第 51 行是 GIC 的 CPU 接口端相关寄存器，其相对于 GIC 基地址的偏移为 0X2000，同样的，获取到 GIC 基地址以后只需要加上 0X2000 即可访问 GIC 的 CPU 接口段寄存器。
那么问题来了？GIC 控制器的寄存器基地址在哪里呢？这个就需要用到 Cortex-A 的 CP15 协处理器了，下一小节就讲解 CP15 协处理器。

17.1.4 CP15 协处理器

关于 CP15 协处理器和其相关寄存器的详细内容请参考下面两份文档：《ARM ArchitectureReference Manual ARMv7-A and ARMv7-R edition.pdf》第 1469 页“B3.17 Oranization of the CP15 registers in a VMSA implementation”。《Cortex-A7 Technical ReferenceManua.pdf》第 55 页“Capter 4 System Control”。

CP15 协处理器一般用于存储系统管理，但是在中断中也会使用到，CP15 协处理器一共有 16 个 32 位寄存器。CP15 协处理器的访问通过如下另个指令完成：

MRC: 将 CP15 协处理器中的寄存器数据读到 ARM 寄存器中。

MCR: 将 ARM 寄存器的数据写入到 CP15 协处理器寄存器中。

MRC 就是读 CP15 寄存器，MCR 就是写 CP15 寄存器，MCR 指令格式如下：

```
MCR{cond} p15, <opc1>, <Rt>, <CRn>, <CRm>, <opc2>
```

cond:指令执行的条件码，如果忽略的话就表示无条件执行。

opc1：协处理器要执行的操作码。

Rt：ARM 源寄存器，要写入到 CP15 寄存器的数据就保存在此寄存器中。

CRn：CP15 协处理器的目标寄存器。

CRm：协处理器中附加的目标寄存器或者源操作数寄存器，如果不需要附加信息就将 CRm 设置为 C0，否则结果不可预测。

opc2：可选的协处理器特定操作码，当不需要的时候要设置为 0。

MRC 的指令格式和 MCR 一样，只不过在 MRC 指令中 Rt 就是目标寄存器，也就是从 CP15 指定寄存器读出来的数据会保存在 Rt 中。而 CRn 就是源寄存器，也就是要读取的写处理器寄存器。

假如我们要将 CP15 中 C0 寄存器的值读取到 R0 寄存器中，那么就可以使用如下命令：

```
MRC p15, 0, r0, c0, c0, 0
```

CP15 协处理器有 16 个 32 位寄存器，c0~c15，本章来看一下 c0、c1、c12 和 c15 这四个寄存器，因为我们本章实验要用到这四个寄存器，其他的寄存器大家参考上面的两个文档即可。

1、c0 寄存器

CP15 协处理器有 16 个 32 位寄存器，c0~c15，在使用 MRC 或者 MCR 指令访问这 16 个寄存器的时候，指令中的 CRn、opc1、CRm 和 opc2 通过不同的搭配，其得到的寄存器含义是不同的。比如 c0 在不同的搭配情况下含义如图 17.1.4.1 所示：

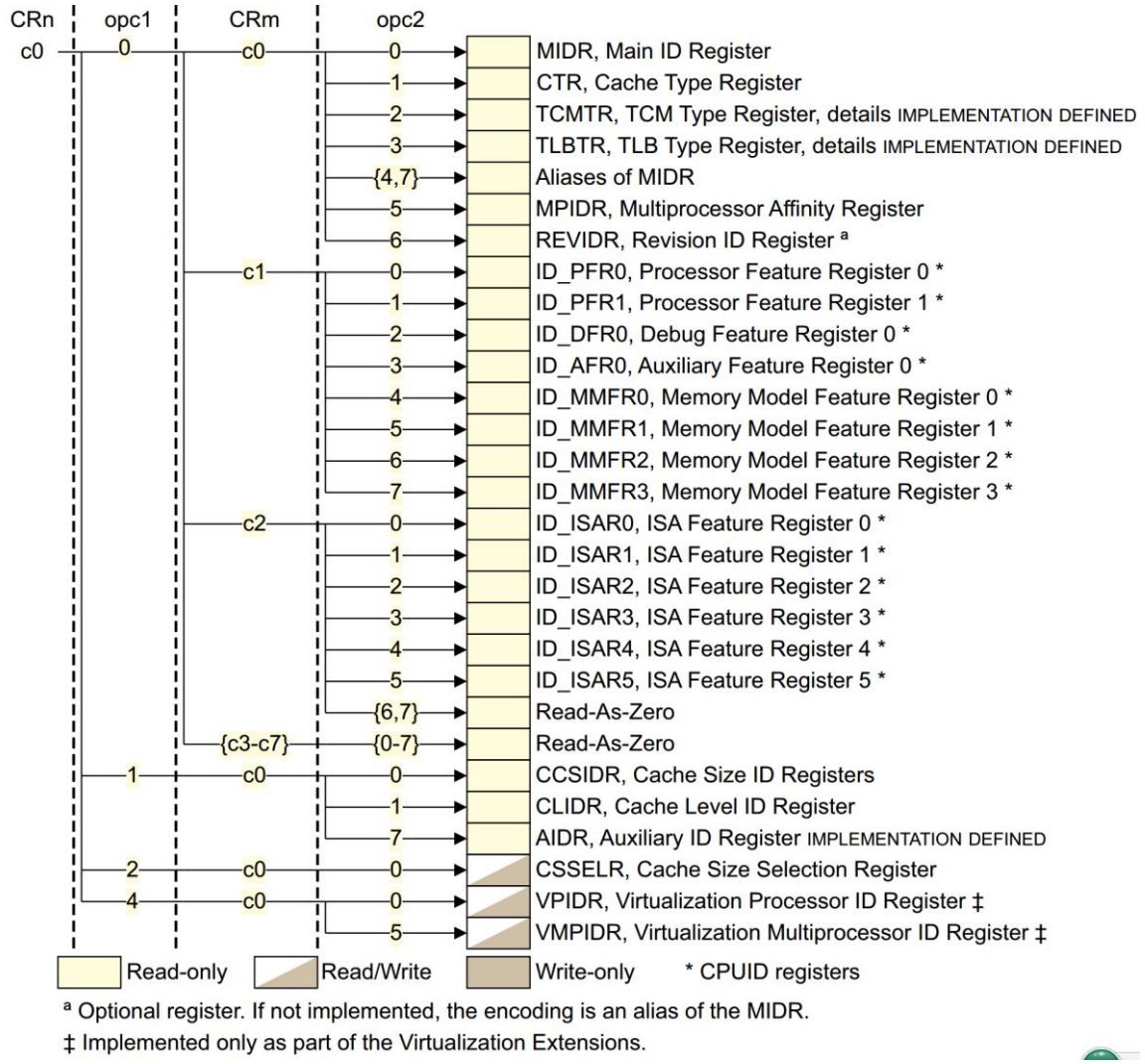

图 17.1.4.1 c0 寄存器不同搭配含义

在图 17.1.4.1 中当 MRC/MCR 指令中的 CRn=c0，opc1=0，CRm=c0，opc2=0 的时候就表示此时的 c0 就是 MIDR 寄存器，也就是主 ID 寄存器，这个也是 c0 的基本作用。对于 Cortex-A7 内核来说，c0 作为 MDIR 寄存器的时候其含义如图 17.1.4.2 所示：

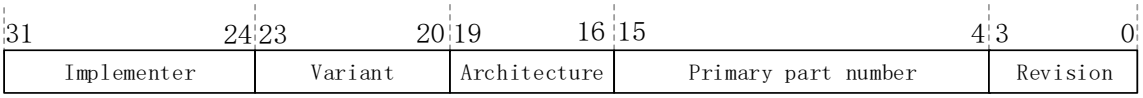

图 17.1.4.2 c0 作为 MIDR 寄存器结构图

在图 17.1.4.2 中各位所代表的含义如下：

bit31:24：厂商编号，0X41，ARM。

bit23:20：内核架构的主版本号，ARM 内核版本一般使用 rnpn 来表示，比如 r0p1，其中 r0 后面的 0 就是内核架构主版本号。

bit19:16：架构代码，0XF，ARMv7 架构。

bit15:4：内核版本号，0XC07，Cortex-A7 MPCore 内核。

bit3:0：内核架构的次版本号，rnpn 中的 pn，比如 r0p1 中 p1 后面的 1 就是次版本号。

2、c1 寄存器

c1 寄存器同样通过不同的配置，其代表的含义也不同，如图 17.1.4.3 所示：

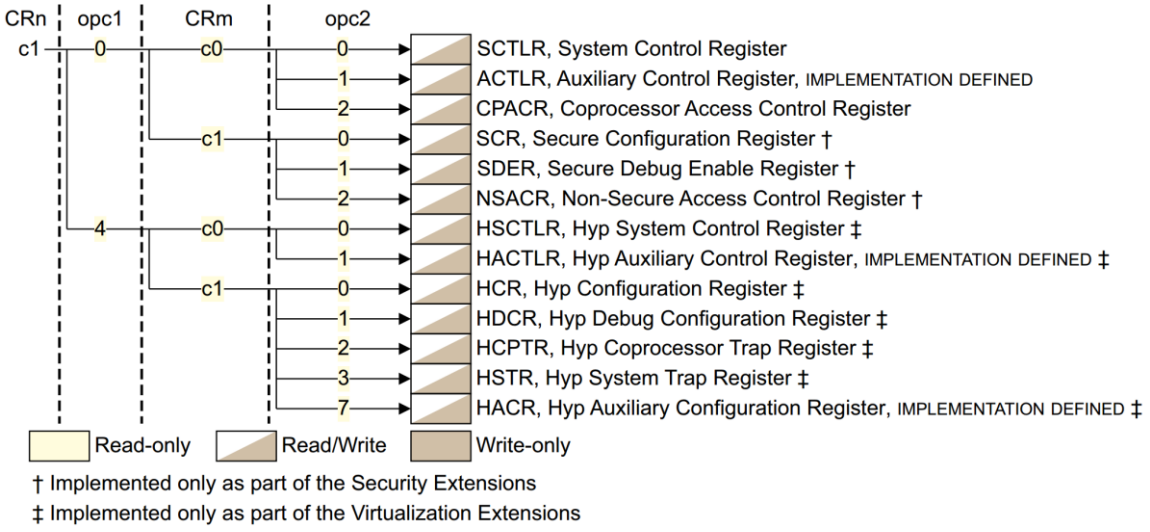

图 17.1.4.3 c1 寄存器不同搭配含义

在图 17.1.4.3 中当 MRC/MCR 指令中的 CRn=c1，opc1=0，CRm=c0，opc2=0 的时候就表示此时的 c1 就是 SCTLR 寄存器，也就是系统控制寄存器，这个是 c1 的基本作用。SCTLR 寄存器主要是完成控制功能的，比如使能或者禁止 MMU、I/D Cache 等，c1 作为 SCTLR 寄存器的时候其含义如图 17.1.4.4 所示：

30	31	29	28	27 26	25	24 21	20	19	18 14	13	12	11	10	9 3	2	1	0
Res	TE	AFE	TRE	Res	EE	Res	UWXN	WXN	Res	V	I	Z	SW	Res	C	A	M

图 17.1.4.4 c1 作为 SCTLR 寄存器结构图

SCTLR 的位比较多，我们就只看本章会用到的几个位：

bit13：V，中断向量表基地址选择位，为 0 的话中断向量表基地址为 0X00000000，软件可以使用 VBAR 来重映射此基地址，也就是中断向量表重定位。为 1 的话中断向量表基地址为 0XFFFF0000，此基地址不能被重映射。

bit12：I，I Cache 使能位，为 0 的话关闭 I Cache，为 1 的话使能 I Cache。

bit11：Z，分支预测使能位，如果开启 MMU 的话，此位也会使能。

bit10：SW，SWP 和 SWPB 使能位，当为 0 的话关闭 SWP 和 SWPB 指令，当为 1 的时候就使能 SWP 和 SWPB 指令。

bit9:3：未使用，保留。

bit2：C，D Cache 和缓存一致性使能位，为 0 的时候禁止 D Cache 和缓存一致性，为 1 时使能。

bit1：A，内存对齐检查使能位，为 0 的时候关闭内存对齐检查，为 1 的时候使能内存对齐检查。

bit0：M，MMU 使能位，为 0 的时候禁止 MMU，为 1 的时候使能 MMU。

如果要读写 SCTLR 的话，就可以使用如下命令：

```
MRC p15, 0, <Rt>, c1, c0, 0   ;读取 SCTLR 寄存器，数据保存到 Rt 中。
MCR p15, 0, <Rt>, c1, c0, 0   ;将 Rt 中的数据写到 SCTLR(c1)寄存器中。
```

2、c12 寄存器

c12 寄存器通过不同的配置，其代表的含义也不同，如图 17.1.4.4 所示：

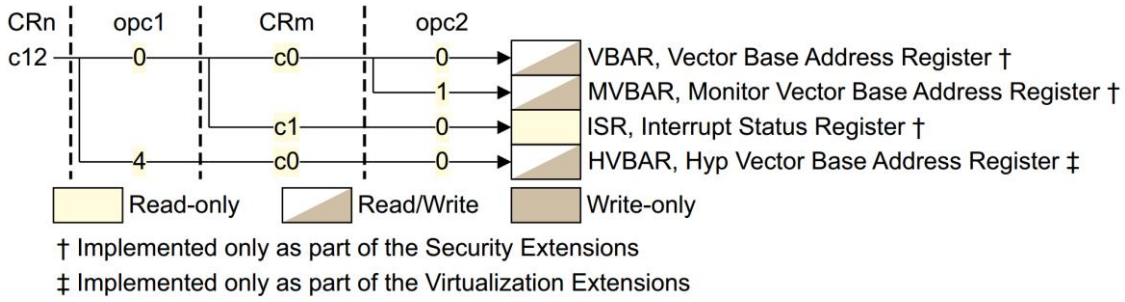

图 17.1.4.4 c12 寄存器不同搭配含义

在图 17.1.4.4 中当 MRC/MCR 指令中的 CRn=c12，opc1=0，CRm=c0，opc2=0 的时候就表示此时 c12 为 VBAR 寄存器，也就是向量表基地址寄存器。设置中断向量表偏移的时候就需要将新的中断向量表基地址写入 VBAR 中，比如在前面的例程中，代码链接的起始地址为 0X87800000，而中断向量表肯定要放到最前面，也就是 0X87800000 这个地址处。所以就需要设置 VBAR 为 0X87800000，设置命令如下：

```
ldr r0, =0X87800000          ; r0=0X87800000
MCR p15, 0, r0, c12, c0, 0   ;将 r0 里面的数据写入到 c12 中，即 c12=0X87800000
```

3、c15 寄存器

c15 寄存器也可以通过不同的配置得到不同的含义，参考文档《Cortex-A7 Technical ReferenceManua.pdf》第 68 页“4.2.16 c15 registers”，其配置如图 17.1.4.5 所示：

CRn	Op1	CRm	Op2	Name	Reset	Description
c15	3[a]	c0	0	CDBGDR0	UNK	Data Register 0, see *Direct access to internal memory* on page 6-9
			1	CDBGDR1	UNK	Data Register 1, see *Direct access to internal memory* on page 6-9
			2	CDBGDR2	UNK	Data Register 2, see *Direct access to internal memory* on page 6-9
		c2	0	CDBGDCT	UNK	Data Cache Tag Read Operation Register, see *Direct access to internal memory* on page 6-9
			1	CDBGICT	UNK	Instruction Cache Tag Read Operation Register, see *Direct access to internal memory* on page 6-9
		c4	0	CDBGDCD	UNK	Data Cache Data Read Operation Register, see *Direct access to internal memory* on page 6-9
			1	CDBGICD	UNK	Instruction Cache Data Read Operation Register, see *Direct access to internal memory* on page 6-9
			2	CDBGTD	UNK	TLB Data Read Operation Register, see *Direct access to internal memory* on page 6-9
	4	c0	0	CBAR	-[b]	*Configuration Base Address Register* on page 4-83

图 17.1.4.5 c15 寄存器不同搭配含义

在图 17.1.4.5 中，我们需要 c15 作为 CBAR 寄存器，因为 GIC 的基地址就保存在 CBAR 中，我们可以通过如下命令获取到 GIC 基地址：

```
MRC p15, 4, r1, c15, c0, 0 ； 获取 GIC 基础地址，基地址保存在 r1 中。
```

获取到 GIC 基地址以后就可以设置 GIC 相关寄存器了，比如我们可以读取当前中断 ID，当前中断 ID 保存在 GICC_IAR 中，寄存器 GICC_IAR 属于 CPU 接口端寄存器，寄存器地址相对于 CPU 接口端起始地址的偏移为 0XC，因此获取当前中断 ID 的代码如下：

```
MRC p15, 4, r1, c15, c0, 0    ;获取 GIC 基地址
ADD r1, r1, #0X2000           ;GIC 基地址加 0X2000 得到 CPU 接口端寄存器起始地址
```

```
LDR r0, [r1, #0XC]          ;读取 CPU 接口端起始地址+0XC 处的寄存器值，也就是寄存器
                            ;GIC_IAR 的值
```

关于 CP15 协处理器就讲解到这里，简单总结一下，通过 c0 寄存器可以获取到处理器内核信息；通过 c1 寄存器可以使能或禁止 MMU、I/D Cache 等；通过 c12 寄存器可以设置中断向量偏移；通过 c15 寄存器可以获取 GIC 基地址。关于 CP15 的其他寄存器，大家自行查阅本节前面列举的 2 份 ARM 官方资料。

17.1.5 中断使能

中断使能包括两部分，一个是 IRQ 或者 FIQ 总中断使能，另一个就是 ID0~ID1019 这 1020 个中断源的使能。

1、IRQ 和 FIQ 总中断使能

IRQ 和 FIQ 分别是外部中断和快速中断的总开关，就类似家里买的进户总电闸，然后 ID0~ID1019 这 1020 个中断源就类似家里面的各个电器开关。要想开电视，那肯定要保证进户总电闸是打开的，因此要想使用 I.MX6U 上的外设中断就必须先打开 IRQ 中断(本教程不使用 FIQ)。在“6.3.2 程序状态寄存器”小节已经讲过了，寄存器 CPSR 的 I=1 禁止 IRQ，当 I=0 使能 IRQ；F=1 禁止 FIQ，F=0 使能 FIQ。我们还有更简单的指令来完成 IRQ 或者 FIQ 的使能和禁止，图表 17.1.5.1 所示：

指令	描述
cpsid i	禁止 IRQ 中断。
cpsie i	使能 IRQ 中断。
cpsid f	禁止 FIQ 中断。
cpsie f	使能 FIQ 中断。

表 17.1.5.1 开关中断指令

2、ID0~ID1019 中断使能和禁止

GIC 寄存器 GICD_ISENABLERn 和 GICD_ ICENABLERn 用来完成外部中断的使能和禁止，对于 Cortex-A7 内核来说中断 ID 只使用了 512 个。一个 bit 控制一个中断 ID 的使能，那么就需要 512/32=16 个 GICD_ISENABLER 寄存器来完成中断的使能。同理，也需要 16 个 GICD_ICENABLER 寄存器来完成中断的禁止。其中 GICD_ISENABLER0 的 bit[15:0]对应 ID15~0 的 SGI 中断，GICD_ISENABLER0 的 bit[31:16]对应 ID31~16 的 PPI 中断。剩下的 GICD_ISENABLER1~GICD_ISENABLER15 就是控制 SPI 中断的。

17.1.6 中断优先级设置

1、优先级数配置

学过 STM32 都知道 Cortex-M 的中断优先级分为抢占优先级和子优先级，两者是可以配置的。同样的 Cortex-A7 的中断优先级也可以分为抢占优先级和子优先级，两者同样是可以配置的。GIC 控制器最多可以支持 256 个优先级，数字越小，优先级越高！Cortex-A7 选择了 32 个优先级。在使用中断的时候需要初始化 GICC_PMR 寄存器，此寄存器用来决定使用几级优先级，寄存器结构如图 17.1.6.1 所示：