XA 用户手册

3 XA存储器的结构

3.1介绍

XA的存储器的结构是哈佛结构,这就意味着代码空间和数据空间(包括SFR)有着分离的地址空间.XA支持16M(24根地址线)的数据和代码两个空间.但不同的型号可能有所不同.

XA支持不同的数据空间和代码空间.代码空间可以是Eprom,EEProm,OTP ROM,Flash和掩膜ROM,数据空间可以是RAM,EEProm,或Flash.

这一章将讨论XA的存储器的组织,包括寄存器,数据空间,代码空间;如何操作这些空间,这些空间如何关联等.

3.2 XA寄存器队列

XA的寄存器队列经过优化处理,以利于运算,逻辑,和地址运算操作.

3.2.1 XA寄存器队列概述

XA结构的寄存器队列总有16个字宽度的寄存器:在基本的XA结构中,只有R0到R7可用.除了R7,这些寄存器可以任意使用.R7是堆栈指针,如图3.1所示.这样,XA提供了7个独立的累加器,用于各种操作.下面可以看到,XA的寄存器可以按位,字节,字,双字进行操作.
附加的寄存器R8到R15,是为以后的XA型号保留,如果将来提供,这些存储器将等同于R0到R7,只是不能进行位操作和指针操作.寄存器队列同XA的其它存储器空间是独立的(兼容模式除外,见第9章)

寄存器阵列详述
图3.2 详细的描述了R0到R7.

字节,字,双字寄存器
所有的寄存器可以按位,字节,字,在一些情况下还可以进行双字操作.寄存器的位将在下一节介绍.对于字节和字操作,以R1为例,由于它是字寄存器,因此简单的字引用为"R1",高有效位的引用为"R1H",低有效位的引用为"R1H".双字寄存器是由两个相邻的寄存器合成的,用于32bit的移位,乘法和除法.寄存器对的引用名是低位寄存器(包含低位的字),而且它是偶数序号.于是有效的双字节寄存器对是(R0,R1), (R2,R3), (R4,R5) and (R6, R7).

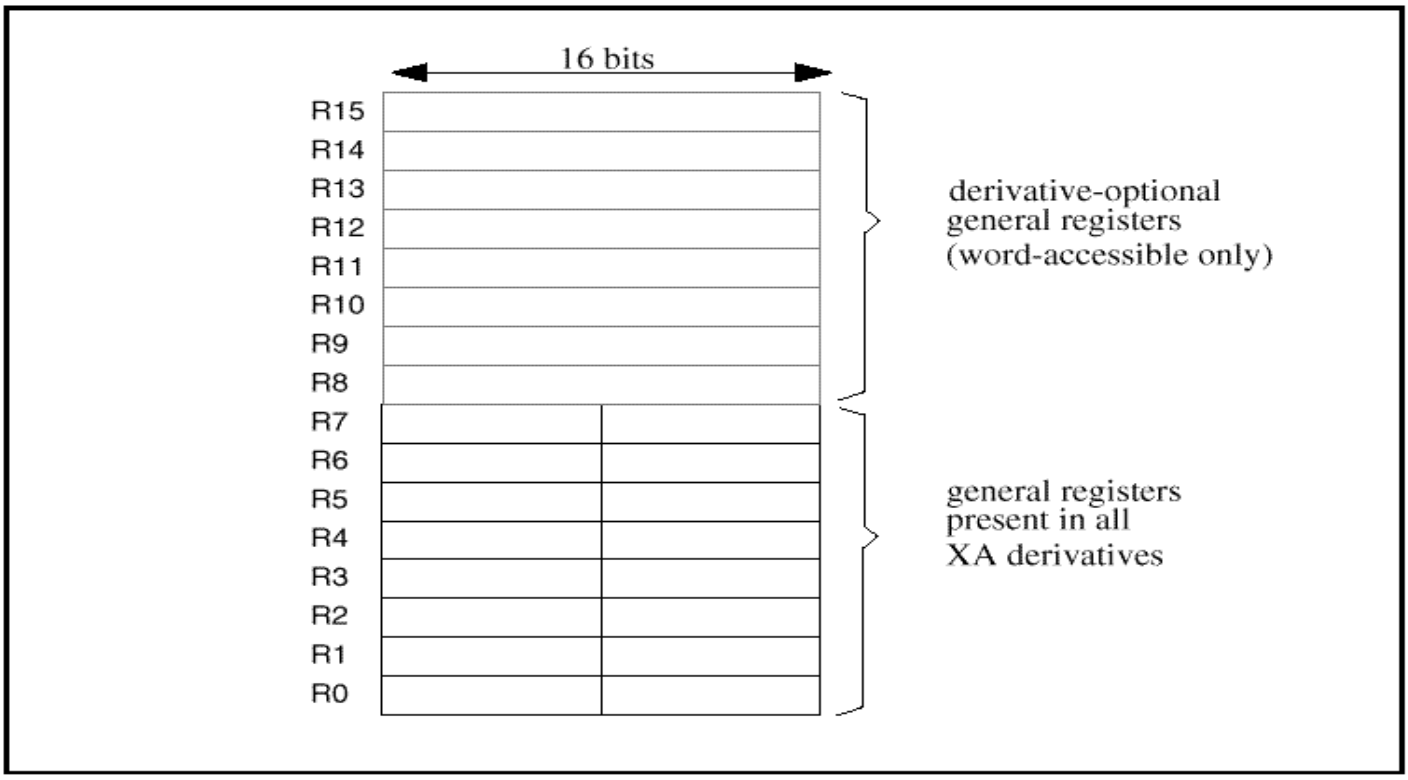


图 **3.1 XA**寄存器队列总图

如4.7节所述,XA有两个堆栈指针,一个是用户模式,另一个是系统模式.在同一时间内,只有一个指针是可操作的,它的值总在R7里.当PSW.SM=0,用户模式被激活,通过R7,USP变成可操作的.当PSW.SM=1,XA运行在系统模式,SSP进入R7中.(注意:在所有的中断体系在系统堆栈中保存,使用SSP,而不管当前的操作模式.)

R0到R3则有所不同.在一个给定的时间里,4组寄存器中只有一组是有效的,引用为R0到R3,其它组的内容则是不可操作的.这样允许高速文本切换,例如中断服务程序.**PSW**的位**[RS1,RS0]**选择当前有效的寄存器组:

RS1	RS0	可见的寄存器组
0	0	0组
0	1	1组
1	0	2组
1	1	3组

PSW.RSn在用户模式和系统模式下都是可改写的.运行在两种模式下的程序都可以改写这些位,以选择寄存器组. 通常,如第4章所述,中断机制能提供一个绝对的寄存器组切换,这样中断处理可以立即在保留的寄存器组中进行操作.

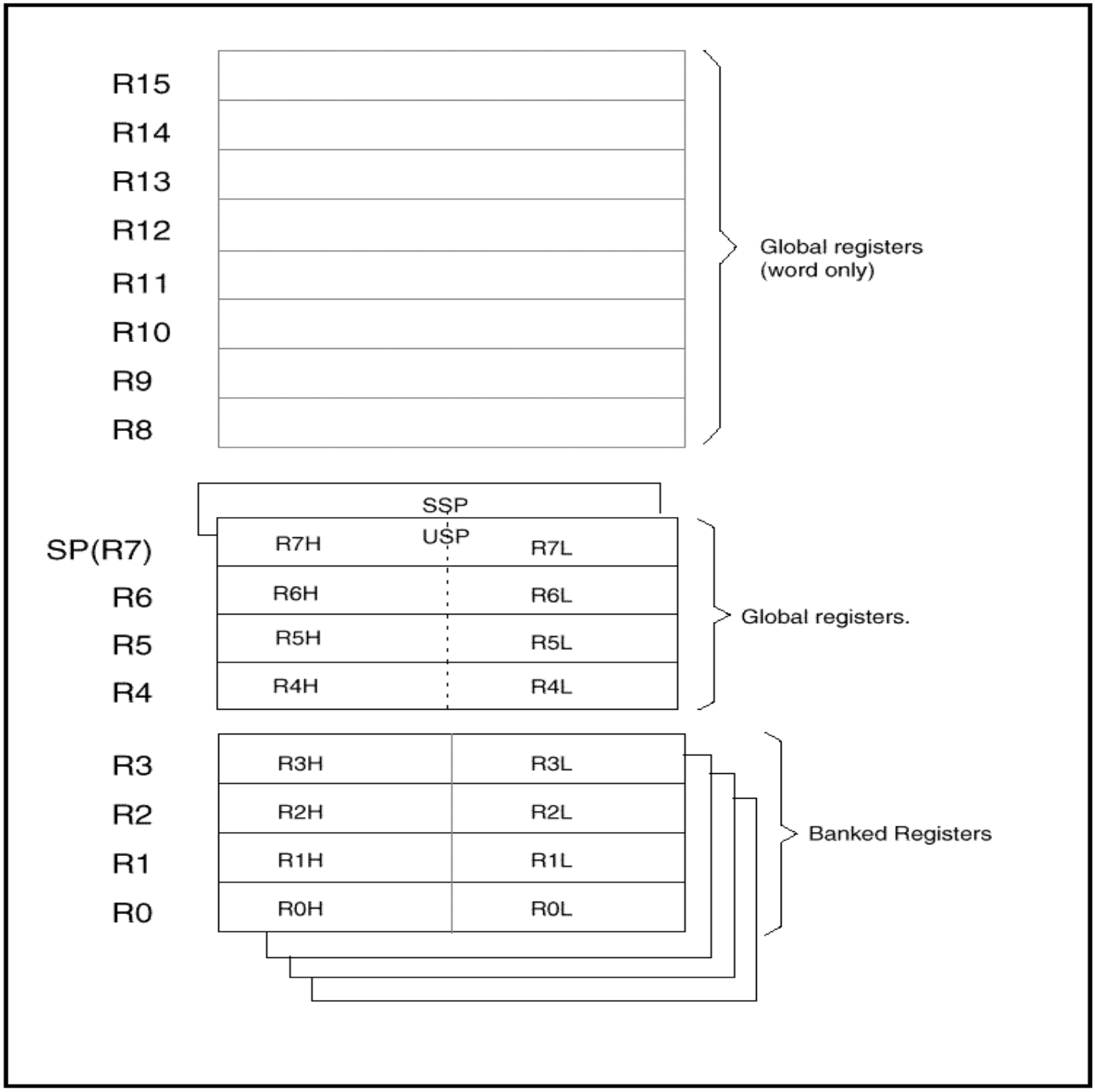


图 **3.2 XA**寄存器队列

寄存器的位操作

XA的寄存器全部是可位寻址的.图3.3表明寄存器阵列的位地址分布.通常,如图所示的绝对地址引用是不必要的.XA的软件工具提供了相关符号用于对寄存器位的操作.例如,bit7可以表达为R0.7,非常明确.

对R0到R3的位的引用来源于当前的寄存器组,当前的寄存器组由**PSW**的**RS1 RS0**决定,没选择的寄存器组是不可操作的;同样,对R7的操作也是对当前堆栈指针(SSP或USP)而言.

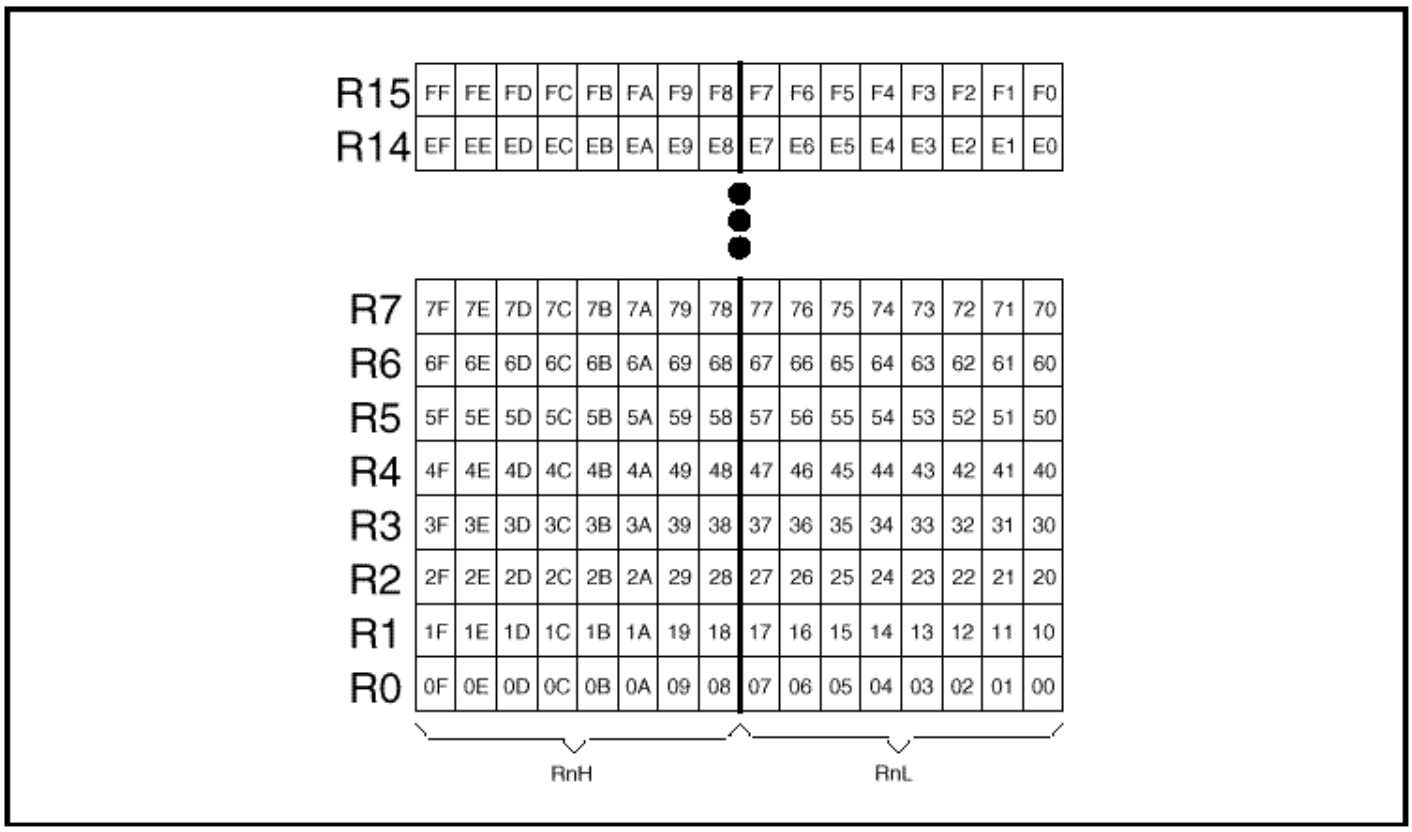


图**3.3**寄存器的位地址

3.3 XA的数据空间

XA把物理空间分成数据空间和程序空间.24-bit地址位,对应16M的地址空间.在给定的应用中,实际用到的地址空间可能比24-bit少,可能用到的小模式只用到16位地址,见第4章.代码和数据存储器可能在片上,或者外部,这要根据XA的型号和用户的使用.不管指定的区域在片上或外部,一般都不会影响存储器的操作.

3.3.1 字节,字, 和排列

XA中寻址是以字节为单位的,一个比特包含8个位.一个字包含两个字节,字的存储位置是低有效位安排在地址的低端.见图3.4.

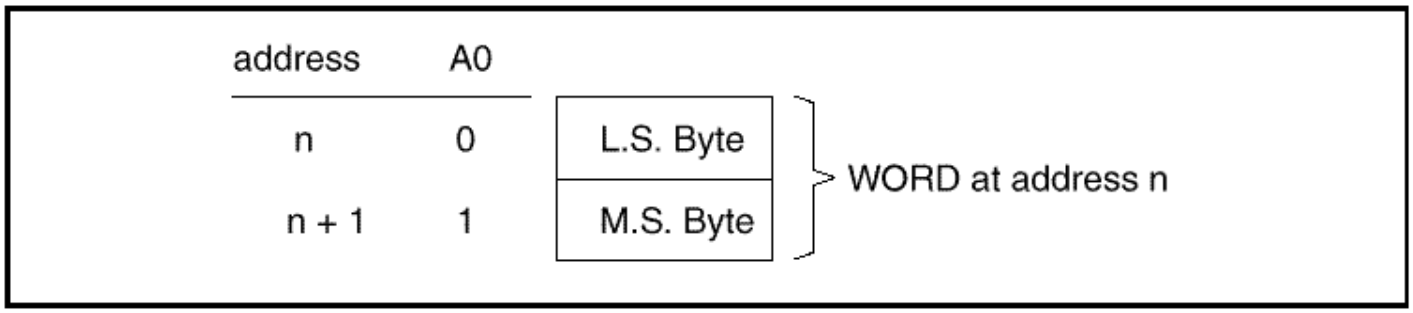


图**3.4** 字节的顺序

任何字操作必须从偶数地址开始(A0=0).如果想操作一个奇数开始的字,则实际操作的是前一个最小的偶数开始的地址,这就是A0必须为0.

XA的外部空间可以按字节或字操作,但是硬件操作方法不会影响偶数地址起始的限制.

3.4 数据空间

数据空间由地址0开始,直到最高的应用有效地址,最大为FFFFFFH.如下所述,数据空间分成256个段,每个段为64K.外部数据空间由内部最大的地址的下一个地址开始.通常,内部有512个字节,从0开始,在所有的应用中都不会变.但是,内部固有的数据大小并没有一定的限制.高端的16个段的数据区(F0:0000到FF:FFFF)为将来的XA系列保留,对于代码段也一样.见3.5节.

3.4.1数据区域的排列

数据区域的排列没有限制,除非数据区全部放入字:字必须从偶数地址开始读取.试图从奇数地址读取将读取前一个偶数地址.

3.4.2外部和内部的重叠

如果外部数据区域的放置同内部数据在逻辑上重叠,内部数据将占先.重叠部分的外部数据可以用MOVX指令读取.见第6章.使用MOVX将迫使XA从外部读取数据.对于不重叠的部分,没有必要使用MOVX指令.

3.43使用和读写操作

数据空间用来读写,并存储读写的数据.从逻辑上将是不可能从数据空间来执行代码的.但确实又是可行的,通常的做法是通过添加逻辑使地址空间和数据空间重叠.在这种情况下,正确理解空间分离是重要的.在这种修正的哈佛结构中,可以写入自己修改的XA代码,但一般不建议这样做.但是内部数据空间不可能这样重叠.

3.4.4 数据空间的寻址

XA的数据空间的寻址经过优化,以适应嵌入处理的需要.数据空间被分成64K的段.这样能为多任务系统提供真正意义上的保护机制,也能为本地操作提高性能,因为只需要较少的地址位.

通过段寄存器寻址

如果设计中需要完整的16M地址空间,就要使用段寄存器提供高8位的地址,这样就可以获得完整的24位地址线.在XA中定义了两个段寄存器,**DS**和**ES**用于对数据空间的操作.如果用户堆栈的位置已经由DS的数值确定,那么使用ES来寻址用户数据就可以很方便.通过段选择寄存器**SSEL**选择了段寄存器后,就可以通过指针寄存器R0到R6,再结合选择的段寄存器,对整个XA数据空间进行访问.见图3.5.

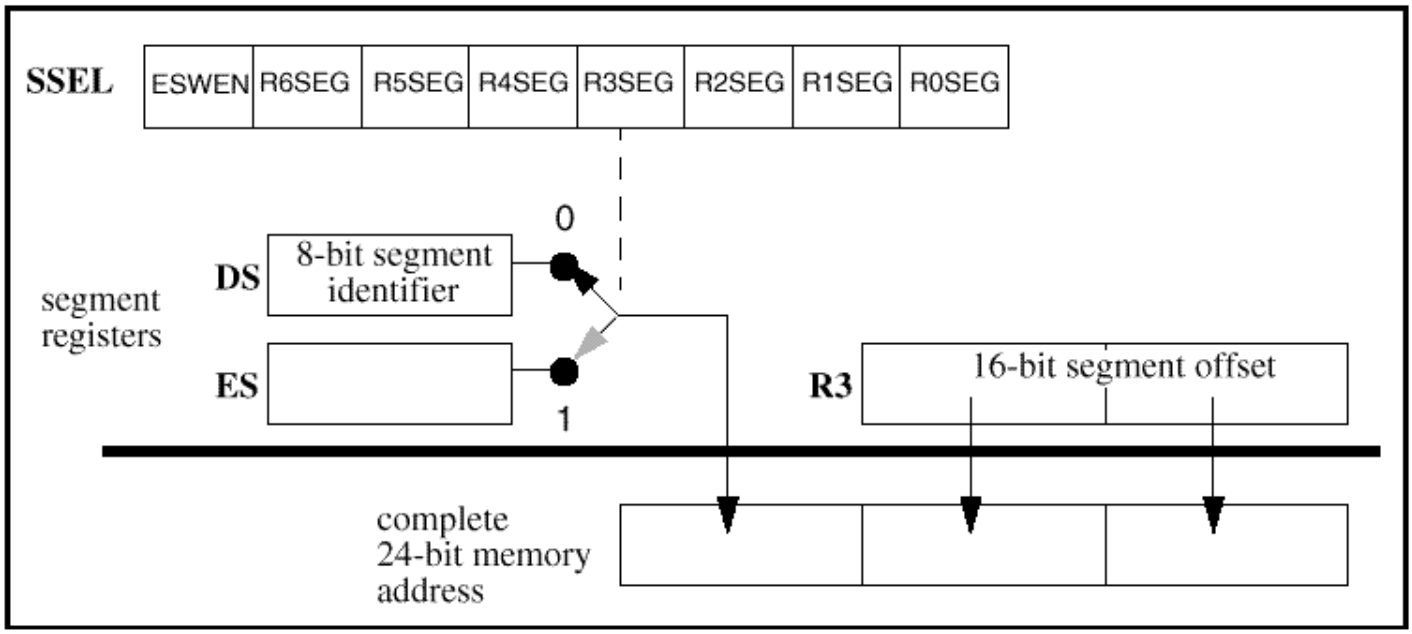


图 **3.5** 地址的产生

在**SSEL**段选择寄存器中,对应的位为0选择**DS**(复位后的缺省值);1对应选择**ES**.举例来讲,如果R3包含指针信息,再连接上ES或DS(由**SSEL**的第3位),就形成一个完整的24bits地址.这样寻址的结果是256个段,每段64K容量.(见图3.6)

如果R7被当成一个普通的间接指针使用,数据段寻址总是在系统模式的段0中,DS段在用户模式中.详情请见节4,节5.

SSEL的第7位**ESWEN**只能在系统模式下修改,1开启在用户模式下通过ES访问数据段的权限,0则关闭此权限.此为在复位后为0.

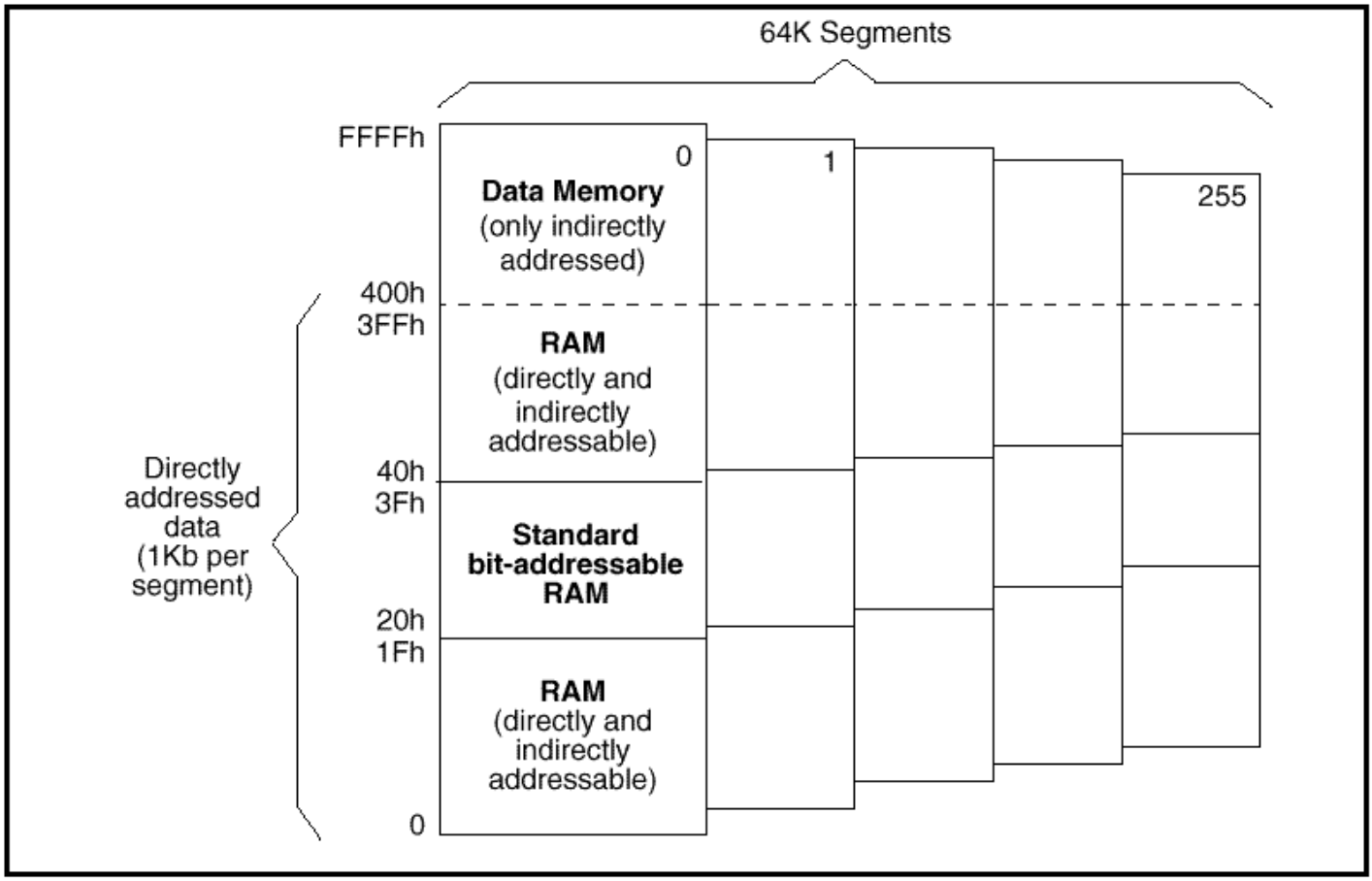


图 **3.6** 数据存储器段

寻址模式

XA提供了灵活的数据寻址模式.运算,逻辑,数据移动指令通常支持如下的数据区域操作:

■间接寻址.8bit的段寄存器,16bit的寄存器指针,共同形成一个24bit的完整地址.
■直接寻址.每个数据段的前1K范围的数据,寻址时地址可以直接出现在指令中.
■**带偏移的间接寻址.**指令中包含一个带符号的字节或字,将被加到指针的内容上,从而同段寄存器DS形成一个完整的24bits地址.
■**自动增量的间接寻址.**地址形成如上,但每次操作后指针寄存器可以自动增加一个数值.
■**位寻址.**位的引用必须使用相应位的绝对地址.
■数据移动指令和一些特殊用途的指令也有一些附加的寻址模式,见第6章.

间接寻址

通过寄存器间接寻址和段寄存器,就可以对全部的16M空间进行操作.如图3.7所示(注意:为了简单,图表省略了**SSEL**如何选择数据段寄存器和附加段寄存器的.)

带偏移量的间接寻址,同普通的间接寻址相比,它的偏移量是可以在8bit或16bit范围内变化的,变化量加到指针寄存器的内容中,再同8bit的段寄存器结合,形成一个完整的地址.这种形式可以对数据结构进行操作,只要用指针寄存器指向数据结构的首址.它也支持堆栈的参数传递.

自动增量的间接寻址是另一种可变化的间接寻址方式,它在操作完成后,指针寄存器的内容自动增加,如果操作对象是字节,则增加量为1;如果操作对象是字,则增加量为2.使用自动增量的间接寻址方法,可以方便的处理小于64K的数据结构.对于大于64K的数据结构,程序代码必须在页边界处调整段寄存器.

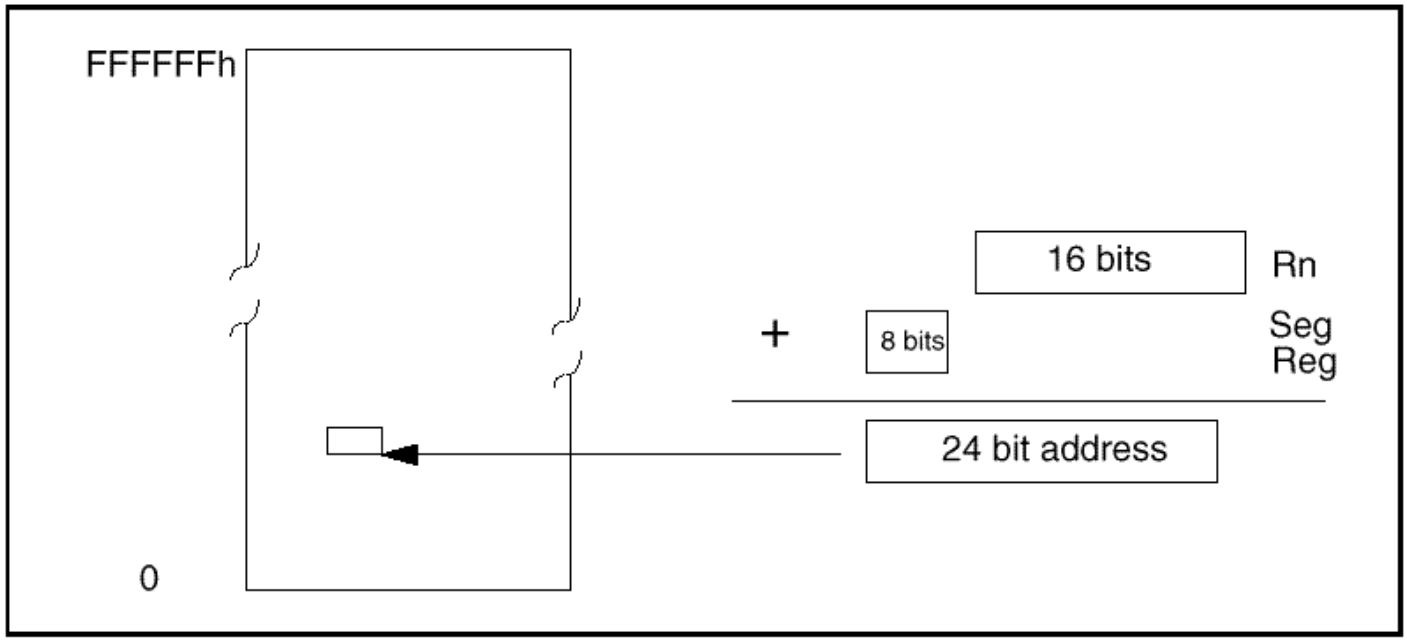


图 **3.7** 间接寻址操作**24 Bit**地址空间

图3.8和图3.9显示了这两种间接寻址的方式.使用间接寻址方式必须注意避免从奇数地址对字对象进行操作.这将操作了前一个的偶数地址,当然这是不希望的.注意,带偏移的间接寻址在这种情况下却是可以的,只要最后的地址是偶数,当然基址和偏移值必须同时是奇数或偶数.

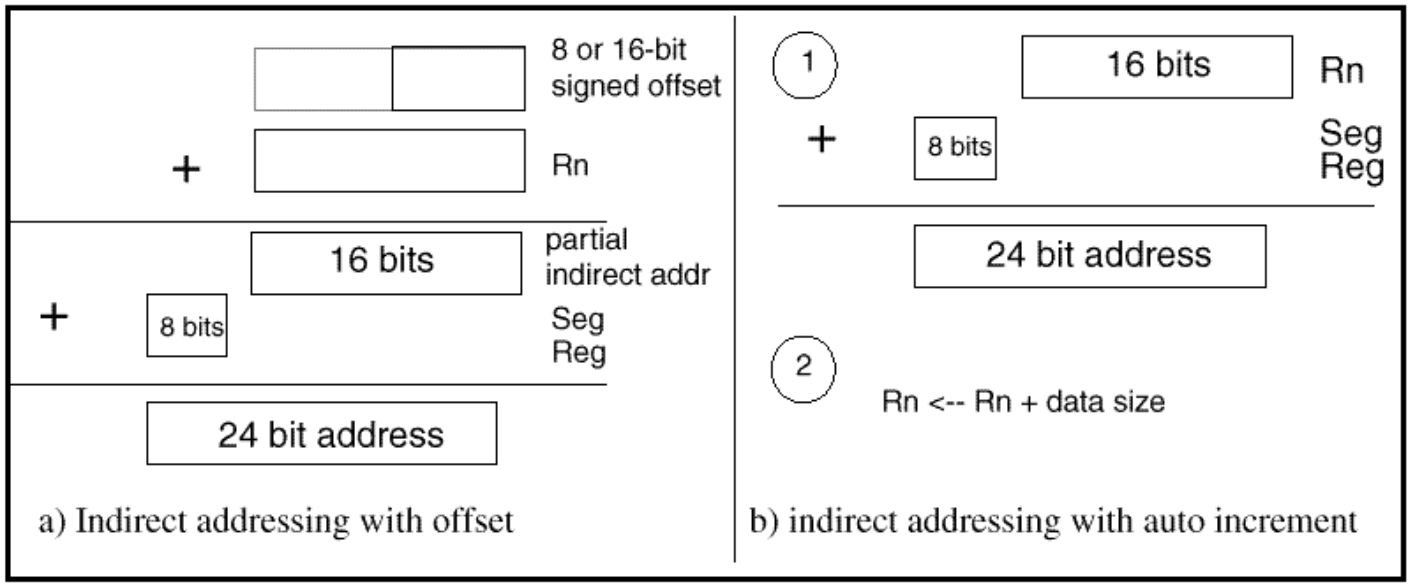


图 **3.8** 间接寻址

直接寻址

每个段的前1K范围是可以直接寻址的.图3.10显示了直接寻址模式下的地址产生情况,总是使用DS段寄存器.直接寻址指令被译码成最大只有10地址bit的有效位,其余的bit位(由11延伸到16)全部为0,这样保证直接寻址范围在1K范围内,同DS结合后,形成完整的24bit地址.在所有的段中,可以使用直接寻址对起始的1K范围的数据进行操作.

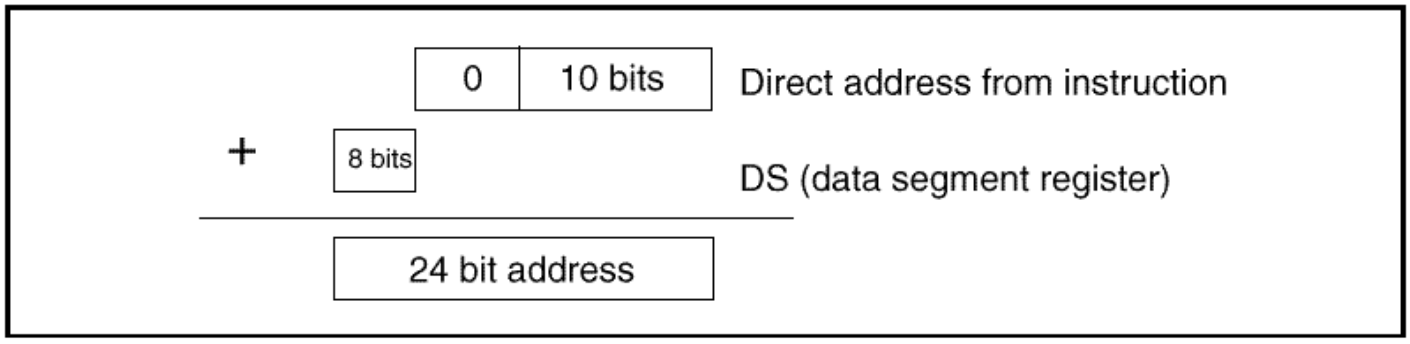


图 **3.9** 直接寻址地址的产生

SFR 寻址

有1K的直接寻址空间,地址为400h到7FFh,是留给SFR的.SFR使用了一部分直接寻址的空间,但它同数据空间不发生联系,完全是一个独立的逻辑空间.见节3.6,查看详细的SFR操作描述.

位寻址

在每个由DS寻址的数据段中,有32字节的可位寻址的空间,地址由20H开始.图3.10显示了数据区按位寻址的地址产生情况.如第6章所述,指令中的bit地址被译码成10bit.图3.11显示了bit的地址位置图.

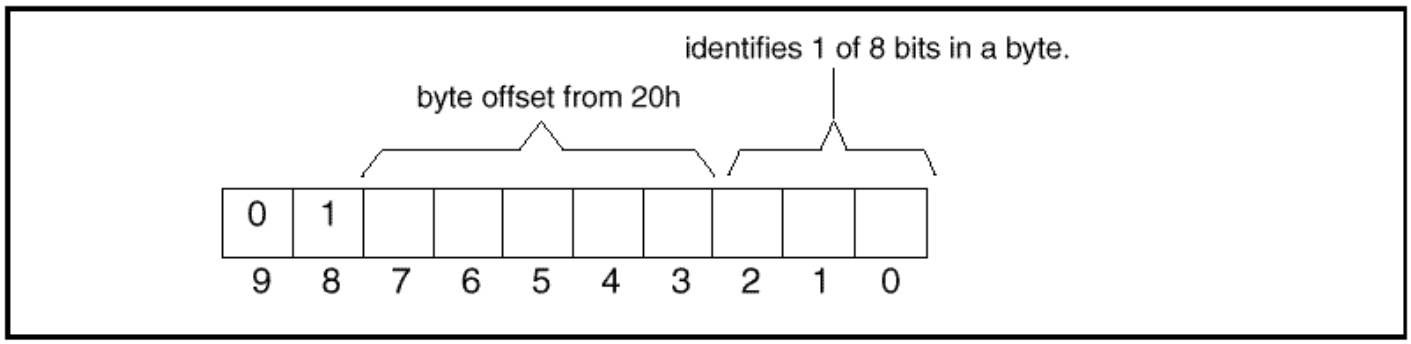


图 **3.10 Bit**地址的产生

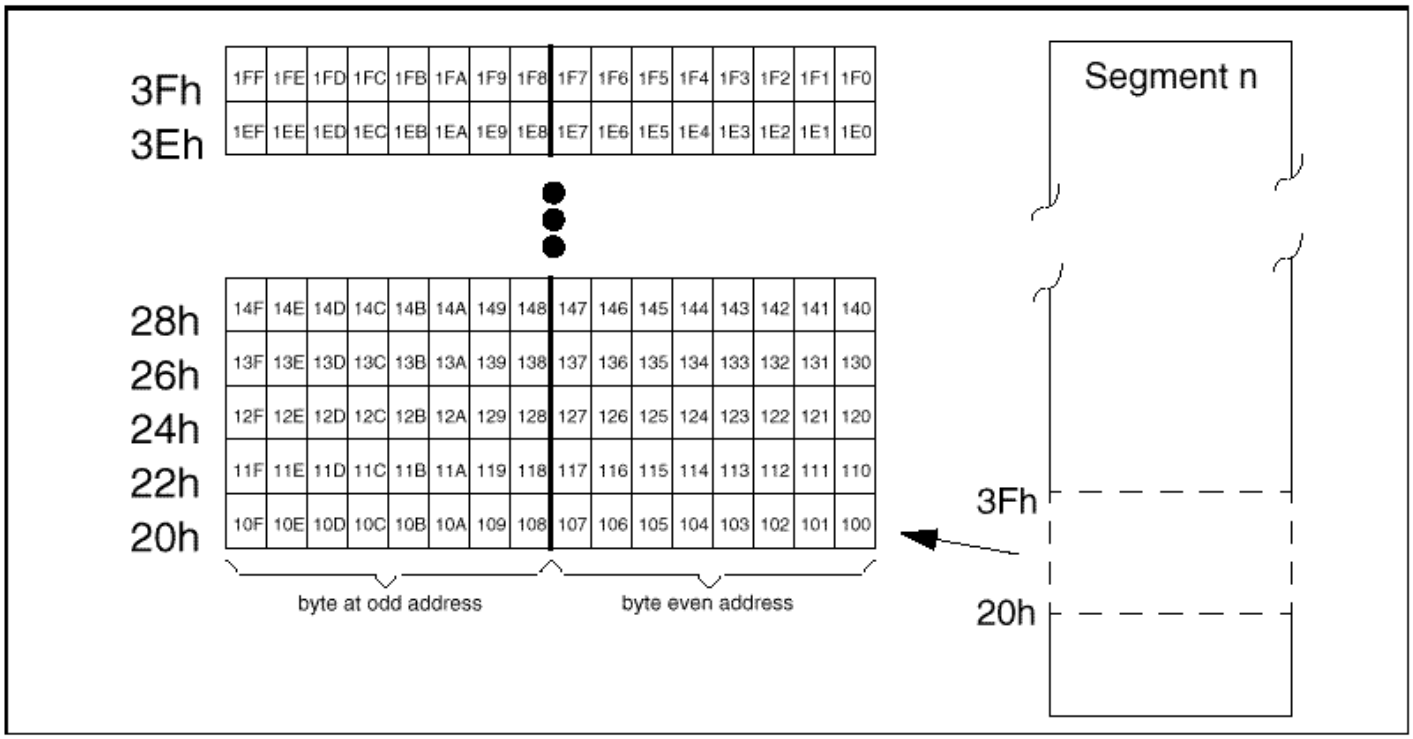


图**3.11** 直接位地址地址分布

3.5 代码区域

代码区域由0开始,延伸到实际中用到的最高地址,最大为FFFFFFH. 片内的地址一般从0开始,片外的地址从片内地址的高端的下一个开始.最高端的16个64K的页(地址从F00000到FFFFFF)为将来特殊的XA型号保留,在数据空间相同的地址范围也予以保留.见节3.4.

3.5.1 代码区域中的排列

由于指令代码长度在1到6字节中变化(见第6章),代码区域中的指令可以定位于奇数地址.但是指令分支的目标,跳转的目标,调用,分支,陷阱,和中断必须排列在偶数地址上.

3.5.2 外部和内部的重叠

如果内部代码地址和外部代码地址在空间上重叠,内部代码地址优先,重叠的外部代码无法操作.但是如果将EA脚接成低电平,将迫使XA使用外部代码.

3.5.3 操作

代码区域是用来存放可执行的指令代码.XA结构也允许将一些常量放入代码区域,同时提供了特殊的操作模式取回这些信息.在指定了立即操作数的数据操作指令中,常量数据隐含的包含在指令中.
根据经验,数据区域和代码区域重叠在一起是可能的.在这种情况下,必须懂得数据空间和代码空间在逻辑上仍然是分离的.在这种结构中,代码空间是只读的,但通过改写数据空间的办法,它又是可写的.

MOVC 在代码空间的寻址

利用XA的特殊指令MOVC,可以读取放在代码区域的常量数据(表格,常量字符串).MOVC有一种XA的独有标准格式,另外有两种格式用于兼容80C51.见第9章,关于80C51的兼容信息.标准的MOVC格式使用一个16位的寄存器作为指针,结合8bit的程序计数器(PC)或代码段寄存器(CS)形成一个完整的24-bits地址,见图3.12.高8-bits的地址选择是有**SSEL**中响应的寄存器的选择位决定的(0=PC,1=CS)

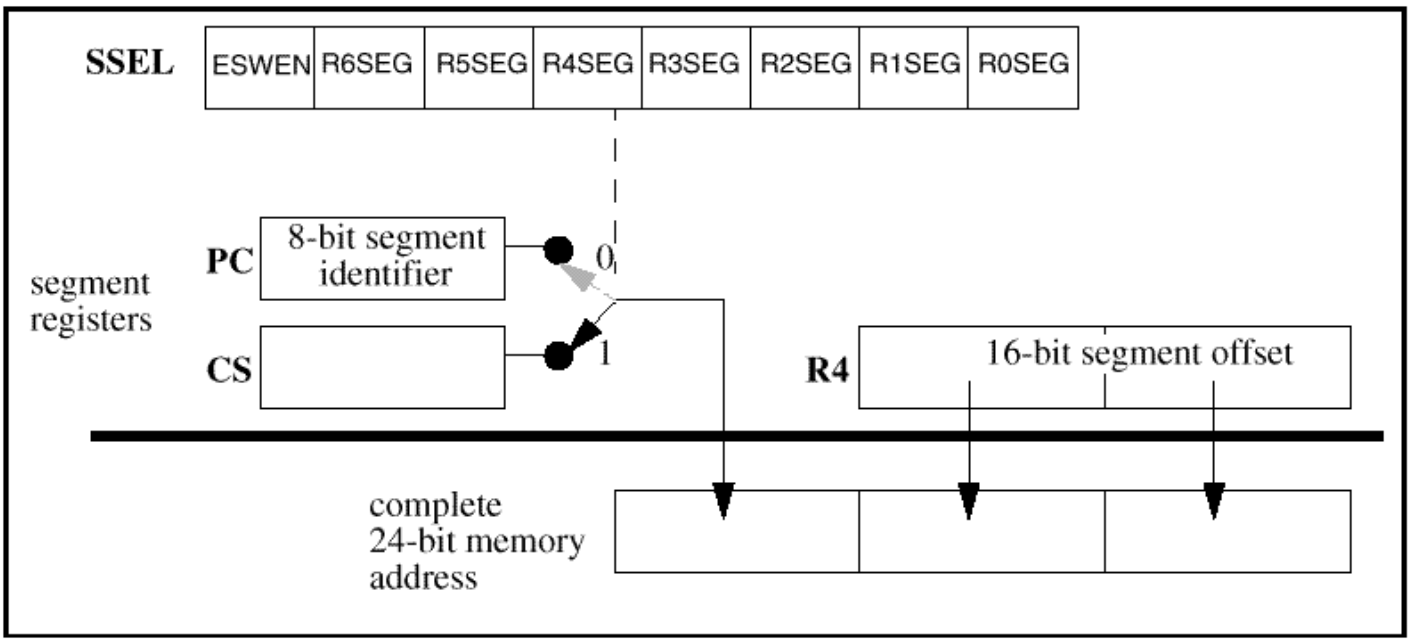


图**3.12 MOVC**在代码区域中的寻址

3.6特殊功能寄存器(SFRs)

特殊功能寄存器(SFRs)提供了程序访问CPU控制和状态寄存器,外设,I/O口的方法.同时也可以在将来的器件中对新加入的外设,外部器件操作.图3.13高亮度显示了可操作的SFR:**PCON**, **SCR**, **SSEL**, **PSWH**, PSWL, CS, ES, DS.同这些寄存器通讯,也包括片上的外设器件,是通过特定的SFR总线(见第8章).
SFR的地址空间是1K字节(见图3.14).前半个(400h到5FFh)是核寄存器和片上但是核外的外设.地址分配在(400h到43Fh)是可以按位寻址的.后半个(600h到7FFh)的SFR空间为片外的SFR保留.片外的SFR空间可以对片外的影射I/O器件提供快速的操作,而不必为每一次操作制造一个指针.

下面是在使用SFR时应该注意的关键
SFR应该用符号代替地址.因为SFR的地址可能因为器件的变化而改变,所以用符号引用是很重要的.XA的软件开发工具以头文件/包含文件的方式提供了SFR的符号常量,并随着器件的变化而更新.你要确认你的应用中使用了正确的头文件/包含文件.虽然XA的基本系列在将来也会保持原来的地址,但这并不保证.在不同的器件中,可选的外设可能有着不同的地址,相同的地址在另外一个器件中可能操作的是一个不同的SFR外设.

SFR可以在任何时间操作,不用动用指针和段.SFR的操作是独立于任何段寄存器以外的,所以SFR的操作地址总是由指令得到的10 bit.

*SFR*不允许间接寻址.如果间接寻址,操作的将是数据区域,就是说,如果间接寻址,该地址的RAM单元如果存在的话将被操作.

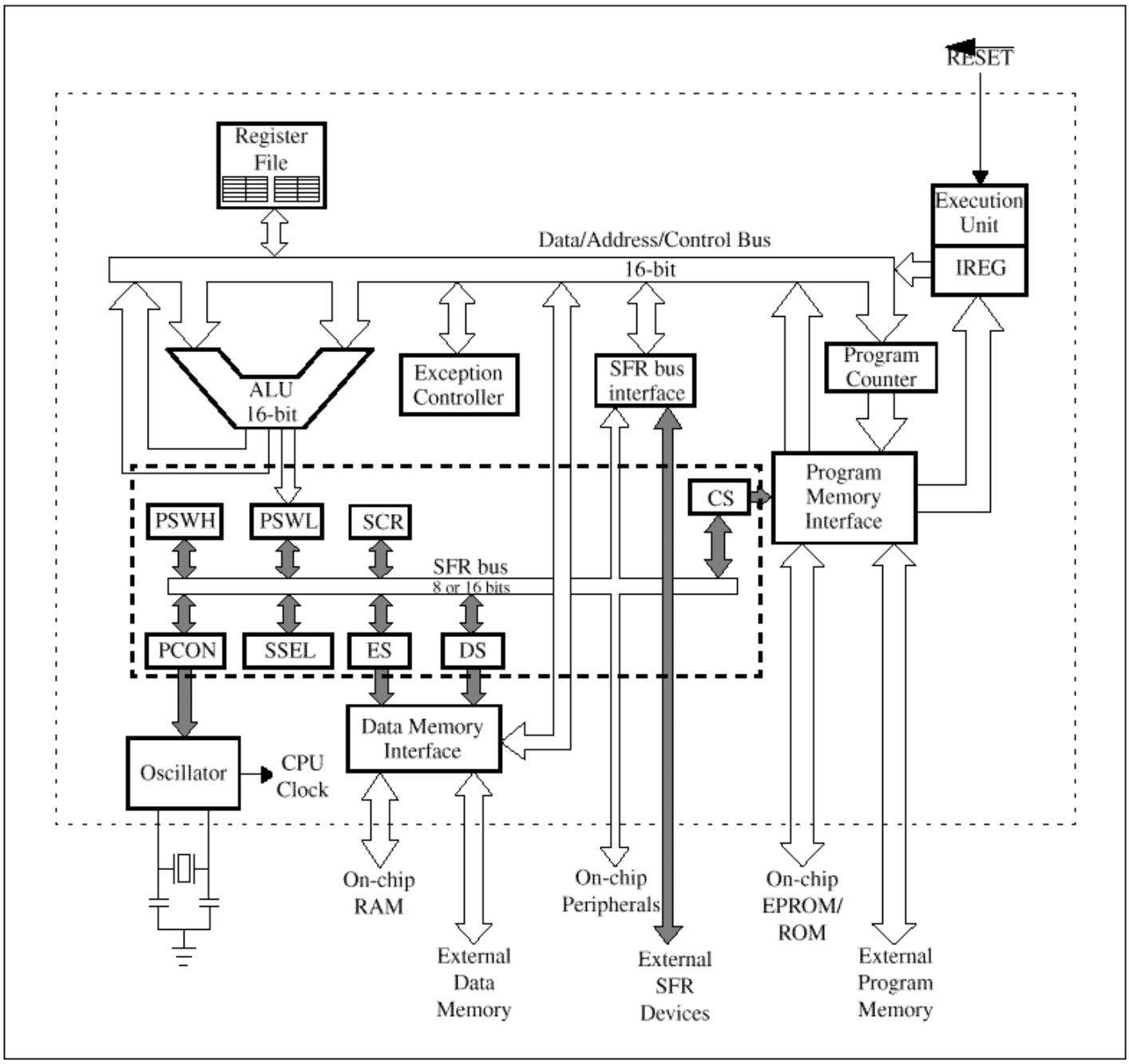


图**3.13 XA** 核的**SFRs(**高亮度显示**)**

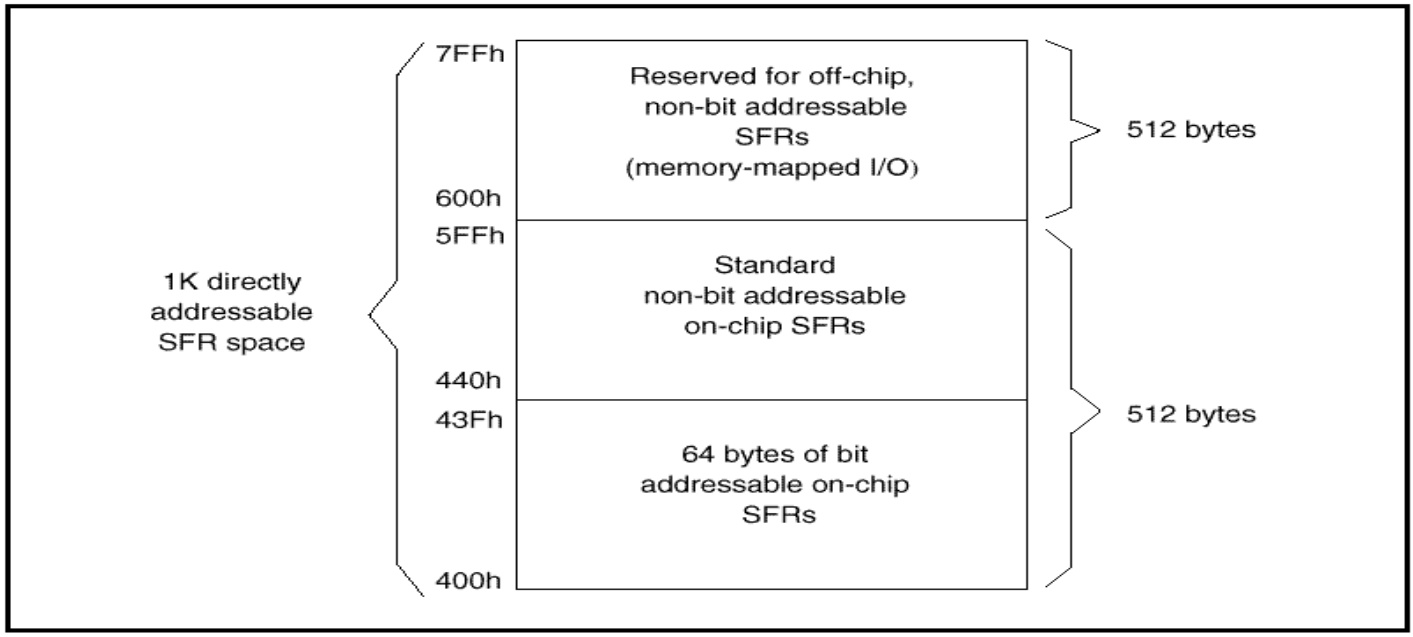


图**3.14 SFR**地址空间

*SFR*的地址必须完全包含在指令中.译码后提供给操作,没有别的方法可以对SFR寻址.外部SFR的操作将由指定的型号确定.操作片外的SFR是为将来的器件保留,基本的XA并不提供.细节请参考器件手册.

3.7位寻址综述

本章的好多节描述了位寻址.XA的结构中共有1024个bit位,很多重要的数据结构都提供立即的位寻址,寄存器阵列中R0到R7(R8到R15,如果提供);在由DS提供的当前页中,直接寻址的RAM,,范围在20h到3Fh,部分片内的SFR.图3.15汇总了XA的部分可位寻址位.

bit space		overlaps bytes...		
start	end	type	start	end
0	0FFh	registers	R0	R15
100h	1FFh	direct RAM	20h	3Fh
200h	3FFh	on-chip SFRs	400h	43Fh

图**3.15** 位寻址总汇

4 CPU 组织

本章描述XA核的中央处理单元(CPU).CPU包含XA所有的状态和逻辑控制.XA的复位时序,系统震荡器同CPU的接口,功率控制,中断和异常处理.XA配置了特殊的功能支持调试.

4.1介绍

图 4.1 是XA的功能块图.

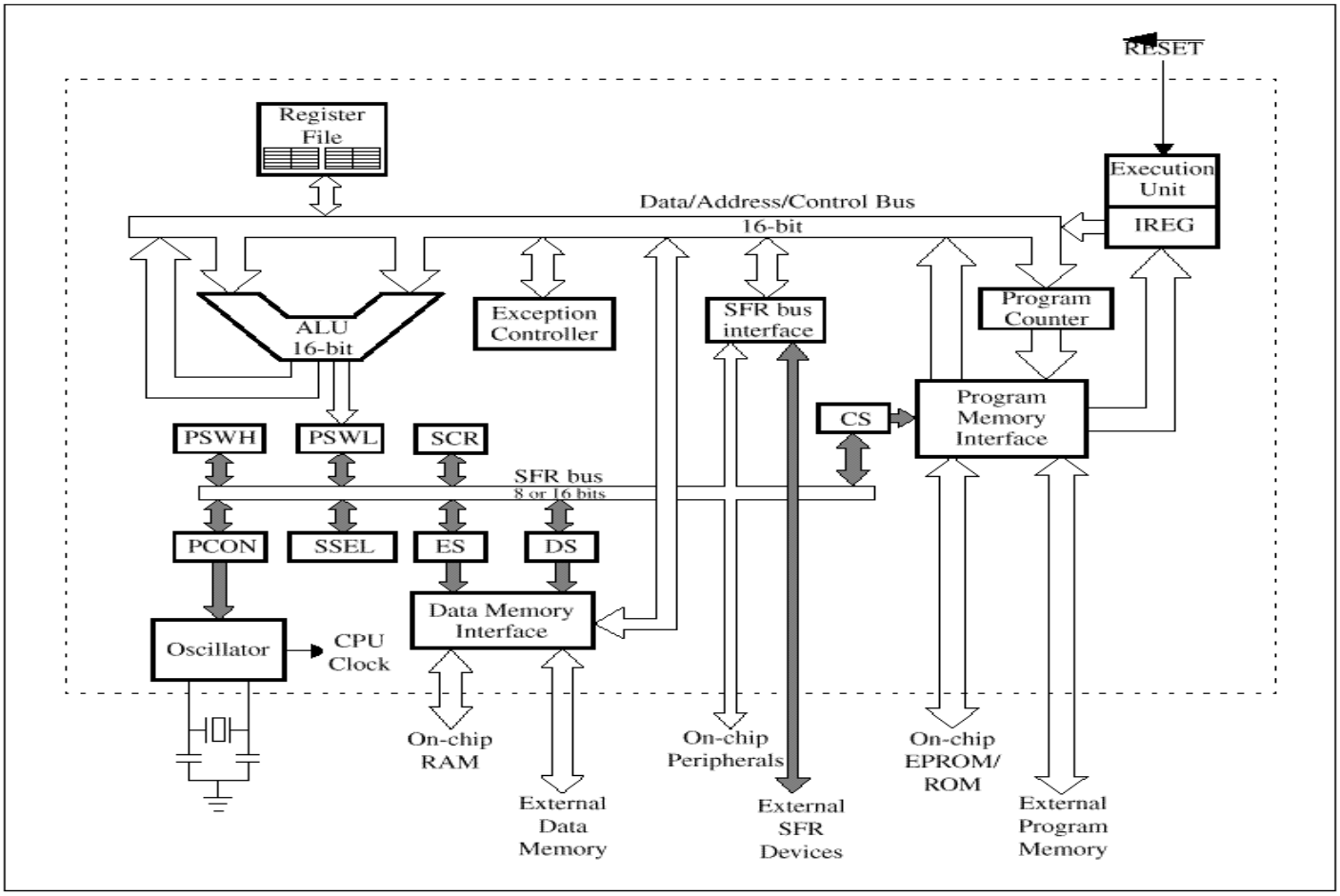


图 **4.1 XA**核

这里是XA的内部原理:XA核的震荡器提供基本的系统时钟.时序和逻辑控制由外部复位信号初始化;一一旦初始化完毕,这个信号提供内部和外部程序和数据的操作时序.这个逻辑引导调入程序计数器,通过程序接口把读取的指令放入指令寄存器中.时序和控制逻辑按顺序通过数据接口传入或传出数据.在同样的控制下,ALU执行运算和逻辑操作.ALU把状态信息放入PSW的低字节PSWL.片上的寄存器队列在中间暂时储存和保存当前的SP指针.PSW的高字节选择高权限的系统模式或有限制的用户模式;控制用于单步调试的跟踪模式,选择当前有效的寄存组,记录当前执行处理的限权.系统配置寄存器初始化后选择XA特有运行模式或80C51的兼容模式.XA核支持片内,片外的RAM,ROM/EPROM,和特殊存储器接口.这一章将详细描述这些要素.

4.2 程序状态字

程序状态字(PSW)是一个两字节的SFR寄存器,它是XA操作的焦点.它的低字节包含了CPU的状态标志,反映了CPU每个指令的操作结果.它们在用户模式和系统模式中都是可读写的.

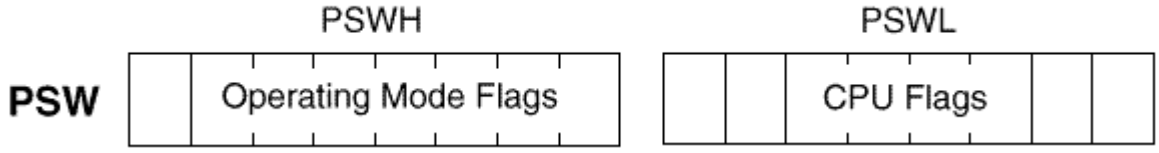


图 **4.2 XA PSW**

PSW的高字节由程序来写入,设定XA的操作模式和参数:系统模式/用户模式,跟踪模式,寄存器组选择位,任务执行优先级.**PSWH**对于任何程序都是可读的,但在用户模式下只有寄存器选择位是可改写的.所有的标志都可以由运行在系统模式下的代码改写.值得一提的是,XA包含一个仿照原始80C51的PSW寄存器.这个寄存器叫做PSW51,它完全兼容80C51代码对PSW的操作.详细的80C51兼容信息请见第9章.

4.2.1 CPU状态标志

PSW的CPU标志(图4.3)能显示进位,辅助进位,溢出,负数,和零.一些指令能改变这些标志,另一些只能改变其中的一部分,还有一些不会影响到这些标志.通常,程序读取这些标志,以决定程序的逻辑流程.第6章详细的描述了每一个指令是如何影响这些标志位的.要想详细了解每个指令对PSW状态标志的影响,请见第6章.

图**4.3 PSW CPU** 状态标志

C, 进位标志,通常影响算术和逻辑运算的结果.它包含算术运算的高位进位,主要是以下指令 ADD, ADDC, CMP, CJNE, DA, SUB, and SUBB. 进位标志也通常在以下指令中作为bit位的中间量,ASL, ASR, LSR, RLC, and RRC.乘法,除法指令将无条件删除进位标志(MUL16, MULU8, MULU16, DIV16, DIV32, DIVU8,DIVU16, and DIVU32).

AC, 辅助进位标志,是一些指令(ADD,ADDC, CMP, SUB,SUBB)的低字节进位标志.当使用十进制调整指令(DA)时,该标志可用于支持BCD运算).

V 是溢出标志**.**

(1) 当执行下列运算(ADD, ADDC, CMP, NEG, SUB, and SUBB)的运算溢出置位.

(2) 当执行除法指令(DIV16, DIV32, DIVU8, DIVU16, DIVU32),如果结果超过了目标寄存器的尺寸和除以0,V将置位.

(3) 当执行乘法运算(MUL16, MULU8, MULU16),如果结果超出了源操作对象的尺寸,也会置位.在这种情况下,V标志通知程序结果比缘操作对象要大,两个int型相乘,得出长整型

N 反映了算术运算和数据移动如果出现负数.PUSH, POP,SEXT, LEA和XCH指令不会影响这个标志位.

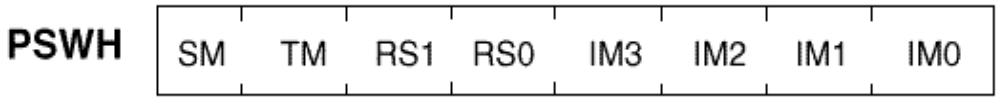
Z 它反映算术运算和数据传输的结果.如果结果为0,该位置位;如果不为0,该位清零.PUSH, POP, SEXT,

LEA,XCH指令不会影响到该位.

在寄存器图中,标记为"-"的位是为将来使用的.当向含有保留位的寄存器写入数据时要注意.因为为了防止以外激活为将来保留的功能,这些位已经赋予了0.

4.2.2 操作模式标志

PSW的操作模式标志能设定不同形式的XA操作模式.在高字节的PSWH中,除了RS0 RS1外,其余的位只能由运行在系统模式下的程序修改.



系统模式位,SM,当该位被设定后,允许当前的程序以全系统模式对XA的寄存器,指令,和存储器进行操作.(大多数的PSWH位只有在SM=1时才允许被改写).当该位被清除时,XA运行在用户模式,对当前运行的程序来讲,一些权限是被禁止的.

跟踪模式位,TM=1时,将启动内建的调试设备,见节4.9的描述.当TM=0时,调试特性关闭.

RS0 RS1从R0到R3的寄存器组中选择一组为当前的有效组.其余的组是不可操作的(请见系统配置寄存器节的兼容模式描述).

IM3到IM0(中断屏蔽位),决定当前执行程序的优先级别.当中断时间发生后,中断控制器把IM的值同中断的优先级做比较,以决定是否启动中断例程.IM的bits如果为0,表明是低优先级别,或者说是可中断代码.如果数值为15(16进制为F)表明是最高级别,不能被中断时间中断.注意:优先级别为15也不能禁止异常和NMI(非屏蔽中断)的发生.
IM的位只能由运行在系统模式中的代码改动.它们的值也可以由中断处理代码读出,以实现以软件为基础的中断优先级.注意:简单的改写中断屏蔽位可能引起优先级倒置,当前运行的代码优先级可能比先前中断的优先级低.XA的一些型号中包含软件中断,能避免混合系统的优先级倒置.详情请见有关软件中断的章节.

4.2.3 程序对PSW的改写

PSW由PSWH和PSWL构成,按SFR操作,如果一个指令的运行也修改PSW的位,那么它对PSW的修改就有潜在的不明确性.XA通过给予一个写全优先级别来解决这个问题.
举例, 运行
MOV.b R0L,#81h
由于传输的字节为两个负数,所以PSW的**N**为1.
但是如果执行
MOV.b PSWL, #81h
将会使C=1,Z=0,N=0,而不会使N=1.这是由于明显写入PSW的操作将有优先权.这种优先权将掩盖PSW实际意义上的更新.当一个新值写入PSW时,例如
OR.b PSWH, #30
PSWL的数值不会变化.

4.2.4 PSW初始化

如下所述,在XA复位后,PSW将从程序地址0处(复位矢量)调入初始化值.Philips建议初始化矢量应该把IM3到IM0设置为1,于是初始化将被屏蔽成高优先级(只有异常和非屏蔽中断可以运行).初始化代码运行完毕后,运行的优先级一般应该减小,通常为0,允许别的任务运行.也建议复位矢量将SM设置为1,这样可以运行在系统模式

4.3 系统配置寄存器

系统配置寄存器(SCR),在节4.5有描述,设置XA的全程操作模式.在系统启动后,应该在SCR中写入合适的

数值.SCR现在定义了4个bits

图**4.5** 系统配置寄存器**(SCR)**

PZ 设置为0,将使XA工作在大模式下,使用全部24条地址线.

设置为1,将使XA工作在小模式下,只使用16条地址线,也就是Page0模式.使用Page0模式可以节约堆栈空间和改进中断响应.详情请见下面的章节.

CM 选择标准的XA特有模式或80C51兼容模式.当选择80C51兼容模式时,将发生两件事.首先,每个数据段的底部32字节将寄存器阵列中的4个寄存器组取代,0组的R0L放在地址0,0组的R0H将防在地址1,等等.第二,R0和R1的指针用法改变.为了模仿80C51的间接寻址,间接引用R0将只使用R0L(R0其余的位将是0)作为实际的指针.间接引用R1将只使用R1L(R1其余的位将是0)作为实际的指针.注意这时XA中R0L和R1L等同与80C51的R0和R1.其它的XA特性不受兼容模式的影响.前面8章讲述的是非80C51兼容模式(CM=0).第9章讲述的是80C51兼容模式(CM=1).

PT1和**PT0** 选择震荡器的约数作为外设的时钟源,特别是记时器,但也可能供给某些XA器件中的一些外设.下面是这些bits的取值和对应的时钟频率.

PT1	PT0	外设时钟
0	0	晶振频率/4
0	1	晶振频率/16
1	0	晶振频率/64
1	1	保留

在寄存器图中,标记为"-"的位是为将来使用的.当向含有保留位的寄存器写入数据时要注意.因为为了防止以外激活为将来保留的功能,这些位已经赋予了0.

4.3.1 XA大模式描述

当通过SCR选择XA的缺省模式时(**CM** = 0和 **PZ** = 0),所有的地址都是24bits,提供16M的地址空间.在有些型号的XA中,外部接口提供的地址少于24bits.在调用和中断时,所有的24bits将被压入堆栈,执行RET和RETI时,24bits地址将被弹出.

4.3.2 XA Page0模式描述

当选择Page 0模式时,XA核只保持16bits的地址.如果在应用中64K地址满足要求,就可以使用这种模式.高8位的外部总线接口可以另作它用.当调用和中断时,只有16bit的字压入堆栈,RET和RETI也只有16bits弹出.使用Page 0 可以节省堆栈空间,加速地址在堆栈中的弹入,弹出.

不推荐在原始启动后进入或推出Page 0模式.首先.进入Page 0必须由运行在Page 0的程序来完成,这是因为程序地址在Page 0起作用时要被截短.在XA予读取指令队列中的指令也必须在Page 0起作用前读取完毕.所有压入堆栈的地址在Page 0起作用后将失效.于是在发生中断时和执行子程序时不能进行Page 0切换.

4.4 复位

"复位"名称特别适用于当第一次加电到XA器件时的硬件输入,一般指硬件初始化时序.可以发生在任何时间.但是也同样适用于复位指令"RESET";另外.看门狗定时器的溢出也有同样的效果.

这一节将描述复位时序和对用户软件和硬件的影响.

4.4.1 复位时序综述

XA器件加电时,指定的硬件复位时序必须由硬件来完成,之后程序才能开始运行.如果加电时没能进行合适的复位,XA可能全部或部分工作失败.XA的复位过程包含以下元件:

外部元件产生复位信号.

内部复位时序发生.
RST端变高.
外部总线宽度和存储器配置确定.
复位异常中断产生.
执行开始代码.
图4.6 显示了这个过程.

4.4.2 加电复位

这一节将讨论XA器件的加电复位时序.
在Vdd加到XA器件上,并保持稳定后,XA的RST输入必须保持低电平,并维持一个最小的复位周期.这个最小的周期为10毫秒.在这个周期内,XA的震荡器启动并稳定,CPU检测出复位条件.这时,CPU启动内部复位时序.RST必须保持足够的低电平时间,使内部的复位时序能够完成.

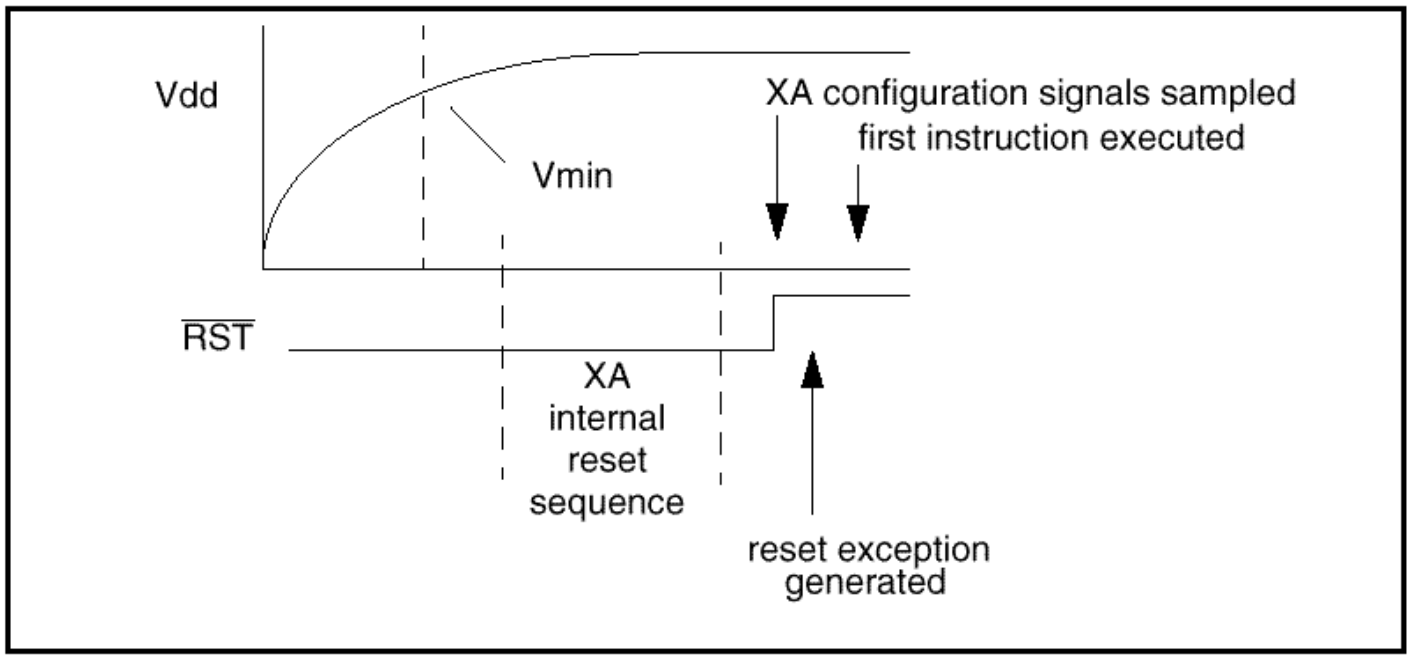


图**4.6 XA**的加电时序

4.4.3 内部复位时序

XA的复位时序发生在加电,或运行时在RST端施加有效时间的复位脉冲.这个时序需要10微秒的时间(10个时钟周期,或更长)结束,在这个时间内RST必须保持低电平.内部复位时序做以下工作:
•对多数核和外设寄存器写0,另一些外设的SFR可能写入别的数值,详情请参考数据手册.
•对CS DS ES清零.
•SSEL=0,设置通过ES进行操作.
•选择寄存器组0.
•设定用户和系统堆栈为0100H.
•清除SCR的PZ,使用24-bits的地址线
•清除SCR的CM,进入XA的特有模式.
•清除EA,关闭所有的可屏蔽中断.
注意:内部复位时序不会初始化内部或外部RAM.PSW的值在这时也不会处理,在下面的描述中PSW将会被替换.
内部复位时序对XA核外部的部分的影响取决于外设和指定XA型号的配置.一般来讲,内部复位有以下影响:
•I/O口设置为输入(准双向口输出配置为FFH)
•SFR大多数清零.
•其它的SFR大多数设置到相应的非零数值.
注意:串行口缓冲器,PCA捕获寄存器,和看门狗寄存器(如果提供)不受影响.详情请见XA的数据手册.
在XA内部复位时序完成以后,器件静止,直到RST端变高.

4.4.4XA复位后的配置

当RST端变高以后,XA的两个输入脚被取样以决定XA存储器和总线的配置.EA和BUSW在复位时序中有着特殊的功能,允许外部硬件确定使用内部或外部存储器,和选择缺省总线的宽度.
在RST变高以后,CPU触发一个复位中断,见下一节所述.

选择内部或外部程序空间

XA可以从内部或外部读取指令.XA的EA端决定使用内部或外部存储器.EA脚的电平在RST的上升沿被取样.如果EA=0,XA将使用外部程序空间,否则使用内部程序空间.这种选择一直保持,直到下一次RST的到来

XA不会检查EA的检查值和实际的硬件存储器区域配置是否不一致.举例,在ROMless的XA型号中设置EA=1,将会使XA执行内部的代码,实际上不存在,这样会引起系统崩溃.

选择外部总线宽度

XA能操作8位或16位的数据总线.脚BUSW告诉XA外部总线的配置.BUSW=0,选择8-bits总线而BUSW=1选择16-bit总线.加电后,XA的缺省值是16-bit总线(这是由于BUSW的微弱上拉).BUSW脚的电平在RST的上升沿被取样.总线的宽度也可以由软件控制总线寄存器BCR来实现.,

RST释放后,BUSW脚可能具有另外的功能.详情请参考数据手册.

4.4.5 复位异常中断

一端RST端电平变高,CPU将产生一个复位异常中断.从地址0处的复位矢量中调入起始PSW值和起始指令的地址.下面是汇编格式的复位初始化程序.

```
code_seg                ;建立代码段
org 0h                  ;由地址0处开始
                        ;以下是复位矢量
dw initial_PSW          ;定义一个字常量
dw startup_code         ;定义一个字常量
org 120h                ;开始地址从120h起,由上述矢量表决定.
startup_code:
                        ; 这里放入启动代码.
```

复位矢量中的PSWL的设定对于系统并不重要,可以设置成0或需要的值.初始化的PSWH是软件初始化的基础,它的取值必须小心.这里的例子设置了建议的PSWH初始化值.

```
system_mode     equ   8000h
max_priority    equ   0F00h
initial_PSW     equ   system_mode + max_priority
```

通常应该把XA初始化成系统模式,这样启动代码就可以无限制的操作全部的XA资源.这需要设定PSWH的初始化bit位SM. Philips建议把启动代码的优先级初始化成最高级15(IM0到IM3全部为1),这样启动代码将被认定为最高级别的处理.如上所述,硬件初始化时序将关闭所有可能的中断,只有非屏蔽中断(NMI)可能成为潜在的中断.通常是用硬件锁住,直到启动处理完成.下面的例子中,初始化的PSWH值将设定系统为系统模式,选择寄存器组0(也可以选择其它),TM=0,关闭跟踪模式.

4.4.6 启动代码

Philips建议启动代码的第一条指令是设置系统配置寄存器SCR的值,在节4.3中描述,以符合系统的结构.下一个建议的步骤是正确的设定堆栈指针.缺省的值通常是不会满足实际的要求的.启动代码时序应该包含一个简单的分支和跳转到应用程序中.在复位中断处理完毕后不能使用RETI指令,因为复位是初始化SP,而不是产生中断堆栈结构.

4.4.7 复位同XA一些子系统的相互作用

下面将叙述复位处理是如何同一些关键子系统相互作用的。

■跟踪异常中断.外部复位将终止跟踪异常中断,见节4.9
■看门狗.在配备看门狗的XA型号中,内部复位将根据不同型号来确定时钟脉冲个数。
■处于正常运行或空闲状态的复位.在空闲模式下XA的震荡器仍在工作,因此RST输入必须保持10微秒的时间(或10个时钟周期),以获得完整的复位.
■在关闭状态下的复位.由于在关闭模式下震荡器已经停止工作,所以RST输入必须保持10毫秒.一个例外是使用了外部震荡器.在这种情况下,如果外部的震荡器仍然在工作,那么在关闭模式下的复位同在空闲模式下的复位是相同.

4.4.8 外部复位电路

RST脚是高阻施密特触发输入.在应用中并不需要特殊的启动需求,实际上它产生的复位周期比电源的上升时间都要长.简单的制作一个复位电路如图4.7

4.5 震荡器

XA包含一个片上的震荡器,可以作为XA CPU的时钟源,也可以使用外部时钟.如果使用内部震荡器,石英晶体或陶瓷共振可以按照图图4.8a连接.如果使用外部时钟,可以接到XTAL1脚上,XTAL2悬空,见图4.8b

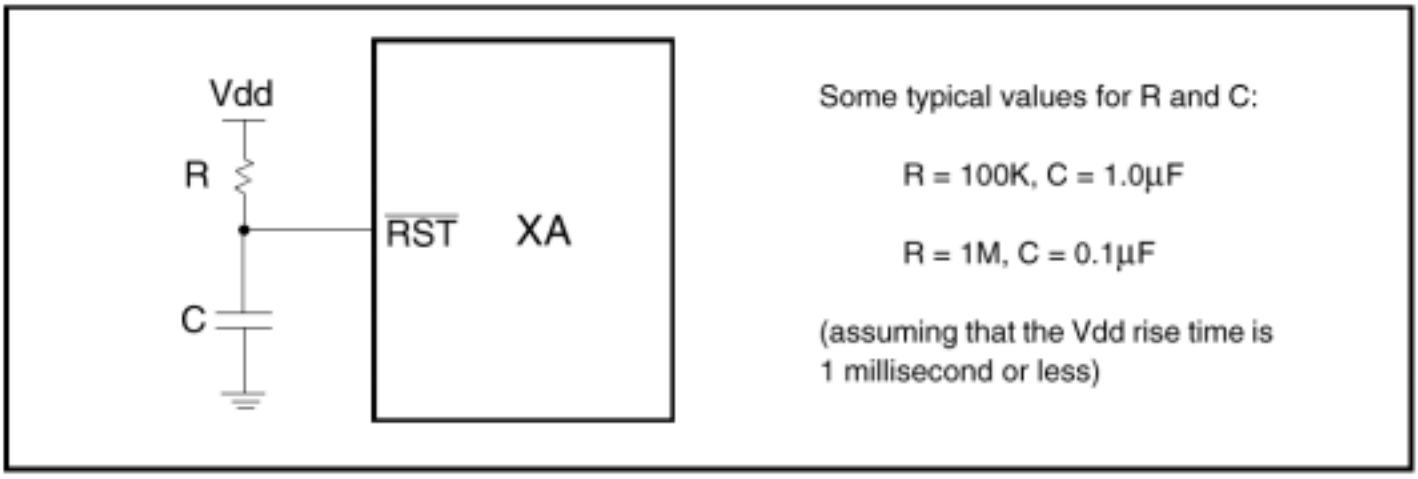


图**4.7** 外部复位电路范例

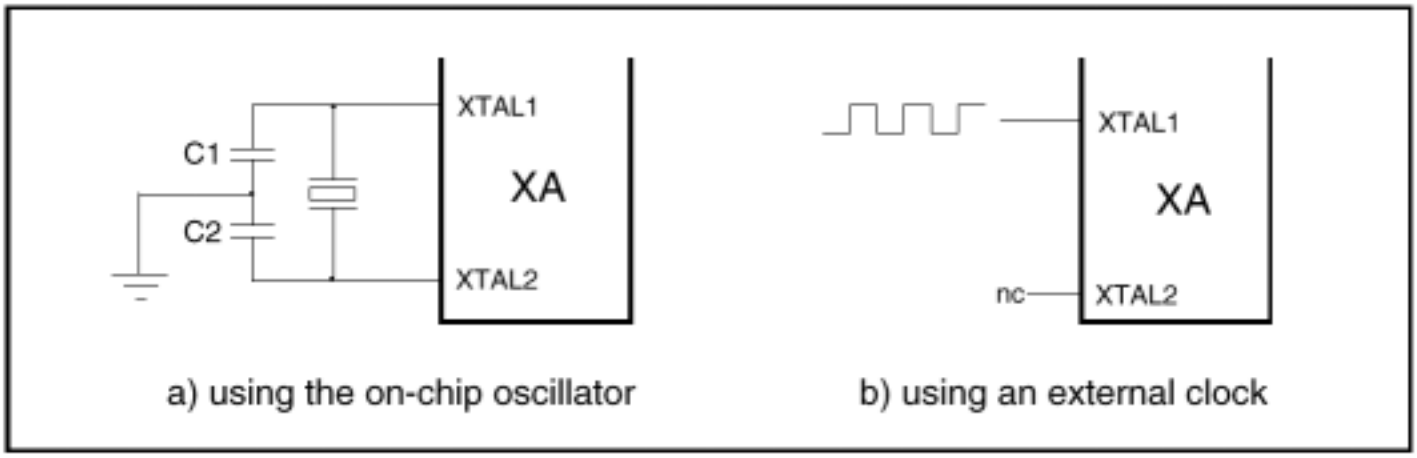


图**4.8 XA** 时钟源

XA片上震荡器包含一个单个反向器以便形成正电抗震荡器.在这种应用中,晶体工作在基频模式下,如同一个电感同外部的电容形成并联震荡.石英晶体或陶瓷共振器连接到XTAL1和XTAL2两端,电容连接到每一个脚和地.如果使用石英晶体,应该使用并联共振的晶体,以便获得可靠的性能.电容的数值应该按照晶体或共振器厂商的建议.对于晶体,25MHz以上一般在18p-24p,以下在28-34p.电容过大或过小将使震荡器不能启动或影响启动时间.

4.6 功率控制

XA有两种模式减少功率消耗:空闲模式,中等程度的功率节约;关闭模式,将XA的功率消耗减少到最小程度.这些模式的实现是靠写入寄存器PCON中正确的位来实现的,见图4.9

图4.9 PCON

空闲模式的启动是置PCON的位IDL=1,CPU将停止运行, 晶振和其它外设仍然工作.
关闭模式的启动是置PCON的位IDL=1,XA将关闭,震荡器停止工作.
复位后**IDL**和**PD**的值为0.如果这两位同时被写为1,PD将优先,XA进入关闭模式.
其它的位是为将来的使用而保留的. 在寄存器图中,标记为"-"的位是为将来使用的.当向含有保留位的寄存器写入数据时要注意.因为为了防止以外激活为将来保留的功能,这些位已经赋予了0.

4.6.1 空闲模式

空闲模式停止程序的运行,但震荡器和外设还继续工作.这样能很大的减少XA的消耗.如果中断允许,那么在外设产生中断时,将使处理器从停止的地方重新执行.在空闲模式下,I/O脚保持进入空闲状态以前的状态.作为外部总线的脚将恢复它们的锁存和配置值(通常是推拉输出,总线相关脚为1).ALE和PSEN保持在它们无效的状态.如果正常推出空闲状态(通过有效的中断),口的数值和配置将恢复到原来的状态.

4.6.2 关闭模式

关闭模式停止程序的运行,关闭片上震荡器.这将停止XA所有的动作.内部寄存器,SFR和内部RAM将保留.进一步降低功耗要降低Vdd(内部RAM的保持电压).参考数据手册中应用中Vdd的值.处理器的重新启动要靠加入复位信号或一个开放的外部中断.当处理器重新启动后,震荡器重新启动,程序从停止的地方重新执行.在关闭模式,ALE和PSEN输出保持在它们无效的状态.口的输出值保持它们的SFR值.这样,不是外部总线的口将保持它们的状态.对于是外部总线的口将变化到原来的锁存和输出值(通常是推拉输出,数值为1).当复位时已经处于关闭模式,全部的口的数值和配置恢复到缺省的复位状态.当XA处于关闭模式时,为了使用外部中断唤醒XA,外部中断必须开放并配置成电平触发模式.当XA处于关闭模式,并通过外部中断唤醒,口的数值和配置将保持原来的状态.XA在由外部中断唤醒关闭模式时,必须有足够的时间允许震荡器重新启动.XA不要求外部中断逻辑保持足够的时间以等待震荡器启动,XA有自己的记时器保证合适的唤醒.经过大约由9892个震荡周期,XA才允许程序执行,这时认为震荡器已经运行并稳定.

注意:如果使用外部时钟,那么在关闭模式下电源电流的减少量要比同等条件下使用片上时钟要少.在这种情况下,要获得全功率节省必须关闭外部时钟或断开通向XA的XTAL1脚的信号.如果外部时钟可以关闭,使用空闲模式比关闭模式跟有利,它能提供结束多种途径结束功率节省模式,避免退出关闭模式的9892个时钟的等待时间.

4.8 XA 中断

XA结构定义了4种类型的中断.下面把它们按优先级排列.
异常中断
事件中断
软件中断
陷阱中断

异常中断反应的是重要的系统级别事件.例如堆栈溢出,被0除,和不可屏蔽中断,它不管当前运行代码的优先级.

事件中断反映的是非决定性的硬件事件,如串行口要求处理,或记时器溢出.事件中断一般是片上的设备和外部中断.只有事件中断的优先级大于现在运行代码代码的优先级时,才能得到处理.事件中断的优先级可以由软件设置.

软件中断是事件中断的扩展,但是它是由软件在SFR中设置需求位而产生的.同样,软件中断的优先级必须大于现在执行代码的优先级才能得到执行.软件中断的优先级是固定的,从1到7.

陷阱中断的执行是因为执行了TRAP指令.所以当指令执行时将获取陷阱中断矢量.

所有形式的中断将触发同样的时序:首先,包含下一个指令地址的数据和当前的PSW将压入堆栈.包含新的PSW和新的处理地址从代码区中读取.新的PSW值替换旧的PSW值,并从新的地址开始执行指定的中断处理
新的PSW可能包含新的SM设置,允许处理程序运行在系统模式或用户模式,新的IM3到IM0,表达了新任务的优先级.这些性能是XA对多任务的基本支持.详情请见第5章.

从中断返回大多数是依靠执行RETI命令完成的,它从堆栈中弹出数据,执行新的PSW值.下面将描述,在用户模式下执行RETI指令将会引起一个异常陷阱,所以中断程序通常是在系统模式下运行的.

XA结构具有优良的机制以确定何时和是否一个中断时序真正的发生.下面将描述,异常中断触发后就将执行.事件中断将延迟到它的优先级大于现在执行代码的优先级.对于异常中断和事件中断,将分类处理同时发生的中断.软件中断和陷阱中断是执行指令后产生的,没有解决冲突的问题.不可屏蔽中断需要认真的对待.它是由XA核外产生的,是一个事件中断.但是,它具有很多异常中断的特性,,由于它是不可屏蔽的.注意NMI,在所有的XA器件中,它并不是总连接到某一个管脚上或其它中断源.

4.8.1 中断类型的详细描述

这一节将详细描述这4种类型的中断.

异常中断

异常中断反映的是非常重要的事件,必须马上处理.XA核中的异常中断有:复位,断点,被0除,堆栈溢出,从用户模式的中断返回,跟踪,和保留的9个附加中断.NMI也列在下面的异常中断表(表4.1)中,这是由于XA核处理NMI的方法和优先级类似于异常中断.由于异常中断被定义为不可屏蔽,它总是被立即执行,而不管现在执行代码的优先级(PSW中IM位定义).如果在非正常情况下,同时有多于一个的异常中断发生,就要由硬件来排序,它决定那一个异常矢量将被获取.在这种情况下,获取的下一个中断矢量的优先级是最高的,它的代码将首先执行.即使正在执行另一个异常中断的处理程序它也会被立即执行.

在编写异常处理时,程序员要注意以下事项:

(1) 由于另一个异常会引起堆栈溢出异常处理程序,所以所有的异常中断处理程序代码不要破坏堆栈溢出.记住堆栈溢出异常仅仅在大于80H穿越到小于80H时刻发生(小于80H并不会发生).

(2)断点(由执行BKPT指令引起,或仿真系统的硬件断点)和跟踪是相互独立的.在两种情况下,处理代码都要知道异常中断发生时用户代码的地址.如果断点发生在跟踪模式中,或者在执行断点处理代码时跟踪模式是打开的,两种中断处理都可以堆栈中看到另一个处理代码的返回地址.

表**4.1:**异常中断种类,矢量,和优先级

表述	矢量地址	服务优先级别
断点Breakpoint (h/w trap 1)	0004-0007	0
跟踪Trace (h/w trap 2)	0008-000B	1
堆栈溢出Stack Overflow (h/w trap 3)	000C-000F	2
除以0Divide by 0 (h/w trap 4)	0010-0013	3
用户返回User RETI (h/w trap 5)	0014-0017	4
保留	0018-003F	5
非屏蔽中断(NMI)	009C-009F	6
复位Reset (h/w, watchdog, s/w)	0000-0003	7 总是立即执行,停止其它中断

事件中断

事件中断一般同片内或片外的外设有关,对XA来讲是异步发生的.XA核本身不包含事件中断源,所以事件中断是由片内的中断控制单元来处理,而不是片外.

在一般的XA器件中,事件中断由片外外设发出,并施加到XA的事件中断检测输入脚.全部的事件中断可以通过中断管理寄存器IE内的EA位来关闭,也可以通过IE内的相关位或它的附件分别的加以屏蔽.当一个外设的事件中断被关闭后,但外设并没有关闭,所以外设中断的标志仍可以被外设置位,中断开放后中断将会发生.如果一个开放的事件中断的优先级比现在运行代码的优先级高,中断将被响应.每个事件中断在中断优先级寄存器中规定了它的优先级别.如果同一时间内有多个事件中断发生,优先级设置将决定先响应那一个.如果在同一时间内发生的中断具有相同的优先级,则由硬件优先级方案决定.XA结构定义了15个中断优先级,可以在中断优先级寄存器中定义.注意,一些XA的型号不支持15个中断优先级,详情请见数据手册.

注意:像其它中断的形式一样,PSW(包括中断屏蔽位)在事件中断响应时将从矢量表中调入.这样,中断服务程序的优先级可能同中断的优先级不同(因为中断服务程序的优先级是从中断矢量中调入的,而中断的优先级是有优先级寄存器决定的).通常,建议将二者的优先级设置成一致.另外,中断发生的优先级一定不能大于执行中断的优先级.否则将会产生无穷的中断嵌套,因为刚进入中断执行,就会被相同的中断源中断.

软件中断

软件中断类似与事件中断,只不过它是由软件设置SFR中的中断请求位.标准的软件中断实施结构提供7个中断,与2个SFR有关联.一个是软件中断请求寄存器SWR,有7个请求位同每一个软件中断相关.另一个SFR是启动寄存器SWE,,也有7个启动位同每一个软件中断相关.软件中断的启动是通过设置SWE中的请求位来实现的.如果SWE中相应的启动位也被设置,则在它成为中断中的最高优先级并且高与当前的执行代码,软件中断就会发生.在软件中断服务程序结束以前,SWR中的软件中断请求位必须由软件清零.软件中断的优先级是固定的,由1到7.如下面的图4.2所示.软件中断是定义在XA核的外部,不是所有的XA器件都能提供;详情请参照数据手册.

软件中断的原始用途是提供一种组织方法,当中断服务开始后,其中一部分服务程序可以工作在较低的优先级水平上.例如,为了快速响应要求严格的外部事件,一个中断程序可以设定一个教高的优先级.这个中断申请有部分代码需要立即执行,而更多部分的代码不需要立即完成.如果ISP的不重要部分用低优先级别来执行,允许其它响应其它重要的ISP,系统的总体性能将会得到改善.

如果一个高优先级ISR,在进入服务程序后,向IM位写入低优先级的数值,简单地降低它的执行优先级,可能产生一种叫做"优先级倒置"的现象.当"优先级倒置"发生后,低优先级的程序在执行,而高优先级的程序却在等待.图4.15展示了这种现象:

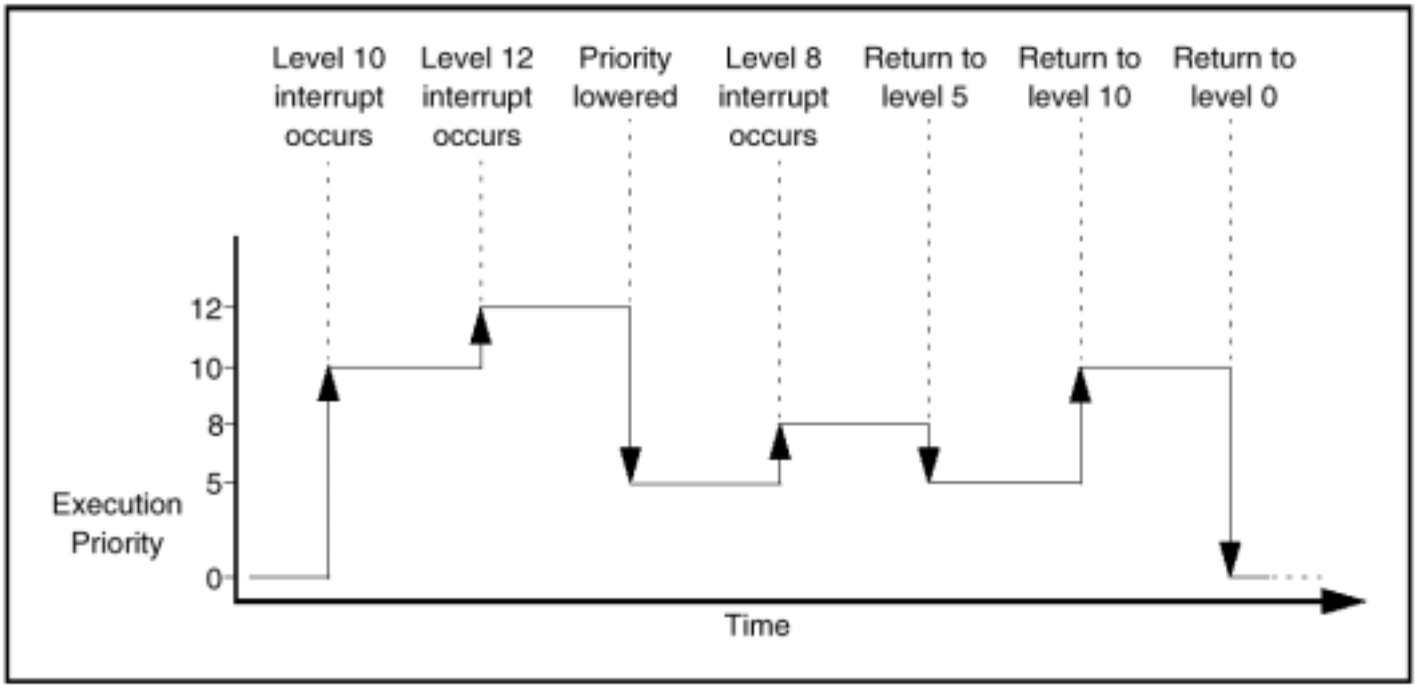


图 **4.15** 优先级倒置示例(见文字说明)

假设代码执行在优先级0,被一个事件中断,中断程序执行在优先级10.它又被一个优先级12的中断中断.优先级为12的中断完成了对时间要求严格的代码部分,它想降低其余代码(对时间无严格要求部分),以便允许更重要的中断发生.于是,它向IM位写入数据,设定执行优先级为5.优先级为5的ISR继续执行到它被优先级为8的事件中断.优先级为8的ISR完成并返回到优先级为5的ISR,优先级为5的ISR继续执行完成,原先优先级为10的ISR才重新启动并最终完成.从上面的例子中可以看到,低优先级的ISR执行并完成,而高优先级的ISR却在堆栈中等待.这就叫"优先级倒置".
在这种情况下,当想改变ISR一部分的优先级时,可以使用软件中断来实现,就不会产生"优先级倒置".ISR必须分为两部分:高优先级部分继续保持原来的中断矢量,低优先部分用软件中断5来实现.当ISR的高优先级部分完成后,软件设置软件中断5的中断申请,返回,ISR的其余部分执行软件中断5,当变为中断级别的最高级后才开始执行.图4.16显示的事件时序同优先级倒置的例子完全相同,但使用了上述的软件中断.注意代码执行了正确的顺序(高优先级先执行).

陷阱中断

陷阱中断有TRAP指令产生.TRAP0到TRAP15可以在应用中定义和是使用.TRAP中断应用中指定的要求,可以作为进入通用程序的便利机制,允许在用户模式和系统模式之间转换.TRAP中断仅仅在TRAP指令被执行时发生,因此不必关心同时发生的优先级组合.TRAP的效果是立即的,执行完TRAP指令后,立即进入TRAP的服务程序.见第6章,有关TRAP指令的详细描述.

4.8.2中断服务的数据要素

有两个数据要素同XA中断相关.第一个是当中断被执行时堆栈框架的建立.另一个是中断矢量表,它定位于代码区域的开始部分.了解每一个的结构和内容,有利于了解XA中断是如何被执行的.

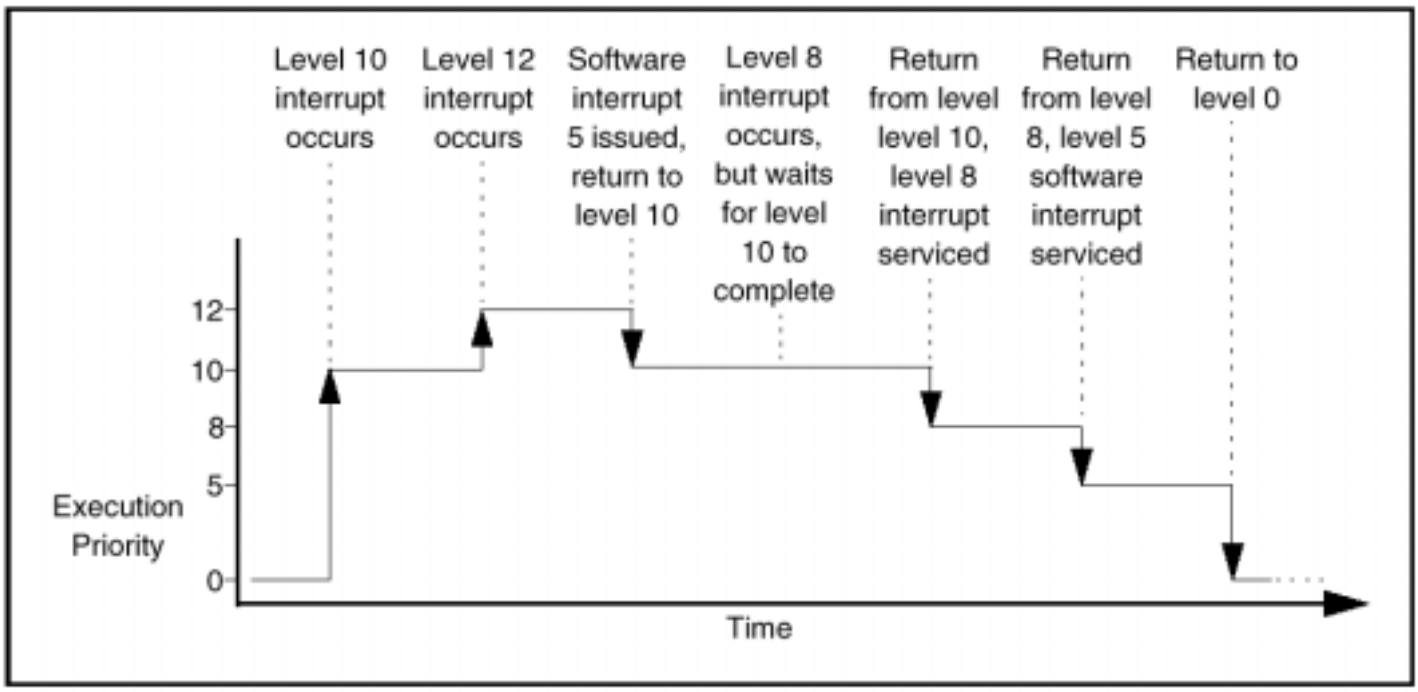


图**4.16** 软件中断的示例(见文字说明)

中断堆栈框架

每一个XA中断都会产生一个堆栈框架.对于异常中断,堆栈框架一直保存到中断服务程序返回被中断的代码,并还原CPU的状态.(Reset中断由复位事件触发.由于复位要重置堆栈指针,堆栈框架不会保存.详情见4.4节).XA原始的24-bit操作模式的堆栈框架见图4.17.这时三个字保存在堆栈中.首先入栈的是当前PC数值的低顺序16-bit,是下一条指令的地址.第二个字是地址的高顺序byte,高位用0填补.这样一个完整的24-bit地址就保存在堆栈中.第三个字是中断服务时当前PSW的数值.
当XA运行在Page 0模式(SCR.PZ=1),堆栈框架要小,因为在这种模式下,XA只使用16位地址.Page 0的堆栈框架示意图见图4.18.很明显,两种尺寸的堆栈框架不能混合;在4.3节,提醒在XA启动后立即设定系统配置寄存器,并在以后保持不变,也是这个原因.

中断矢量表

XA使用程序区域开始的284个字节(0-11B)作为中断矢量表.矢量表包含71个双字入口,每一个对应指定的中断事件.每一个双字入口包含中断服务程序的16bit地址和16bit的PSW替换数值.由于矢量地址是16bit,所以服务程序的第一条指令必须定位与XA的第一个64K.所有服务程序的第一条指令必须是字相邻.替换PSW的主要原因是为服务程序选择用户模式或系统模式,寄存器开关选择,执行优先级设置.详细的PSW元素,见4.22节

从程序地址0开始的16个矢量为异常中断矢量保留.第二个16个矢量为TRAP中断保留.下面的32个矢量为事件中断保留.最后7个矢量是软件中断.

图4.19显示了XA矢量表和组成结构.异常中断矢量,6个是XA特有的,10个保留.16个TRAP中断可以随便使用.分配给事件中断的矢量依赖于不同的器件,并随XA器件的外设配置和管脚而变化.不使用的中断矢量通常应该指向空的服务程序.空的服务程序应该清除中断标志(如果它不能自己清除)和执行RETI以返回用户程序.这对异常中断和NMI尤其重要,因为它们可能产生在用户不希望出现时产生.如果这些矢量指向空白的服务程序,则系统可以忽略这些不希望的异常和中断而继续执行.

注意在使用硬件开发工具时,它可能喜欢不初始化不使用的矢量位置,允许开发工具标出这些条件下不希望的事件.

4.9 跟踪模式调试

XA有一个可选的跟踪模式,每一条指令的结束将产生一个特定的跟踪异常中断.跟踪模式支持用户提供的调试/监视程序,可以单步运行整个程序,即使在ROM中.

4.9.1 跟踪模式操作

只要设置相关程序的PSW.TM,就能启动跟踪模式.使用跟踪模式必须对XA的指令运行时序详细了解,因为跟踪异常中断的发生与一个指令的执行时序有关.图4.20概括了XA的指令时序.

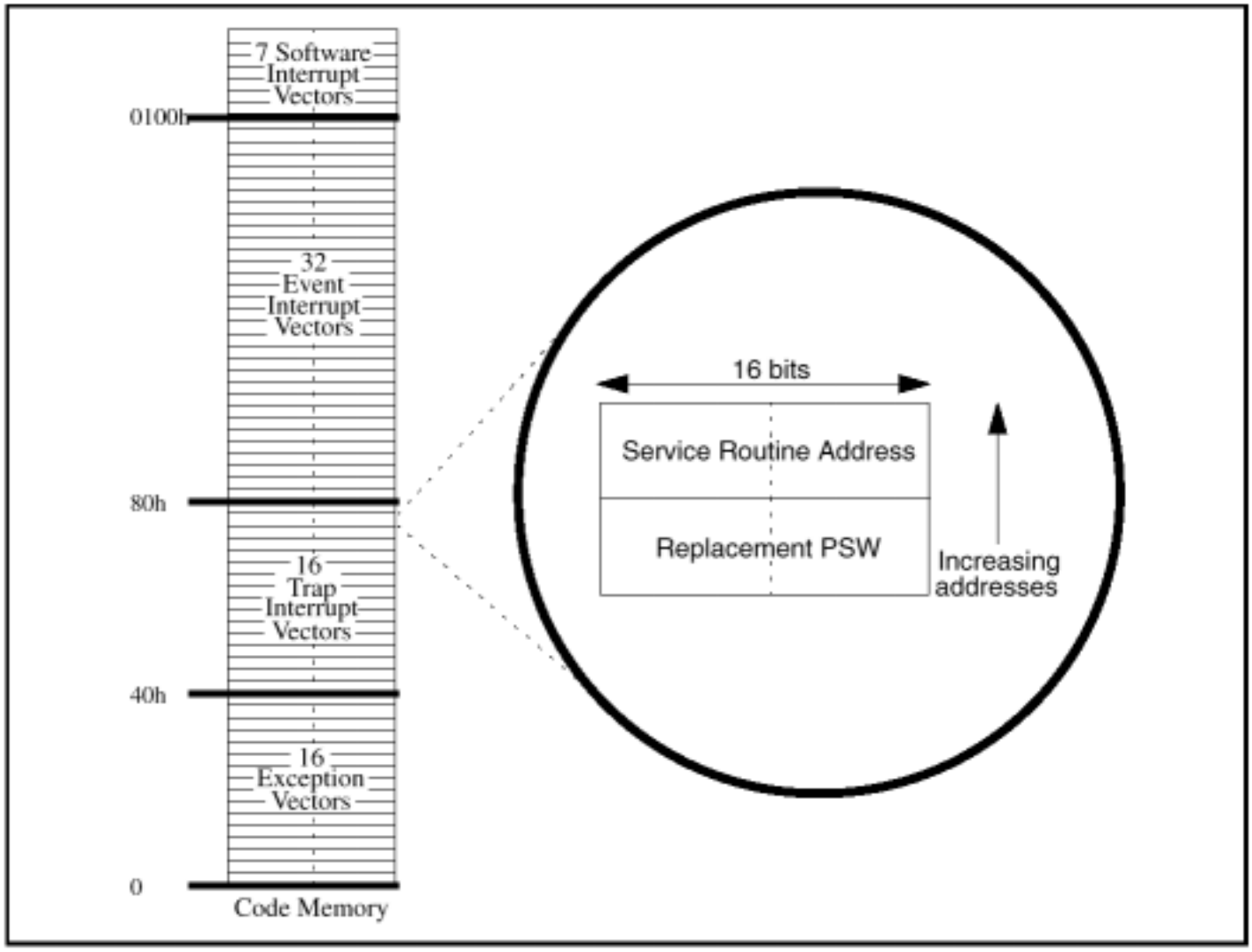


图**4.19**　中断矢量

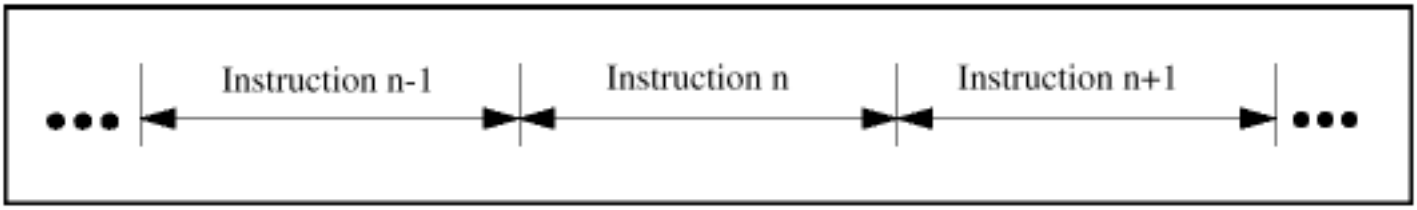


图4.20　XA指令时序略图

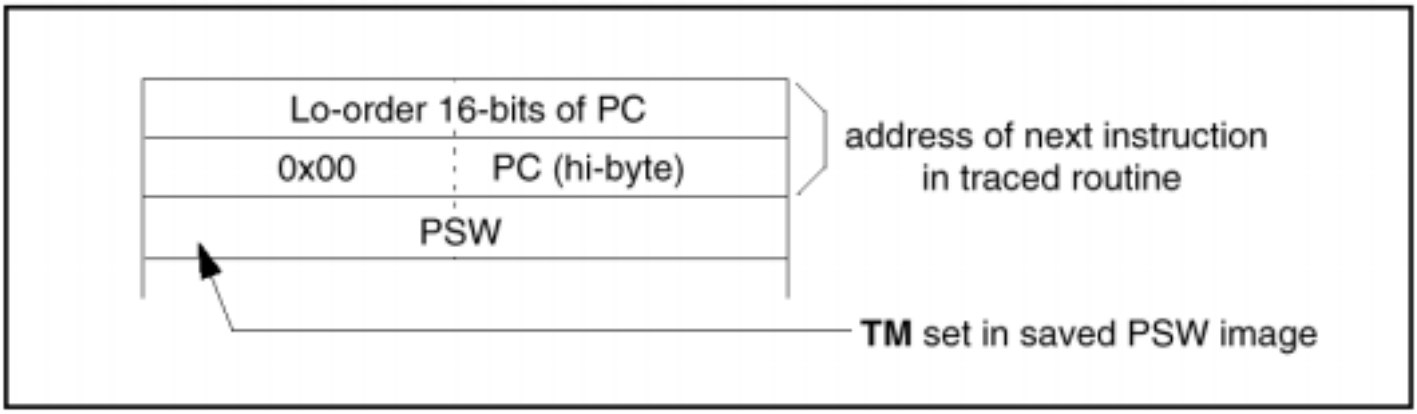
图4.21是时序的详细模型:首先,在指令周期的开始,TM信号被锁存.下一步,指令被检查是否有效;没定义的指令和不允许的操作(象在用户模式下通过ES写)将不执行,不会产生跟踪.然后时序时序检查指令在当前设置下的合法性(当前仅有在用户模式下的IRET),如果有异常中断则执行.如果,仅为如果,没有上述条件发生,指令才会真正的执行.执行完毕后,如果在指令周期开始锁存的跟踪模式位是1,则开始执行跟踪程序.最后,时序检查中断,如果有,则执行.

注意:外部复位可能发生在图4.21的任何位置.如果发生,则会中断处理.

这个时序的例外是指令如果将设置TM=1,将不会产生跟踪,这是因为指令实际执行时TM不会锁存.另一个例外是如果指令产生一个异常中断,则不会被跟踪.最后,如果指令在执行TRAP指令并发生了事件中断,则TRAP将被执行,然后执行跟踪,最后是执行中断.

4.9.2 跟踪模式的启动和停止

由于PSW.TM是PSW的保护部分,只有系统模式下的程序代码才能启动和关闭跟踪模式.在应用中,可以调用一个TRAP,替换的PSW将清除该位,或执行一个RETI指令人为形成一个异常中断,在这以前将需要的堆栈框架压入堆栈顶部,如下:



跟踪模式在TM位清除以前将持续运行.如果想停止跟踪模式,可以在跟踪模式服务程序中检查系统堆栈顶部的堆栈框架,在返回被跟踪的程序之前清除TM位.也可以用类似的方法启动跟踪模式.注意由异常中断产生的堆栈框架总是放在系统堆栈中.在修改TM位时,跟踪服务程序最好检验一下堆栈框架的内容是否同被跟踪的处理程序一致

5 实时多任务

多任务,望文生义,允许多个执行一定功能的部分程序代码,即任务同时运行.这意味着多个任务在同一时间内执行,同时执行不同的工作.

在一些高端应用领域(如汽车)需要对一些高速事件立即响应,如发动机管理,牵引控制和ABS防抱死系统,因此在很多高性能的嵌入控制应用中,倾向与使用多任务系统.

实时应用系统通常由多个任务组成的.每一个任务处理一个指定的应用程序.建立一个实时系统能把一个复杂的系统程序细分成独立的.容易管理的模块.每个任务根据分配给它们的优先级同其它模块一起分享处理器.

在实时多任务系统中,高端系统是主要的.切换任务包括终止和启动任务时的数据转移,必要时唤醒任务的大量数据.因此应该尽量减小高端系统.在一些情况下,一些任务对实时系统的响应时间有要求,这使系统就更加复杂化.

下面章节分析这些要求和XA在这些应用中的适应性.

5.1 XA对多任务的支持

XA在多个方面支持多任务系统.XA在结构上直接支持多任务操作系统,它提供了两级权利(系统和用户)用于不同任务之间的隔离.高操作性能,中断驱动,多任务应用系统需要的保护在XA中都是可以实现的.XA结构提供以下特性满足多任务的实现:

5.11 双堆栈指针

XA结构定义了系统指针(SSP)和用户指针(USP).双堆栈指针支持快速的任务切换,简化多任务监视核的建立.在切换到多任务核和返回应用系统时,能减少堆栈指针的存储和恢复过程.它也可以在大的系统中使用外部数据存储器来加快中断处理.用户堆栈可以定位于较慢的外部存储器,而系统堆栈定位于内部的SRAM,以便中断的快速响应.双堆栈指针增强了自身的修复能力.当一个任务不正常时,系统本身的堆栈仍然是好的.

5.12 寄存器组

XA支持4组8字节/4字的寄存器,外加12共享寄存器.寄存器组可以在文本切换时用于存储和恢复寄存器.

5.1.3 中断响应和管理

在多任务环境中,对中断响应的要求是很高的.在实时多任务环境中,一个快速的中断对于任务切换是很重要的.XA通过改进中断响应时间能提供一个快速的任务转换环境.

中断处理机制要在堆栈中保存PC(根据Page 0的标志PZ,有1-2个字)和PSW(一个字).中断堆栈驻留在内部数据空间,调用中断包括保存3个字要占用23个时钟周期.予获取服务程序要占用3个附加的时钟周期.当中断或异常/陷阱发生时,当前运行的指令要优先于中断服务.也就是说,在处理完当前的指令以前,不会处理引起中断的时间,这就增大了有效的中断反应时间.在XA中,最长的非中断指令是有符号的32X16除法,要占用24个时钟周期.

所以最坏情况下的中断反应是[24+23+3]=50个时钟周期(16.0M下为3.125微秒, 20.0M下为2.5微秒, 30.0M下为1.67微秒).保存其余寄存器可以通过切换寄存器组.

通常情况下,有16个寄存器要保存在堆栈中,要占用32个时钟周期.全部的反应时间+中断开始的管理最大为68个时钟周期(16.0M下为4.25微秒, 20.0M下为3.4微秒, 30.0M下为2.27微秒).它允许非常快的多任务环境的文本切换.

5.1.4 保护

这里的论述简单的说明了XA支持什么和不支持什么.双数据指针和使用低优先级的管理模式并不意味着全面的保护.假定运行在微处理器中的代码不用考虑来自低优先级任务的对系统的有意侵入.下面给出XA结构的保护特性表.注意标注"不允许"的特性表明:如果在用户模式下企图使用将是不会实现的.但当这些企图发生时,不会有异常和标志发生.

XA中的保护特性

表5.1 段和堆栈寄存器的保护

模式	对DS写	通过DS写	对ES写	通过ES写	通过DS读	通过ES读	通过SSP读	对SSP写	对SSEL.7写
系统	允许	允许	允许	允许	允许	允许	允许	允许	允许
用户	不允许	允许	允许	可选择	允许	允许	不可能	不可能	不允许

注:SSEL的最高位(bit 7)选择是否允许在用户模式下通过ES写.只有在系统模式下才允许操作.

表5.2:在保护模式下的PSW位

模式	对SM位写	对RS0:1位写	对TM位写	对IM0:3位写
系统模式	允许	允许	允许	允许
用户模式	不允许	允许	不允许	不允许

另外,系统堆栈还受到保护,防止在用户模式下执行RETI指令.如果在用户模式下试图使用RETI指令,将引起一个异常中断.如果有必要在用户模式下使用TRAP程序,则用户的RETI异常中断处理必须有执行返回用户模式代码的指令.为了实现返回,用户的RETI异常处理必须从堆栈中弹出最顶端的返回地址(2到3个字,根据XA是否在Page 0模式),然后执行RETI指令.

通过数据段的保护

在用户模式下,每个任务通过不同的数据空间(除了预先设定的共享)同其它任务隔离起来.如果两个任务的地址空间无共享数据,一个任务不会影响到另一个任务的数据,但是它可以读全地址空间的数据.代码的共享是安全的,因为代码是不可能被修改的.一个处于应用模式下的程序不允许写段寄存器,这样就限制了每一个出错的任务的可写范围在它指定的区域内.大多数的应用程序,如果不需要使用多任务系统或外部存储器,不需要任何保护,复位后将留在系统模式,可以操作任何的系统资源.

在一个给定的时间里,一个执行中的XA程序可以对两个段进行操作.它们是数据段DS,堆栈和本地变量驻留在里面;附加段ES,可以用来读取远距离数据结构.通过对任务模块寻址的限制,可以在多任务环境中对系统资源进行有效可靠的控制.

通过双数据指针的保护

XA提供了两级用户/管理保护机制.它们是用户或应用模式和系统或管理模式.在一个多任务环境中,处于管理级别的任务将受到保护,不会受到处于应用级别的任务的干扰.

XA有两个堆栈指针,分别叫做系统指针(SSP)和用户指针(USP).在多任务环境中,一个堆栈指针用于管理系统而另一个指针用于当前活动的任务.这样的保护机制能提供系统软件和用户应用程序的隔离.两个指针也有助于改善中断的性能.如果一个指定应用程序的堆栈超出了片内RAM所能提供的空间,或片内RAM用做其它要求严格的用途(由于片内的RAM操作起来要比片外的快),那么主程序可以放到片外而中断堆栈(使用系统SP)可以放在片内.

所以,XA的这些特性比其它类似的产品更适合于多任务系统.