



北京交通大学

BEIJING JIAOTONG UNIVERSITY

北京交通大学

课程名称：嵌入式系统设计

实验题目：ARM指令系统

学号：22281188

姓名：江家玮

班级：计科2204班

指导老师：唐宏老师

报告日期：2024-09-18

目录

- 1 参考ADS使用.PPT熟悉ADS
- 2 关注相关寄存器的变化，实现工作模式切换
 - 2.1 回答问题
 - 2.2 分别修改程序，实现如下功能
 - 2.3 在程序中MOV R14, #4语句后，增加语句
 - 2.4 程序中，使用了LDR伪指令，请分析该伪指令和LDR加载指令在功能和应用上有何区别
 - 2.4.1 LDR伪指令
 - 2.4.2 LDR加载指令
- 3 按求最大公约数.doc的说明编写汇编指令

1 参考ADS使用.PPT熟悉ADS

单步运行42页的指令,熟悉寻址方式,观察指令运行后相关寄存器和内存的变化。分别对 STMIA R4!, {R0-R3} 指令运行后的对应内存和 LDMFD R13!, {R1-R4} 指令运行后的对应寄存器截图。

- STMIA R4!, {R0-R3} 指令运行后:

ARM7TDMI - Registers

Register	Value
r2	0x0000000A
r3	0x0001E200
r4	0x00090024
r5	0x00000000
r6	0x00000000
r7	0x00000000
r8	0x00000000
r9	0x00000000
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13	0x00090200
r14	0x00000000
pc	0x00008030
cpsr	nzcvqIfT_SVC
spsr	nzcvqift_Res
User/System	{...}

8

9

10

11

12

13

14

15

16

17

18

19

20

21

MOV R1,R0

ADD R0,R1,R2

STR R0,[R4]

LDR R3,[R4]

MOV R0,R1, LSL#1

STR R0,[R4,#4]

LDR R3,[R4,#4]!

STMIA R4!,{R0-R3}

LDMIA R4!,{R5,R6,R7,R8}

STMFD R13!,{R5,R6,R7,R8}

LDMFD R13!,{R1-R4}

B START

END

ARM7TDMI - Memory Start addr: 0x90010

÷

×

Tab1 - Hex - No prefix				Tab2 - Hex - No prefix				Tab3 - Hex - No prefix				Tab4 - Hex - No prefix							
Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	ASCII		
0x00090010	0A	F1	00	00	00	E2	01	00	00	F1	00	00	0A	00	00	00		
0x00090020	00	E2	01	00	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090030	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090040	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090050	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090060	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090070	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090080	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090090	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000900A0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000900B0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000900C0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000900D0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000900E0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000900F0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090100	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090110	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090120	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090130	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090140	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090150	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090160	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090170	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090180	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090190	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000901A0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000901B0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000901C0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000901D0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000901E0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x000901F0	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7		
0x00090200	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090210	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090220	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		
0x00090230	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8		

- LDMFD R13!, {R1-R4} 指令运行后:

ARM7TDMI - Registers

Register	Value
Current	{...}
r0	0x0001E200
r1	0xE800E800
r2	0xE7FF0010
r3	0xE800E800
r4	0xE7FF0010
r5	0xE800E800
r6	0xE7FF0010
r7	0xE800E800
r8	0xE7FF0010
r9	0x00000000
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13	0x00090200
r14	0x00000000
pc	0x0000803C
cpsr	nzcvqIfT_SVC
spsr	nzcvqift_Res

4

START

LDR R4,=0x00090010

5

LDR R13,=0x00090200

6

MOV R0,#0x0F100

7

MOV R2,#10

8

MOV R1,R0

9

ADD R0,R1,R2

10

STR R0,[R4]

11

LDR R3,[R4]

12

MOV R0,R1, LSL#1

13

STR R0,[R4,#4]

14

LDR R3,[R4,#4]!

15

STMIA R4!,{R0-R3}

16

LDMIA R4!,{R5,R6,R7,R8}

17

STMFD R13!,{R5,R6,R7,R8}

18

LDMFD R13!,{R1-R4}

19

B START

20

21

END

ARM7TDMI - Memory

Start addr: 0x90010

Tab1 - Hex - No prefix

Tab2 - Hex - No prefix

Tab3 - Hex - No prefix

Tab4 - Hex - No prefix

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	ASCII
0x00090010	0A	F1	00	00	00	E2	01	00	00	F1	00	00	0A	00	00	00
0x00090020	00	E2	01	00	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x00090030	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x00090040	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x00090050	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x00090060	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x00090070	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x00090080	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x00090090	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x000900A0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x000900B0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x000900C0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8
0x000900D0	10	00	FF	E7	00	E8	00	E8	10	00	FF	E7	00	E8	00	E8

2 关注相关寄存器的变化，实现工作模式切换

2.1 回答问题

- 执行到指令 `MOV R0, #0x000001F0` 处理器处于什么模式？

在执行到 `MOV R0, #0x000001F0` 之前的程序并没有显式地更改处理器模式，因此，处理器此时处于 **SVC (Supervisor) 模式**。这是因为当程序开始执行时，通常默认情况下处理器处于SVC模式，特别是在处理复位异常（RESET）的情况下。

- 执行到指令 `MOV R13, #3` 处理器处于什么模式？

处理器处于 **System 模式**。

2.2 分别修改程序，实现如下功能

- 由管理模式SVC mode，切换为System mode，再切换为用户模式。
- 由管理模式SVC mode，切换为FIQ mode，然后切换为用户模式。进入用户模式后，继续尝试可否用类似的方法切换为IRQ mode。
- 提交修改部分的程序代码，观察实际切换的结果和相关寄存器的变化，对结果加以解释和说明。

1	; 由管理模式SVC mode，切换为System mode，再切换为用户模式。	
2	RESET_HANDLER	
3	MOV	R0, #0x000001F0
4	LDR	R1, =0xF0000000
5	STR	R0, [R1]
6		
7	; 进入管理模式 (SVC mode)	
8	MRS	R0, CPSR
9	MOV	R13, #1

```

10      MOV     R14, #2
11
12      ; 切换到 System 模式
13      MRS     R0, CPSR
14      BIC     R0, R0, #0x1F      ; 清除模式位
15      ORR     R0, R0, #0x1F      ; 设置为 System 模式
16      MSR     CPSR_c, R0        ; 更新 CPSR, 切换到 System 模式
17
18      ; 在 System 模式下操作
19      MOV     R13, #3
20      MOV     R14, #4
21
22      ; 切换到用户模式 (User mode)
23      MRS     R0, CPSR
24      BIC     R0, R0, #0x1F      ; 清除模式位
25      ORR     R0, R0, #0x10      ; 设置为用户模式
26      MSR     CPSR_c, R0        ; 更新 CPSR, 切换到用户模式
27
28      ; 在用户模式下执行
29      MOV     R13, #5
30      MOV     R14, #6

```

切换到 System 模式：

- 在执行 `BIC R0, R0, #0x1F` 和 `ORR R0, R0, #0x1F` 后，处理器成功切换到 System 模式。
- System 模式与 SVC 模式共用大多数寄存器（R0-R12、R15、CPSR 等），但是使用了不同的栈指针（R13）和链接寄存器（R14）。切换到 System 模式后，寄存器 R13 和 R14 被重新赋值。

切换到用户模式：

- 切换到 System 模式后，R13 和 R14 指向 System 模式下独立的栈指针和链接寄存器。切换到用户模式后，R13 和 R14 再次更新，说明模式切换成功。

Current	{...}		
r0	0x000000D3	23	
r1	0xF0000000	24	; 进入管理模式 (SVC mode)
r2	0x00000000	25	MRS R0, CPSR
r3	0x00000000	26	MOV R13, #1
r4	0x00000000	27	MOV R14, #2
r5	0x00000000	28	
r6	0x00000000	29	; 切换到 System 模式
r7	0x00000000	30	MRS R0, CPSR
r8	0x00000000	31	BIC R0, R0, #0x1F ; 清除
r9	0x00000000	32	ORR R0, R0, #0x1F ; 设置
r10	0x00000000	33	MSR CPSR_c, R0 ; 更新
r11	0x00000000	34	
r12	0x00000000	35	; 在 System 模式下操作
r13	0x00000001	36	MOV R13, #3
r14	0x00000002	37	MOV R14, #4
pc	0x0000403C	38	
cpsr	nzcqvIfT_SVC	39	; 切换到用户模式 (User mode)
spsr	nzcqvift_Res	40	MRS R0, CPSR
		41	BIC R0, R0, #0x1F ; 清除
		42	ORR R0, R0, #0x10 ; 设置
		43	MSR CPSR_c, R0 ; 更新

ARM7TDMI - Registers	
Register	Value
Current	{...}
-r0	0x000000D0
-r1	0xF0000000
-r2	0x00000000
-r3	0x00000000
-r4	0x00000000
-r5	0x00000000
-r6	0x00000000
-r7	0x00000000
-r8	0x00000000
-r9	0x00000000
-r10	0x00000000
-r11	0x00000000
-r12	0x00000000
-r13	0x00000005
-r14	0x00000006
-pc	0x0000406C
-cpsr	nzcvcqIfT_User
-spsr	Unavailable
User/System	{...}
FIQ	{...}
IRQ	{...}
SVC	{...}
Abort	{...}

```

20      MOV     R0, #0x000001F0
21      LDR     R1, =0xF0000000
22      STR     R0, [R1]
23
24      ; 进入管理模式 (SVC mode)
25      MRS     R0, CPSR
26      MOV     R13, #1
27      MOV     R14, #2
28
29      ; 切换到 System 模式
30      MRS     R0, CPSR
31      BIC     R0, R0, #0x1F      ; 清除模式位
32      ORR     R0, R0, #0x1F      ; 设置为 System 模式
33      MSR     CPSR_c, R0        ; 更新 CPSR, 切换到 System 模式
34
35      ; 在 System 模式下操作
36      MOV     R13, #3
37      MOV     R14, #4
38
39      ; 切换到用户模式 (User mode)
40      MRS     R0, CPSR
41      BIC     R0, R0, #0x1F      ; 清除模式位
42      ORR     R0, R0, #0x10      ; 设置为用户模式
43      MSR     CPSR_c, R0        ; 更新 CPSR, 切换到用户模式
44
45      ; 在用户模式下执行
46      MOV     R13, #5
47      MOV     R14, #6
48
49      STOP    B
50      FND

```

```

1      ; 管理模式SVC mode, 切换为FIQ mode, 然后切换为用户模式
2      RESET_HANDLER
3          MOV     R0, #0x000001F0
4          LDR     R1, =0xF0000000
5          STR     R0, [R1]
6
7          ; 进入管理模式 (SVC mode)
8          MRS     R0, CPSR
9          MOV     R13, #1
10         MOV     R14, #2
11
12         ; 切换到 FIQ 模式
13         MRS     R0, CPSR
14         BIC     R0, R0, #0x1F      ; 清除模式位
15         ORR     R0, R0, #0x11      ; 设置为 FIQ 模式
16         MSR     CPSR_c, R0        ; 更新 CPSR, 切换到 FIQ 模式
17
18         ; 在 FIQ 模式下操作
19         MOV     R13, #3
20         MOV     R14, #4
21
22         ; 切换到用户模式 (User mode)
23         MRS     R0, CPSR
24         BIC     R0, R0, #0x1F      ; 清除模式位
25         ORR     R0, R0, #0x10      ; 设置为用户模式
26         MSR     CPSR_c, R0        ; 更新 CPSR, 切换到用户模式
27
28         ; 在用户模式下执行
29         MOV     R13, #5
30         MOV     R14, #6

```

切换到 FIQ 模式:

- FIQ 模式下, R13 和 R14 的值分别变为 3 和 4。
- FIQ 模式的寄存器使用与 SVC 模式不同, 它有自己的 R8-R14 寄存器, 允许更快地处理紧急任务。切换到 FIQ 模式后, R13 和 R14 被重新赋值, 说明切换成功。

切换到用户模式:

- 处理器成功切换到用户模式后，R13 和 R14 被重新赋值为 5 和 6，说明模式切换和寄存器更新是独立进行的。
- FIQ 模式下使用专用的 R13 和 R14，在切换回用户模式后，这些寄存器又被重新赋值。这表明 ARM 处理器不同模式下有独立的寄存器组，模式切换不会影响其他模式的寄存器内容。

ARM7TDMI - Registers		
Register	Value	
Current	{...}	
r0	0x000001F0	20 MOV R0, #0x000001F0
r1	0xF0000000	21 LDR R1, =0xF0000000
r2	0x00000000	22 STR R0, [R1]
r3	0x00000000	23
r4	0x00000000	24 ; 进入管理模式 (SVC mode)
r5	0x00000000	25 MRS R0, CPSR
r6	0x00000000	26 MOV R13, #1
r7	0x00000000	27 MOV R14, #2
r8	0x00000000	28
r9	0x00000000	29 ; 切换到 FIQ 模式
r10	0x00000000	30 MRS R0, CPSR
r11	0x00000000	31 BIC R0, R0, #0x1F ; 清除模式位
r12	0x00000000	32 ORR R0, R0, #0x11 ; 设置为 FIQ 模式
r13	0x00000000	33 MSR CPSR_c, R0 ; 更新 CPSR, 切换到 FIQ 模式
r14	0x00000000	34
pc	0x00004030	35 ; 在 FIQ 模式下操作
cpsr	nzcvqIfT_SVC	36 MOV R13, #3
spcr	nzcvqIfT_Res	37 MOV R14, #4
User/System	{...}	38
FIQ	{...}	39 ; 切换到用户模式 (User mode)
IRQ	{...}	40 MRS R0, CPSR
SVC	{...}	41 BIC R0, R0, #0x1F ; 清除模式位
		42 ORR R0, R0, #0x10 ; 设置为用户模式
		43 MSR CPSR_c, R0 ; 更新 CPSR, 切换到用户模式
		44
		45 ; 在用户模式下执行
		46 MOV R13, #5
		47 MOV R14, #6
		48 STOP

ARM7TDMI - Registers		
Register	Value	
Current	{...}	
r0	0x000000D3	19 RESET_HANDLER
r1	0xF0000000	20 MOV R0, #0x000001F0
r2	0x00000000	21 LDR R1, =0xF0000000
r3	0x00000000	22 STR R0, [R1]
r4	0x00000000	23
r5	0x00000000	24 ; 进入管理模式 (SVC mode)
r6	0x00000000	25 MRS R0, CPSR
r7	0x00000000	26 MOV R13, #1
r8	0x00000000	27 MOV R14, #2
r9	0x00000000	28
r10	0x00000000	29 ; 切换到 FIQ 模式
r11	0x00000000	30 MRS R0, CPSR
r12	0x00000000	31 BIC R0, R0, #0x1F ; 清除模式位
r13	0x00000001	32 ORR R0, R0, #0x11 ; 设置为 FIQ 模式
r14	0x00000002	33 MSR CPSR_c, R0 ; 更新 CPSR, 切换到 FIQ 模式
pc	0x0000403C	34
cpsr	nzcvqIfT_SVC	35 ; 在 FIQ 模式下操作
spcr	nzcvqIfT_Res	36 MOV R13, #3
User/System	{...}	37 MOV R14, #4
		38
		39 ; 切换到用户模式 (User mode)
		40 MRS R0, CPSR
		41 BIC R0, R0, #0x1F ; 清除模式位
		42 ORR R0, R0, #0x10 ; 设置为用户模式
		43 MSR CPSR_c, R0 ; 更新 CPSR, 切换到用户模式
		44
		45 ; 在用户模式下执行

ARM7TDMI - Registers		
Register	Value	
Current	{...}	
r0	0x000000D1	20 MOV R0, #0x000001F0
r1	0xF0000000	21 LDR R1, =0xF0000000
r2	0x00000000	22 STR R0, [R1]
r3	0x00000000	23
r4	0x00000000	24 ; 进入管理模式 (SVC mode)
r5	0x00000000	25 MRS R0, CPSR
r6	0x00000000	26 MOV R13, #1
r7	0x00000000	27 MOV R14, #2
r8	0x00000000	28
r9	0x00000000	29 ; 切换到 FIQ 模式
r10	0x00000000	30 MRS R0, CPSR
r11	0x00000000	31 BIC R0, R0, #0x1F ; 清除模式位
r12	0x00000000	32 ORR R0, R0, #0x11 ; 设置为 FIQ 模式
r13	0x00000000	33 MSR CPSR_c, R0 ; 更新 CPSR, 切换到 FIQ 模式
r14	0x00000000	34
pc	0x0000404C	35 ; 在 FIQ 模式下操作
cpsr	nzcvqIfT_FIQ	36 MOV R13, #3
spcr	nzcvqIfT_Res	37 MOV R14, #4
User/System	{...}	38
		39 ; 切换到用户模式 (User mode)
		40 MRS R0, CPSR
		41 BIC R0, R0, #0x1F ; 清除模式位
		42 ORR R0, R0, #0x10 ; 设置为用户模式
		43 MSR CPSR_c, R0 ; 更新 CPSR, 切换到用户模式
		44
		45 ; 在用户模式下执行

ARM7TDMI - Registers			
Register	Value		
Current	{...}	21	LDR R1, #0xF0000000
r0	0x000000D0	22	STR R0, [R1]
r1	0xF0000000	23	
r2	0x00000000	24	; 进入管理模式 (SVC mode)
r3	0x00000000	25	MRS R0, CPSR
r4	0x00000000	26	MOV R13, #1
r5	0x00000000	27	MOV R14, #2
r6	0x00000000	28	
r7	0x00000000	29	; 切换到 FIQ 模式
r8	0x00000000	30	MRS R0, CPSR
r9	0x00000000	31	BIC R0, R0, #0x1F ; 清除模式位
r10	0x00000000	32	ORR R0, R0, #0x11 ; 设置为 FIQ 模式
r11	0x00000000	33	MSR CPSR_c, R0 ; 更新 CPSR, 切换到 FIQ 模式
r12	0x00000000	34	
r13	0x00000005	35	; 在 FIQ 模式下操作
r14	0x00000006	36	MOV R13, #3
pc	0x0000406C	37	MOV R14, #4
cpsr	nzcvqIfT_User	38	
spsr	Unavailable	39	; 切换到用户模式 (User mode)
User/System	{...}	40	MRS R0, CPSR
FIQ	{...}	41	BIC R0, R0, #0x1F ; 清除模式位
IRQ	{...}	42	ORR R0, R0, #0x10 ; 设置为用户模式
SVC	{...}	43	MSR CPSR_c, R0 ; 更新 CPSR, 切换到用户模式
		44	
		45	; 在用户模式下执行
		46	MOV R13, #5
		47	MOV R14, #6
		48	STOP

```

1 ; 管理模式 (SVC mode) 切换到FIQ模式, 再切换到用户模式, 在用户模式下继续尝试切换为IRQ模式
2 RESET_HANDLER
3     MOV     R0, #0x000001F0
4     LDR     R1, =0xF0000000
5     STR     R0, [R1]
6
7     ; 进入管理模式 (SVC mode)
8     MRS     R0, CPSR
9     MOV     R13, #1
10    MOV     R14, #2
11
12    ; 切换到 FIQ 模式
13    MRS     R0, CPSR
14    BIC     R0, R0, #0x1F ; 清除模式位
15    ORR     R0, R0, #0x11 ; 设置为 FIQ 模式
16    MSR     CPSR_c, R0 ; 更新 CPSR, 切换到 FIQ 模式
17
18    ; 在 FIQ 模式下操作
19    MOV     R13, #3
20    MOV     R14, #4
21
22    ; 切换到用户模式 (User mode)
23    MRS     R0, CPSR
24    BIC     R0, R0, #0x1F ; 清除模式位
25    ORR     R0, R0, #0x10 ; 设置为用户模式
26    MSR     CPSR_c, R0 ; 更新 CPSR, 切换到用户模式
27
28    ; 在用户模式下操作
29    MOV     R13, #5
30    MOV     R14, #6
31
32    ; 尝试从用户模式切换到 IRQ 模式
33    MRS     R0, CPSR
34    BIC     R0, R0, #0x1F ; 清除模式位
35    ORR     R0, R0, #0x12 ; 设置为 IRQ 模式
36    MSR     CPSR_c, R0 ; 更新 CPSR, 切换到 IRQ 模式
37
38    ; 在 IRQ 模式下操作
39    MOV     R13, #7
40    MOV     R14, #8

```

从用户模式切换到 IRQ 模式：

- 在用户模式下，处理器通过直接操作 `CPSR` 寄存器成功切换到 IRQ 模式，R13 和 R14 的值变为 7 和 8。
- IRQ 模式有自己的 R13 和 R14，切换成功后寄存器值发生改变。
- 用户模式下的切换是可行的，因为处理器模式的切换依赖于对 `CPSR` 寄存器的操作。通过 `MSR` 指令修改 `CPSR`，处理器可以从用户模式切换到 IRQ 模式，且寄存器内容独立于其他模式。

2.3 在程序中MOV R14, #4语句后，增加语句

```
1  ADD PC, PC, #4
2  ADD R0, R0, #0x1
3  ADD R0, R0, #0x2
4  ADD R0, R0, #0x3
5  ADD R0, R0, #0x4
```

通过分析程序执行的结果，分析在ARM流水线下相对寻址指令（以程序计数器PC的当前值为基地址，偏移量相加得到有效地址）的执行过程：

ARM使用三阶段流水线架构，分别为**取指令**、**译码**、**执行**。在执行过程中，由于流水线的原因，PC的值与正在执行的指令地址存在一个常量偏移。

```
1  ADD PC, PC, #4
```

由于**PC在ARM处理器中指向当前指令的地址加上8**，因为取指令时PC会提前两条指令。因此：

- 当PC指向一条指令时，实际执行的指令位于当前PC+8的地址。
- 因此，当执行 `ADD PC, PC, #4` 时，PC的值将会增加4字节，相当于让程序跳过下一条指令，执行后续的指令。

因此，增加语句中：

```
1  ADD PC, PC, #4      ; PC = PC + 4
2  ADD R0, R0, #1      ; 不执行，因PC被跳过
3  ADD R0, R0, #2      ; 执行
4  ADD R0, R0, #3      ; 执行
5  ADD R0, R0, #4      ; 执行
```

2.4 程序中，使用了LDR伪指令，请分析该伪指令和LDR加载指令在功能和应用上有何区别

2.4.1 LDR伪指令

- **功能：** `LDR伪指令` 通常用于从内存中加载一个立即数（常量）到寄存器中。当常量的值无法直接通过立即数的形式嵌入到指令中时（即超出了指令可以表示的范围），编译器会在指令的后续部分存储该常量，并生成加载该常量的指令。
 - 在程序的某个位置存储常量值。
 - 使用 `LDR` 加载指令从该位置读取常量值到寄存器中。

举一个例子如下：


```
1 | LDR R0, =0xF0000000 ; LDR伪指令
```

伪指令 `LDR R0, =0xF0000000` 将常量 `0xF0000000` 加载到寄存器 `R0`。编译器会在程序的某个地方存储 `0xF0000000`，并生成对应的指令从该位置加载值到 `R0` 中。

- 展开后的指令是这样的：

```
1 | LDR R0, [PC, #offset] ; 从相对于PC的偏移位置加载常量到R0
```

2.4.2 LDR加载指令

- 功能：** `LDR加载指令` 是ARM指令集中用于从内存地址中加载数据到寄存器中的指令。它可以使用基址加偏移（或基址加寄存器）来从内存中读取一个32位值到寄存器。
 - 该指令直接从内存中读取指定地址上的数据，并将其加载到寄存器中。

例子：

```
1 | LDR R0, [R1] ; LDR加载指令
```

`LDR R0, [R1]` 将内存地址 `R1` 指向的内容加载到寄存器 `R0` 中。这是直接的内存访问操作。

那我们总结如下

特性	LDR伪指令	LDR加载指令
用途	用于加载无法通过立即数表示的常量	用于从内存地址中加载数据
加载的值	常量值（通常是立即数）	内存地址指向的数据
常见应用	当常量过大，不能直接用立即数表示	内存数据访问、加载变量的值
实现方式	编译器伪指令，会存储常量并生成指令	直接的指令，访问内存地址
地址来源	PC指向的常量池中的数据	任意内存地址

3 按求最大公约数.doc的说明编写汇编指令

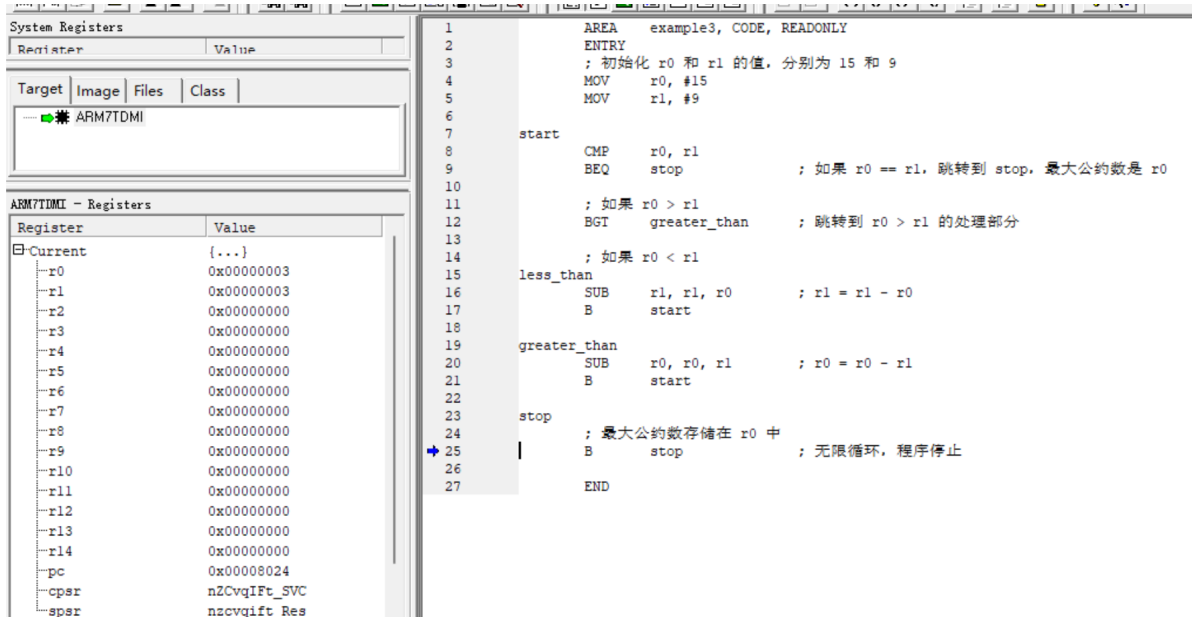
AXD窗口截图，图中包含程序代码和运行完成后的寄存器内容。

```
1 | AREA example3, CODE, READONLY
2 | ENTRY
3 | ; 初始化 r0 和 r1 的值，分别为 15 和 9
4 | MOV r0, #15
5 | MOV r1, #9
6 |
7 | start
8 | CMP r0, r1
9 | BEQ stop ; 如果 r0 == r1, 跳转到 stop, 最大公约数是 r0
10 |
11 | ; 如果 r0 > r1
12 | BGT greater_than ; 跳转到 r0 > r1 的处理部分
13 |
14 | ; 如果 r0 < r1
15 | less_than
```

```

16      SUB    r1, r1, r0      ; r1 = r1 - r0
17      B      start
18
19  greater_than
20      SUB    r0, r0, r1      ; r0 = r0 - r1
21      B      start
22
23  stop
24      ; 最大公约数存储在 r0 中
25      B      stop           ; 无限循环，程序停止
26
27      END

```



The screenshot shows the ARM7TDMI debugger interface. On the left, the 'System Registers' window displays the 'ARM7TDMI' target. Below it, the 'Registers' window shows the current values of the registers. On the right, the assembly code is displayed with line numbers 1 through 27. The code is for a program to find the Greatest Common Divisor (GCD) of two numbers, 15 and 9.

System Registers

Register	Value
ARM7TDMI	

Registers

Register	Value
r0	0x00000003
r1	0x00000003
r2	0x00000000
r3	0x00000000
r4	0x00000000
r5	0x00000000
r6	0x00000000
r7	0x00000000
r8	0x00000000
r9	0x00000000
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13	0x00000000
r14	0x00000000
pc	0x00008024
cpsr	n2CvqIFt_SVC
sp	nscvqift_Res

Assembly Code

```

1  AREA    example3, CODE, READONLY
2  ENTRY
3  ; 初始化 r0 和 r1 的值，分别为 15 和 9
4  MOV     r0, #15
5  MOV     r1, #9
6
7  start
8  CMP     r0, r1
9  BEQ     stop      ; 如果 r0 == r1. 跳转到 stop, 最大公约数是 r0
10
11 ; 如果 r0 > r1
12 BGT     greater_than ; 跳转到 r0 > r1 的处理部分
13
14 ; 如果 r0 < r1
15 less_than
16 SUB     r1, r1, r0      ; r1 = r1 - r0
17 B      start
18
19 greater_than
20 SUB     r0, r0, r1      ; r0 = r0 - r1
21 B      start
22
23 stop
24 ; 最大公约数存储在 r0 中
25 B      stop           ; 无限循环，程序停止
26
27 END

```