使用内嵌汇编的方式设计允许和禁止中断程序
- 验证汇编程序调用 C 程序函数和访问 C 程序变量的执行过程

三. 实验步骤

(1) 验证汇编程序调用 C 程序函数和访问 C 程序变量的执行过程
采用 ARMulator 方法调试，选择用 ARM9 作为目标处理器

(1) 新建 ARM 工程 exp3

启动 ADS 开发环境，选择 File → new[Project] 选项，使用 ARM Executable Image 工程模板创建 exp3

(2) 新建 C 程序文件 test.c，并将其添加到工程 exp3 中
被调用的 C 程序

```
#include <stdio.h>
unsigned long sum=6;
extern int add(int a, int b, int c, int d, int e)
{ return a+b+c+d+e; }
```

(3) 新建汇编程序文件 test_asm.s，并将其添加到工程 exp3 中
选择 File → New[File] 选项，新建汇编源程序文件 test_asm.s，调用 C 程序中的函数 add 和变量 sum，并添加到工

MOV R2, #5

MOV R3, #4

MOV Rx, #5

STR Rx, [SP, #-4])

BL add

LDR R1, =sum

STR R0, [R1]

ADD SP, SP, #4

LDR PC, [SP], #4

END

实验日期：_____年_____月_____日 实验名称：_____

④ 设置工程 exp3 的编译和链接选项

选择 Edit → DebugRel settings 选项，打开 DebugRel setting 对话框，设置工程编译和链接选项，在 language settings → ARM Assembler 选项中，选择 Target 选项卡，修改处理器类型为 ARM920T。

⑤ 在AXD中加载映像文件

打开 AXD Debugger，首先确认调试目标机是否设置为 Armulator，选择 Options → Configure Target (choose Target) 选项，确认 ARM920T 为选中状态。然后单击 Configure 按钮，打开 Armulator Configuration 对话框，确认 Armulator 模拟的处理器类型是 ARM920T。

选择 File → Load Image 选项，载入工程 exp3 目录 exp3\exp3 Data\DebugRel 下编译生成的映像文件 exp3.axf。

⑥ 调试准备

在 AXD 中，打开各个观察窗口，做调试准备。

① 选择 Processor Views → Register 选项，打开 ARM 寄存器显示窗口。

② 选择 Processor Views → Memory 选项，打开 ARM 存储器显示窗口，并根据堆栈指针，在 Memory Start Address 地址栏输入便于观察堆栈的内容地址。

③ 选择 Processor Views → Variables 选项，打开 ARM 变量观察窗口，并单击 Global 标签，下面会显示变量 sum 的值；如果要看变量 sum 的地址，则可右击 sum，在菜单中选择 Locate Using Address 选项。

④ 选择 Processor Views → Disassembly 选项，打开反汇编显示窗口。



实验日期：_____年_____月_____日 实验名称：_____

(1) 单步运行程序

在AXD中，选择 Execute → Step 菜单项或按 F10 键，或使用调试工具单步运行程序，查看相关寄存器和存储空间的值的变化情况，把每一步的执行结果填入表中。执行 File → Reload Current Image 命令重新加载和运行当前映像文件。

单步运行程序，观察相关寄存器值的变化情况，并填入表中。

(2) 题目1：用汇编语言调用C实现；题目2：实现字符串的快速反转

| *...start.s...* |

不使用半空格，只半空格

global _start

| *...main.c...* |
extern void copychar(ch)

实验日期：2018年6月1日 实验名称：

char:

LDMIA R0!, {R5-R8}

long long Nx =

STMIA R1!, {R5-R8}

for (i=1; i<=N; i++)

SUBS R4, R4, #1

Nx = Nx * i;

Copy 2

return Nx;

ANDS R2, R2, #3

BNE copy - 4 word

BEQ stop

LDMFD SP!, {R5-R8}

Copy - HWord.

Copy 1:

LDRH R3, [R0], #2

ANDS R1, R2, #15

STRH R3, [R1], #2

BEO stop

ANDS R2, Rj, #1

MOV S R4, R2, LSR #2

BEQ stop

BEO copy 2

Copy - B:

Copy - Word:

LDRB R3, [R0]

LDR R3, [R0], #4

STRB R3, [R1]

STR R3, [R1], #0

stop : B stop

SUBS R4, R4, #1

BNG copy - Word

一、实验目的

- 熟悉 VMware Workstation 虚拟机的使用
- 熟练运用 Linux 常用命令

二、实验设备

硬件：PC机

软件：VMware Workstation 虚拟机、red hat linux 操作系统

三、实验预习要求

- 阅读 1.5 节内容

四 实验内容

使用 VMware Workstation 虚拟机运行 linux 操作系统，练习 Linux 命令。

- 文件与目录相关命令：掌握创建目录、文件及文件的拷贝、移动、权限
建立软连接、硬连接及删除等常用命令。设计命令序列完成如下操作：

(1) 在 /root 目录下建立一个目录 test

(2) 进入到该目录下

(10) 进入到

(11) 查看 a

(12) 修改

(13) 使用

(14) 使用 ls

(15) 使用

(16) 使用

(17) 使用

2. U 盘

(1) 抽取出

(2) 然后

(3) 将 U

(4) 卸载

(5) 查看

- (10) 进入到 a2/a 目录下，把 a 中的文件全部 mv 到 a2 目录下
- (11) 查看 a2 目录下所有文件的权限
- (12) 修改 a、c、b、d 权限为 777、777
- (13) 使用 ln 建立软链接 link1 指向 a、c 硬链接 link2 指向 b、c
- (14) 使用 ls -l
- (15) 使用 rm 删除 a、c ls -l 查看
- (16) 使用 rm 删除 a 目录
- (17) 使用 rm 命令将 test 盘全部删除

2. U 盘 + 打包 + 压缩 + 解压

- (1) 插好 U 盘，先查看已经挂载的设备
- (2) 然后把 U 盘挂载到 /mnt/usb 目录下
- (3) 将 U 盘中的 test 文件夹拷贝到 /root 目录下
- (4) 卸载 U 盘，进入到 test 盘下将里面的文件用 tar 打包
- (5) 查看文件大小
- (6) 再用 gzip 压缩
- (7) 查看文件大小
- (8) 使用 tar 解压和解包

五. 实验步骤

- (1) 使用 VMWare Workstation 虚拟机运行 Linux 操作系统

1. 运行 VMWare Workstation 应用程序
2. Start this virtual machine 启动 Linux 操作系统

3. 进入系统后输入账号和密码

4. 成功进入系统后选择应用程序 → 系统工具 → 终端

5. 终端运行以后，就可以在这里输入linux命令并按回车键

(二) 使用linux命令完成的操作

1. linux常用命令

(1) 掌握创建目录、文件及文件的拷贝、移动等常用命令

① 在 /root 目录下，建立一个目录 test。代码如下：

```
[root@JLUV2H] # mkdir /test
```

```
[root@JLUV2H jluzh] # ls
```

test 公共 模板 视频 图片 文档 下载 音乐 家庭

② 进入到该目录下。代码如下：

```
[root@JLUV2H jluzh] # cd /test
```

```
[root@JLUV2H test] #
```

③ 在该目录下，建立目录 a1, a2, b1, b2 代码如下：

```
[root@JLUV2H test] # mkdir a1 a2 b1 b2
```

```
[root@JLUV2H test] # ls
```

5. 终端运行以后，就可以在这里输入linux命令并按回车键

(二) 使用linux命令完成的操作

1. linux 常用命令

(1) 掌握创建目录、文件及文件的拷贝、移动等常用命令

① 在 /root 目录下，建立一个目录 test。代码如下：

```
[root@JLUZH] # mkdir /test
```

```
[root@JLUZH jluzh] # ls
```

test 公共的 模板 视频 图片 文档 下载 音乐 家园

② 进入到该目录下。代码如下：

```
[root@JLUZH jluzh] # cd /test
```

```
[root@JLUZH test] #
```

③ 在该目录下，建立目录 a1, a2, b1, b2 代码如下：

实验日期: _____ 年 _____ 月 _____ 日 实验名称: _____

⑤ 将 a 目录下所有文件拷贝到 a1 目录。代码如下:

```
[root@JLUZH test] # cp -a ..\a\* .\a1\
```

```
[root@JLUZH test] # cd a1
```

```
[root@JLUZH a1] # ls
```

```
a1 a2 b.c test1 test2
```

```
[root@JLUZH a1] # cd a1
```

```
[root@JLUZH a1] # ls
```

```
a1 a2 b.c test1 test2
```

⑥ 将 a1 目录下所有 c 文件拷贝到 b1 目录。代码如下:

```
[root@JLUZH test] # cp -a ..\a1\* .\b1\
```

```
[root@JLUZH test] # cd b1
```

```
[root@JLUZH b1] # ls
```

```
a.c b.c
```

⑦ 将 a1 目录下所有 test* 文件拷贝到 b2 目录。代码如下:

```
[root@JLUZH test] # cp -a ..\a1\test* .\b2\
```

```
[root@JLUZH test] # cd b2
```

```
[root@JLUZH b2] # ls
```

```
test1 test2
```

⑧ 重命名 a.c 为 a。代码如下:

```
[root@JLUZH a1] # ls
```

[root@JLUZH a] # mv . 1a.4.a

[root@JLUZH a] # ls

a a1 b.c test1 test2

⑧ 将 a 目录下所有文件全部移到 a2 目录下，代码如下：

[root@JLUZH test] # mv -i . 1a.1 * . 1a2

[root@JLUZH test] # cd a2

[root@JLUZH a2] # ls

a a1 b.c test1 test2

⑨ 进入到 a2/a1 目录下，把 a1 中的文件全部 mv 到 a2 目录下，代码如下：

[root@JLUZH test] # mv -i . 1a2/a1/* . 1a2

mv：是否覆盖 ‘. 1a2/test2’? y

[root@JLUZH test] # cd a2/a1

[root@JLUZH a2] # ls

a a1 b.c test1 test2

⑩ 查看 a2 目录下所有文件的权限，代码如下：

[root@JLUZH test] # ls -l . 1a2

总计 20

drwxr-xr-x 2 root root 4096 11-04 17:00 a

drw-rxr-xr-x 2 root root 4096 11-04 21:08 a1

drwxr-xr-x 2 root root 4096 11-04 17:10 b.c

drwxr-xr-x 2 root root 4096 11-04 17:10 test1

1. 在 ADS 中新建工程 并设置开发环境

2. 在 code warrior 环境下编辑 编译和链接汇编
生成可执行文件

3. 在 AXD 中调试汇编程序

4. 使用命令行界面编辑 编译和链接汇编语言

四 实验步骤

1. 新建工程

打开 code warrior 选 File → new . 使用 ARM Executable
新建个工程，存入 D 盘 mywork.

2. 设置编译和链接选项。

：我们使用的是模拟机。设置编译器的模拟器

为 XScale；在 ARM Linker 中选 Output 并选 Linktype 并选
Linktype 确认 Ro Base 为 0x8000，修改 RW Base 为 0x9000