附录A

# Cortex-M3指令小结

此附录实际上是从Cortex-M3技术参考手册中译版摘抄并改编的。并且在可能的情况下，使用类C语言的风格来讲解指令的功能。另外要解释的是

U8表示unsigned char，无符号16位整数

U16表示unsigned short，无符号16位整数

S8表示signed char，带符号8位整数

S16表示signed short，带符号16位整数

缺省情况下，如果使用普通的char和short，都是指带符号整数

当借C语言的数组表示法，如Rn[Rm]时，是按整数运算的方式求得Rn+Rm的值，然后把该值当作一个32位地址，再取出该地址的值。在计算地址时，并不乘以“数据类型所占用的字节数”，这与C语言的数组/指针运算是概念上的不同，切记切记！

简单地概括，这里的Rn[Rm]等效于

\*( (U32 \*) (Rn+Rm) )，其中Rn,Rm均为32位整数类型

还有两条重要的通用规则：

表 **1-1 16**位 **Cortex-M3**指令汇总

* **凡是在指令中有可选的预移位操作的，预移位后的值是中间结果，不写回被移位的寄存器**

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| Rd+= Rm+C | ADC <Rd>, <Rm> |
| Rd= Rn+Imm3 | ADD <Rd>, <Rn>, #<immed\_3> |
| Rd+= Imm8 | ADD <Rd>, #<immed\_8> |
| Rd=Rn+Rm | ADD <Rd>, <Rn>, <Rm> |
| Rd+=Rm | ADD <Rd>, <Rm> |
| Rd=PC+Imm8\*4 | ADD <Rd>, PC, #<immed\_8>\*4 |
| Rd=SP+Imm8\*4 | ADD <Rd>, SP, #<immed\_8>\*4 |
| Rd=SP+Imm7\*4 或  SP+=Imm7\*4 | ADD <Rd>, SP, #<immed\_7>\*4或 ADD SP, SP, #<immed\_7>\*4 |
| Rd &= Rm | AND <Rd>, <Rm> |
| Rd = Rm 算术右移 Imm5 | ASR <Rd>, <Rm>, #<immed\_5> |
| Rd 算术右移= Rs | ASR <Rd>, <Rs> |

* **凡是在{S}的指令中使用了S后缀的，都按照运算结果更新APSR中的标志位。**

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| 按<contd>条件决定是否分支 | B<cond> <target address> |
| 无条件分支 | B<tartet address> |
| Rd &= ~Rs | BIC <Rd>, <Rs> |
| 软件断点 | BKPT <immed\_8> |
| 带链接分支 | BL <Rm> |
| 比较结果不为零时分支 | CBNZ <Rn>, <label> |
| 比较结果为零时分支 | CBZ <Rn>, <Rm> |
| 将Rm取二进制补码后再与Rn比较（注意：不是取反！！！） | CMN <Rn>, <Rm> |
| Rn与 8位立即数比较 ，并根据结果更新标志位的值 | CMP <Rn>, #<immed\_8> |
| Rn与Rm比较，并根据结果更新标志位的值 | CMP <Rn>, <Rm> |
| 高寄存器与高或低寄存器比较,并根据结果更新标志位的值。在实际编程时，可以无视这两条指令的区别，当作一条指令来用。 | CMP <Rn>, <Rm> |
| 改变处理器状态 | CPS <effect>, <iflags> |
| 将高或低寄存器的值复制到另一个高或低寄存器中 | CPY <Rd>, <Rm> |
| Rd^=Rm | EOR <Rd>, <Rm> |
| 以下一条指令为条件；  以下面两条指令为条件；  以下面三条指令为条件；  以下面四条指令为条件 | IT<cond>  IT<x> <cond>  IT<x><y> <cond>  IT<x><y><z> <cond> |
| 多个连续的存储器字加载 | LDMIA <Rn>!, <register> |
| 将基址寄存器与 5位立即数偏移的和的地址处的数据加载到寄存器中  Rd= Rn[Imm5\*4] | LDR <Rd>, [<Rn>, #<immed\_5\*4>] |
| Rd= Rn[Rm] | LDR <Rd>, [<Rn>, <Rm>] |
| Rd= PC[Imm8\*4+4] | LDR <Rd>, [PC, #<immed\_8>\*4] |
| Rd= SP[Imm8\*4] | LDR <Rd>, [SP, #<immed\_8>\*4] |
| Rd= (U8) Rn[Imm5] | LDRB <Rd>, [<Rn>, #<immed\_5>] |
| Rd= (U8) Rn[Rm] | LDRB <Rd>, [<Rn>, <Rm>] |
| Rd= (U16) Rn[Imm5\*2] | LDRH <Rd>, [<Rn>, #<immed\_5>\*2] |
| Rd= (U16) Rn[Rm] | LDRH <Rd>, [<Rn>, <Rm>] |
| 加载Rn+Rm的地址处的字节,并带符号扩展到Rd中 | LDRSB <Rd>, [<Rn>, <Rm>] |
| 加载Rn+Rm的地址处的半字,并带符号扩展到Rd中 | LDRSH <Rd>, [<Rn>, <Rm>] |
| Rd= Rm<<Imm5 | LSL <Rd>, <Rm>, #<immed\_5> |
| Rd<<= Rs | LSL <Rd>, <Rs> |
| Rd= Rm>>Imm5 | LSR <Rd>, <Rm>, #<immed\_5> |
| Rd>>= Rs | LSR <Rd>, <Rs> |
| Rd= (U32) Imm8 | MOV <Rd>, #<immed\_8> |

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| Rd=Rn | MOV <Rd>, <Rn> |
| Rd=Rm。实际使用时，可把这两条MOV指令当成一条指令来用——译者注 | MOV <Rd>, <Rm> |
| Rd\*=Rm | MUL <Rd>, <Rm> |
| Rd= ~Rm （注意，是取反，不是取补码！！！） | MVN <Rd>, <Rm> |
| Rd= ~Rm + 1 | NEG <Rd>, <Rm> |
| 无操作 | NOP <C> |
| Rd|= Rm | ORR <Rd>, <Rm> |
| 寄存器出栈 | POP <寄存器> |
| 若干寄存器和 PC出栈 | POP <寄存器，PC> |
| 若干寄存器压栈 | PUSH <registers> |
| 若干寄存器和 LR压栈 | PUSH <registers, LR> |
| Rd=Rn字内的字节反转 | REV <Rd>, <Rn> |
| Rd=Rn两个半字内的字节反转 | REV16 <Rd>, <Rn> |
| 将Rn低半字内的字节反转，再把反转后的值带符号位扩展到32位后，复制到Rd中 | REVSH <Rd>, <Rn> |
| Rd 圆圈右移= Rs | ROR <Rd>, <Rs> |
| Rd-= Rm+C | SBC <Rd>, <Rm> |
| 发送事件 | SEV <c> |
| 将多个寄存器字保存到连续的存储单元中，首地址由Rn给出。每保存完一个Rn+4 | STMIA <Rn>!, <registers> |
| Rn[Imm5\*4]=Rd | STR <Rd>, [<Rn>, #<immed\_5>\*4] |
| Rn[Rm]=Rd | STR <Rd>, [<Rn>, <Rm>] |
| SP[Imm8\*4]=Rd | STR <Rd>, [SP, #<immed\_8> \* 4] |
| \*( (U8\*) (Rn+Imm5) ) = (U8) Rd | STRB <Rd>, [<Rn>, #<immed\_5>] |
| \*( (U8\*) (Rn+Rm) ) = (U8) Rd | STRB <Rd>, [<Rn>, <Rm>] |
| \*( (U16\*) (Rn+Imm5\*2) ) = (U16) Rd | STRH <Rd>, [<Rn>, #<immed\_5> \* 2] |
| \*( (U16\*) (Rn+Rm) ) = (U16) Rd | STRH <Rd>, [<Rn>, <Rm>] |
| Rd-= Imm8 | SUB <Rd>, #<immed\_8> |
| Rd= Rn-Rm | SUB <Rd>, <Rn>, <Rm> |
| SP-= Imm7\*4 | SUB SP, #<immed\_7> \* 4 |
| 操作系统服务调用，带8位立即数调用代码 | SVC <immed\_8> |
| 从寄存器中提取字节[7:0]，传送到寄存器中，并用符号位扩展到32位 | SXTB <Rd>, <Rm> |
| 从寄存器中提取半字[15:0]，传送到寄存器中，并用符号位扩展到32位 | SXTH <Rd>, <Rm> |
| 执行Rn & Rm，并根据结果更新标志位 | TST <Rn>, <Rm> |
| 从寄存器中提取字节[7:0]，传送到寄存器中，并用零位扩展到 32位  Rd= (U8) Rm | UXTB <Rd>, <Rm> |

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| 从寄存器中提取半字[15:0]，传送到寄存器中，并用零位扩展到32位  Rd= (U16) Rm | UXTH <Rd>, <Rm> |
| 等待事件 | WFE <c> |
| 等待中断 | WFI <c> |

表 1-2列出了 32位 Coxtex-M3指令。表 **1-2 32**位 **Coxtex-M3**指令汇总

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| Rd=Rn+Imm12+C。有S就按结果更新标志位。S的作用下同。 | ADC{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12> |
| Rd= Rn与移位后的Rm及C位相加 | ADC{S}.W <Rd>, <Rn>, <Rm>{, <shift>} |
| Rd= Rn+Imm12 | ADD{S}.W <Rd>, <Rn>,#<modify\_constant(immed\_12)> |
| Rd=Rd与移位后的Rm相加 | ADD{S}.W <Rd>, <Rm>{, <shift>} |
| Rd= Rn+Imm12 | ADDW.W <Rd>, <Rn>, #<immed\_12> |
| Rd= Rn & Imm12 | AND{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12> |
| Rd=Rn与移位后的Rm按位与 | AND{S}.W <Rd>, <Rn>, Rm>{, <shift>} |
| Rd = Rn>>Rm。有S就按结果更新标志位 | ASR{S}.W <Rd>, <Rn>, <Rm> |
| 条件分支 | B{cond}.W <label> |
| 位区清零 | BFC.W <Rd>, #<lsb>, #<width> |
| 将一个寄存器的位区插入另一个寄存器中 | BFI.W <Rd>, <Rn>, #<lsb>, #<width> |
| Rd= Rn & ~Imm12 | BIC{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12)> |
| Rd&= 移位后的Rn取反 | BIC{S}.W <Rd>, <Rn>, {, <shift>} |
| 带链接的分支 | BL <label> |
| 带链接的分支（立即数） | BL<c> <label> |
| 无条件分支 | B.W <label> |
| Rd=Rn中前导零的数目 | CLZ.W <Rd>, <Rn> |
| Rn与12位立即数取补后的值比较 | CMN.W <Rn>, #<modify\_constant(immed\_12)> |
| Rn与移位后的Rm取补后的值比较 | CMN.W <Rn>, <Rm>{, <shift>} |
| Rn与12位立即数比较 | CMP.W <Rn>, #<modify\_constant(immed\_12)> |
| Rn与按需移位后的Rm比较  Rm的值不变 | CMP.W <Rn>, <Rm>{, <shift>} |
| 数据存储器隔离 | DMB <c> |
| 数据同步隔离 | DSB <c> |
| Rd= Rn ^ Imm12 | EOR{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12)> |
| Rd=Rn与按需移位后的Rm作异或操作  Rm的值不变 | EOR{S}.W <Rd>, <Rn>, <Rm>{, <shift>} |
| 指令同步排序（barrier） | ISB <c> |
| 多存储器寄存器加载，加载后加4或加载前减 4 | LDM{IA|DB}.W <Rn>{!}, <registers> |
| Rxf= Rn[ofs12] | LDR.W <Rxf>, [<Rn>, #<offset\_12>] |
| PC= Rn[ofs12] | LDR.W PC, [<Rn>, #<offset\_12>] |

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| 无此指令 | LDR.W PC, #<+/-<offset\_8> |
| Rxf= \*Rn;  Rn+= ofs8; | LDR.W <Rxf>, [<Rn>], #+/–<offset\_8> |
| Rn+= ofs8;  Rxf= \*Rn | LDR.W <Rxf>, [<Rn>, #<+/–<offset\_8>]! |
| PC= Rn[ofs8];  Rn+= ofs8 | LDR.W PC, [<Rn>, #+/–<offset\_8>]! |
| Rxf=Rn[按需左移后的Rm]  左移只能是0,1,2,3 | LDR.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}] |
| PC=Rn[按需左移后的Rm]  左移只能是0,1,2,3 | LDR.W PC, [<Rn>, <Rm>{, LSL #<shift>}] |
| Rxf= PC[ofs12] | LDR.W <Rxf>, [PC, #+/–<offset\_12>] |
| PC= PC[ofs12] | LDR.W PC, [PC, #+/–<offset\_12>] |
| Rxf=(U8) Rn[ofs12] | LDRB.W <Rxf>, [<Rn>, #<offset\_12>] |
| Rxf= (U8) \*Rn;  Rn+= ofs8 | LDRB.W <Rxf>. [<Rn>], #+/-<offset\_8> |
| Rxf= (U8) Rn[左移后的Rm];  左移只能是0,1,2,3 | LDRB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}] |
| Rxf= Rn[ofs8];  Rn+= ofs8 | LDRB.W <Rxf>, [<Rn>, #<+/–<offset\_8>]! |
| Rxf= PC[ofs12] | LDRB.W <Rxf>, [PC, #+/–<offset\_12>] |
| 读取Rn地址加上8位偏移量乘以4的处的双字到Rxf(低32位), Rxf2(高32位)，前索引 。并且可选在加载后更新Rn | LDRD.W <Rxf>, <Rxf2>, [<Rn>, #+/–<offset\_8> \* 4]{!} |
| 读取Rn处的双字到Rxf(低32位), Rxf2(高32位)，并且在加载后Rn+= ofs8\*4 | LDRD.W <Rxf>, <Rxf2>, [<Rn>], #+/–<offset\_8> \* 4 |
| Rxf= (U16) Rn[ofs12] | LDRH.W <Rxf>, [<Rn>, #<offset\_12>] |
| Rxf= (U16) Rn[ofs8];  Rn+=ofs8; | LDRH.W <Rxf>, [<Rn>, #<+/–<offset\_8>]! |
| Rxf= (U16) \*Rn;  Rn+= ofs8; | LDRH.W <Rxf>. [<Rn>], #+/-<offset\_8> |
| Rxf= (U16) Rn[左移后的Rm];  左移只能是0,1,2,3 | LDRH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}] |
| Rxf= (U16) PC[ofs12] | LDRH.W <Rxf>, [PC, #+/–<offset\_12>] |

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| 加载Rn+ofs12地址处的字节，并带符号扩展到Rxf中 | LDRSB.W <Rxf>, [<Rn>, #<offset\_12>] |
| 加载Rn地址处的字节，并带符号扩展到Rxf中。然后Rn+=ofs8 | LDRSB.W <Rxf>. [<Rn>], #+/-<offset\_8> |
| 先做Rn+=ofs8，再加载新Rn地址处的字节，并带符号扩展到Rxf中。 | LDRSB.W <Rxf>, [<Rn>, #<+/–<offset\_8>]! |
| 先把Rm按要求左移0,1,2,3位，  再加载Rn+新Rm地址处的字节，并带符号扩展到Rxf中 | LDRSB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}] |
| 加载PC+ofs12地址处的字节，并带符号扩展到Rxf中 | LDRSB.W <Rxf>, [PC, #+/–<offset\_12>] |
| 加载Rn+ofs12地址处的半字，并带符号扩展到Rxf中 | LDRSH.W <Rxf>, [<Rn>, #<offset\_12>] |
| 加载Rn地址处的半字，并带符号扩展到Rxf中。然后Rn+=ofs8 | LDRSH.W <Rxf>. [<Rn>], #+/-<offset\_8> |
| 先做Rn+=ofs8，再加载新Rn地址处的半字，并带符号扩展到Rxf中。 | LDRSH.W <Rxf>, [<Rn>, #<+/–<offset\_8>]! |
| 先把Rm按要求左移0,1,2,3位，  再加载Rn+新Rm地址处的半字，并带符号扩展到Rxf中 | LDRSH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}] |
| 加载PC+ofs12地址处的半字，并带符号扩展到Rxf中 | LDRSH.W <Rxf>, [PC, #+/–<offset\_12>] |
| Rd= Rn<<Rm | LSL{S}.W <Rd>, <Rn>, <Rm> |
| Rd= Rn>>Rm | LSR{S}.W <Rd>, <Rn>, <Rm> |
| Rd= Racc+Rn\*Rm | MLA.W <Rd>, <Rn>, <Rm>, <Racc> |
| Rd=Racc-Rn\*Rm | MLS.W <Rd>, <Rn>, <Rm>, <Racc> |
| Rd= Imm12 | MOV{S}.W <Rd>, #<modify\_constant(immed\_12)> |
| 先按需移位Rm，然后Rd=新Rm | MOV{S}.W <Rd>, <Rm>{, <shift>} |
| 将16位立即数传送到Rd的高半字中 ，Rd的低半字**不受影响** | MOVT.W <Rd>, #<immed\_16> |
| 将16位立即数传送到Rd的低半字中，并**把高半字清零** | MOVW.W <Rd>, #<immed\_16> |
| 把特殊功能寄存器的值传送到Rd中 | MRS<c> <Rd>, <psr> |
| 把Rn的值传送到特殊功能寄存器中 | MSR<c> <psr>\_<fields>,<Rn> |
| Rd= Rn\*Rm | MUL.W <Rd>, <Rn>, <Rm> |
| 无操作 | NOP.W |
| Rd= Rn | ~Imm12 | ORN{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12)> |
| 先按需要移位Rm，然后  Rd= Rn | ~新Rm | ORN[S}.W <Rd>, <Rn>, <Rm>{, <shift>} |
| Rd= Rn | Imm12 | ORR{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12) |

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| 先按需要移位Rm，然后  Rd= Rn | 新Rm | ORR{S}.W <Rd>, <Rn>, <Rm>{, <shift>} |
| Rd=把Rm的位反转后的值 | RBIT.W <Rd>, <Rm> |
| Rd=Rm字内的字节逆向 | REV.W <Rd>, <Rm> |
| Rd=Rn每个半字内的字节逆向 | REV16.W <Rd>, <Rn> |
| Rd=Rn低半字内的字节逆向后再符号扩展 | REVSH.W <Rd>, <Rn> |
| Rd= Rn圆圈右移Rm | ROR{S}.W <Rd>, <Rn>, <Rm> |
| Rd= Imm12-Rd | RSB{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12)> |
| 先按需移位Rm，然后  Rd= 新Rm-Rn | RSB{S}.W <Rd>, <Rn>, <Rm>{, <shift>} |
| Rd= Imm12-Rn-C | SBC{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12)> |
| 先按需移位Rm，然后  Rd=Rn-新Rm-C | SBC{S}.W <Rd>, <Rn>, <Rm>{, <shift>} |
| 抽取Rn中以lsb号位为最低有效位，共width宽度的位段，并带符号扩展到Rd中 | SBFX.W <Rd>, <Rn>, #<lsb>, #<width> |
| 带符号除法 , Rd= Rn/Rm | SDIV<c> <Rd>,<Rn>,<Rm> |
| 发送事件 | SEV<c> |
| 带符号64位乘加，RdHi:RdLo+= Rn\*Rm | SMLAL.W <RdLo>, <RdHi>, <Rn>, <Rm> |
| 带符号64位乘法，RdHi:RdLo= Rn\*Rm | SMULL.W <RdLo>, <RdHi>, <Rn>, <Rm> |
| 先按需移位Rn，再把Rn向低Imm位执行带符号饱和操作，并把结果带符号扩展后写到Rd中 | SSAT <c> <Rd>, #<imm>, <Rn>{, <shift>} |
| 多个寄存器字连续保存到由Rn给出的首地址中 ，并且在Rn上，每存储一个后自增(IA)/每存储一个前自减(DB) | STM{IA|DB}.W <Rn>{!}, <registers> |
| Rn[ofs12]=Rxf | STR.W <Rxf>, [<Rn>, #<offset\_12>] |
| \*Rn=Fxf;  Rn+=ofs8 | STR.W <Rxf>, [<Rn>], #+/–<offset\_8> |
| 先按需左移Rm，然后  Rn[新Rm]=Rxf，左移格数只能是0,1,2,3 | STR.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}] |
| Rn[ofs8]=Rxf  若有“！”，则还执行Rn+=ofs8 | STR{T}.W <Rxf>, [<Rn>, #+/–<offset\_8>]{!} |
| \*( (U8\*) (Rn+ofs8) ) = (U8) Rxf  若有“！”，则还执行Rn+=ofs8 | STRB{T}.W <Rxf>, [<Rn>, #+/–<offset\_8>]{!} |
| \*( (U8\*) (Rn+ofs12) ) = (U8) Rxf | STRB.W <Rxf>, [<Rn>, #<offset\_12>] |
| \*( (U8\*) Rn ) = (U8) Rxf  Rn+=ofs8 | STRB.W <Rxf>, [<Rn>], #+/–<offset\_8> |
| 先按需左移Rm，左移格数只能是0,1,2,3，再  \*( (U8\*) (Rn+新Rm) ) = (U8) Rxf | STRB.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}] |
| \*(Rn+ofs8\*4)=Rxf;  \*(Rn+ofs8\*4+4)=Rxf2  若有“！”，则Rn+=ofs8 | STRD.W <Rxf>, <Rxf2>, [<Rn>, #+/–<offset\_8> \* 4]{!} |
| \*Rn=Rxf;  \*(Rn 4)=Rxf2;  Rn+=ofs8\*4 | STRD.W <Rxf>, <Rxf2>, [<Rn>], #+/–<offset\_8> \* 4 |
| \*( (U16\*) (Rn+ofs12) ) = (U16) Rxf | STRH.W <Rxf>, [<Rn>, #<offset\_12>] |
| 先按需左移Rm，左移格数只能是0,1,2,3，再  \*( (U16\*) (Rn+新Rm) ) = (U16) Rxf | STRH.W <Rxf>, [<Rn>, <Rm>{, LSL #<shift>}] |

|  |  |
| --- | --- |
| 操作 | 汇编指令 |
| \*( (U16\*) (Rn+ofs8) ) = (U16) Rxf  若有“！”，则还执行Rn+=ofs8 | STRH{T}.W <Rxf>, [<Rn>, #+/–<offset\_8>]{!} |
| \*( (U16\*) Rn ) = (U16) Rxf  Rn+=ofs8 | STRH.W <Rxf>, [<Rn>], #+/–<offset\_8> |
| Rd= Rn-Imm12 | SUB{S}.W <Rd>, <Rn>, #<modify\_constant(immed\_12)> |
| 先按需移位Rm  Rd= Rn-新Rm | SUB{S}.W <Rd>, <Rn>, <Rm>{, <shift>} |
| Rd= Rn-Imm12 | SUBW.W <Rd>, <Rn>, #<immed\_12> |
| 先按需圆圈移位Rm，然后取出Rm的低8位，带符号扩展到32位并存储到Rd | SXTB.W <Rd>, <Rm>{, <rotation>} |
| 先按需圆圈移位Rm，然后取出Rm的低16位，带符号扩展到32位并存储到Rd | SXTH.W <Rd>, <Rm>{, <rotation>} |
| PC+= ( (U8)\*(Rn+Rm) )\*2 | TBB [<Rn>, <Rm>] |
| PC+= ( (U16)\*(Rn+Rm\*2) )\*2 | TBH [<Rn>, <Rm>, LSL #1] |
| Rn与Imm12按位异或，并根据结果更新标志位 | TEQ.W <Rn>, #<modify\_constant(immed\_12)> |
| 先按需移位Rm，然后  Rn与Rm按位异或，并根据结果更新标志位 | TEQ.W <Rn>, <Rm>{, <shift>} |
| Rn与Imm12按位与，并根据结果更新标志位 | TST.W <Rn>, #<modify\_constant(immed\_12)> |
| 先按需移位Rm，然后  Rn与Rm按位与，并根据结果更新标志位 | TST.W <Rn>, <Rm>{, <shift>} |
| 抽取Rn中以lsb号位为最低有效位，共width宽度的位段，并无符号扩展到Rd中 | UBFX.W <Rd>, <Rn>, #<lsb>, #<width> |
| 无符号除法 Rd= Rn/Rm | UDIV<c> <Rd>,<Rn>,<Rm> |
| 无符号64位乘加，RdHi:RdLo+= Rn\*Rm | UMLAL.W <RdLo>, <RdHi>, <Rn>, <Rm> |
| 无符号64位乘法，RdHi:RdLo= Rn\*Rm | UMULL.W <RdLo>, <RdHi>, <Rn>, <Rm> |
| 先按需移位Rn，再把Rn向低Imm位执行带符号饱和操作，并把结果无符号扩展后写到Rd中 | USAT <c> <Rd>, #<imm>, <Rn>{, <shift>} |
| 先按需圆圈移位Rm，然后取出Rm的低8位，无符号扩展到32位并存储到Rd | UXTB.W <Rd>, <Rm>{, <rotation>} |
| 先按需圆圈移位Rm，然后取出Rm的低16位，无号扩展到32位并存储到Rd | UXTH.W <Rd>, <Rm>{, <rotation>} |
| 等待事件 | WFE.W |
| 等待中断 | WFI.W |