XA 用户手册

1 XA系列-高性能, 80C51兼容的增强结构, 16-Bit CMOS 微控制器

1.1介绍

微控制器在电子世界中扮演日益重要的角色.过去很多系统必须依赖与机械或简单的模拟电子控制,但是通过嵌入微控制器戏剧性地改进了功能和可靠性,同时也减少了体积和成本.微控制器也能满足通用的需求,这样在多次设计时能减少软硬件的重复设计和成本.

现在系统对微控制器的要求比前几年要苛刻的多.不管我们叫它什么,微控制器?嵌入控制器?或单片机?使用这些器件的系统正要求越来越高的性能和片内集成度.

随着微控制器开始进入更加复杂的控制环境,大吞吐量,高地址处理,片内高集成度的需求导致16位微控制器的开发,以便处理比8位机更多的信息.但是,简单的集成更多的位数和外围功能并不能满足控制系统的发展要求.新的微控制器必须提供高级语言支持,强大的调试环境,实时控制的有效方法,满足系统要求苛刻的功能和成本.

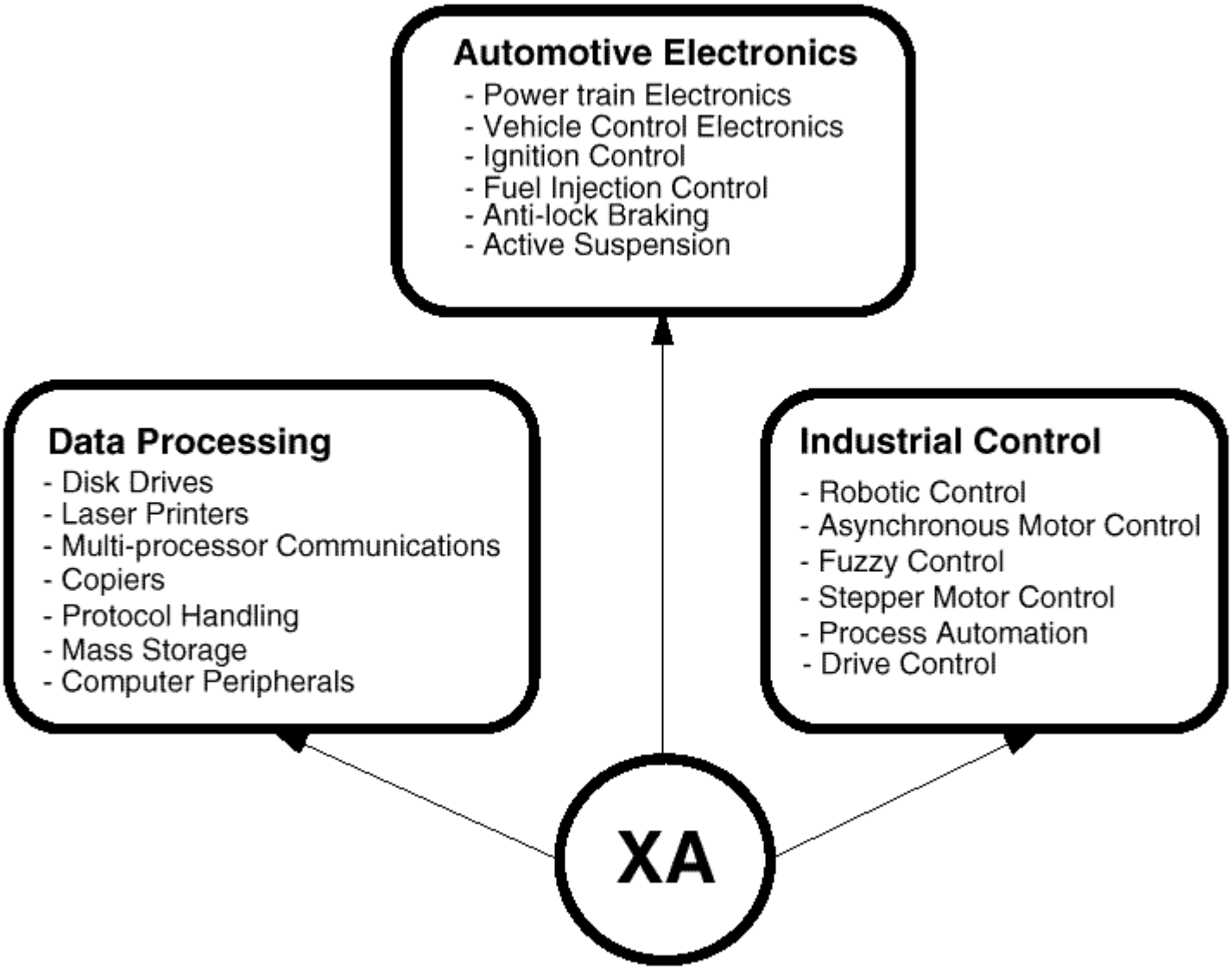

图1 Philips的XA微处理器的应用

Philips的XA扩展系列能满足以上的要求,该系列在各种高性能嵌入式系统控制中能提供最高的性能价格比,包括实时多任务环境.XA系列对片内存储器,I/O口和外设增加了功能,以满足不同领域的需求.内核结构很容易扩展成系列以便满足不同范围的用户需求.有效的指令集支持更快的计算,数据传输,多任务,改善外部事件的响应和有效的高级语言编程.

XA同80C51在代码上保持向上兼容,能为系统的升级提供平滑的设计过渡,而达到性能的增强.

1.2 XA的结构特征

■同标准的8XC51向上兼容(源汇编级)
■24-bit的地址范围(16M的代码空间和数据空间)
■16-bit静态CPU
■可以同时使用16-bit字和8-bit字节.
■增强的指令集
■高效率代码,大部分指令为2-4比特.
■快速的16X16乘法和32X16除法指令.
■16比特堆栈指针和通用的指针寄存器
■支持32个矢量中断-31个可屏蔽中断和1个不可屏蔽中断.
■支持16个软件中断和16个硬件陷阱.
■电源节约方式:掉电方式和空闲方式.
■硬件支持多任务.

2 结构概述

2.1 介绍

Philips的XA(扩展结构)使用当今高速微处理器通用的寄存器-寄存器的结构,能提供最好的性能价格比.XA向上兼容80C51,使原来的80C51用户能得到更高的性能和更大的存储空间;作为高效通用的16位控制器,XA同时支持多任务操作系统和高级语言(例如 C), 但又保留80C51的按位运算的特点.

在概述中将介绍XA结构的术语和概念,为手册后面的详细描述做准备.

2.2 存储区域的组织

XA结构包含多个不同的存储区域.结构和指令译码都加以优化以适应基于寄存器的操作;另外,数学和逻辑操作可以直接在数据区域中完成,解决了使用单一累加器的瓶颈问题.

2.2.1 寄存器列

寄存器列(图2.1)允许同时操作8个字数据;这8个字也可以寻址为16字节.底部的4个字寄存器是组方式,或者说有4组寄存器,在同一时间里可以由他们的其中一组占用底部4个字寄存器的位置.这种特性在中断服务中需要文本切换时非常有用,为复杂的算法提供更多的寄存器空间.在一些指令中,如32比特的移位,乘法,除法,邻近的字寄存器可以形成一个双字寄存器.

除了底部没被选择的寄存器组外,全部的寄存器阵列可以按位寻址.可以被位处理指令操作.

现有的XA指令译码允许将来对寄存器阵列进行扩展,附加上8个字寄存器,但这8个附加的寄存器将是字寄存器,而不能作为指针或按字节寻址.

全部的XA寄存器阵列结构是80C51寄存器结构的扩展集合. 详细信息请参考80C51兼容章节.

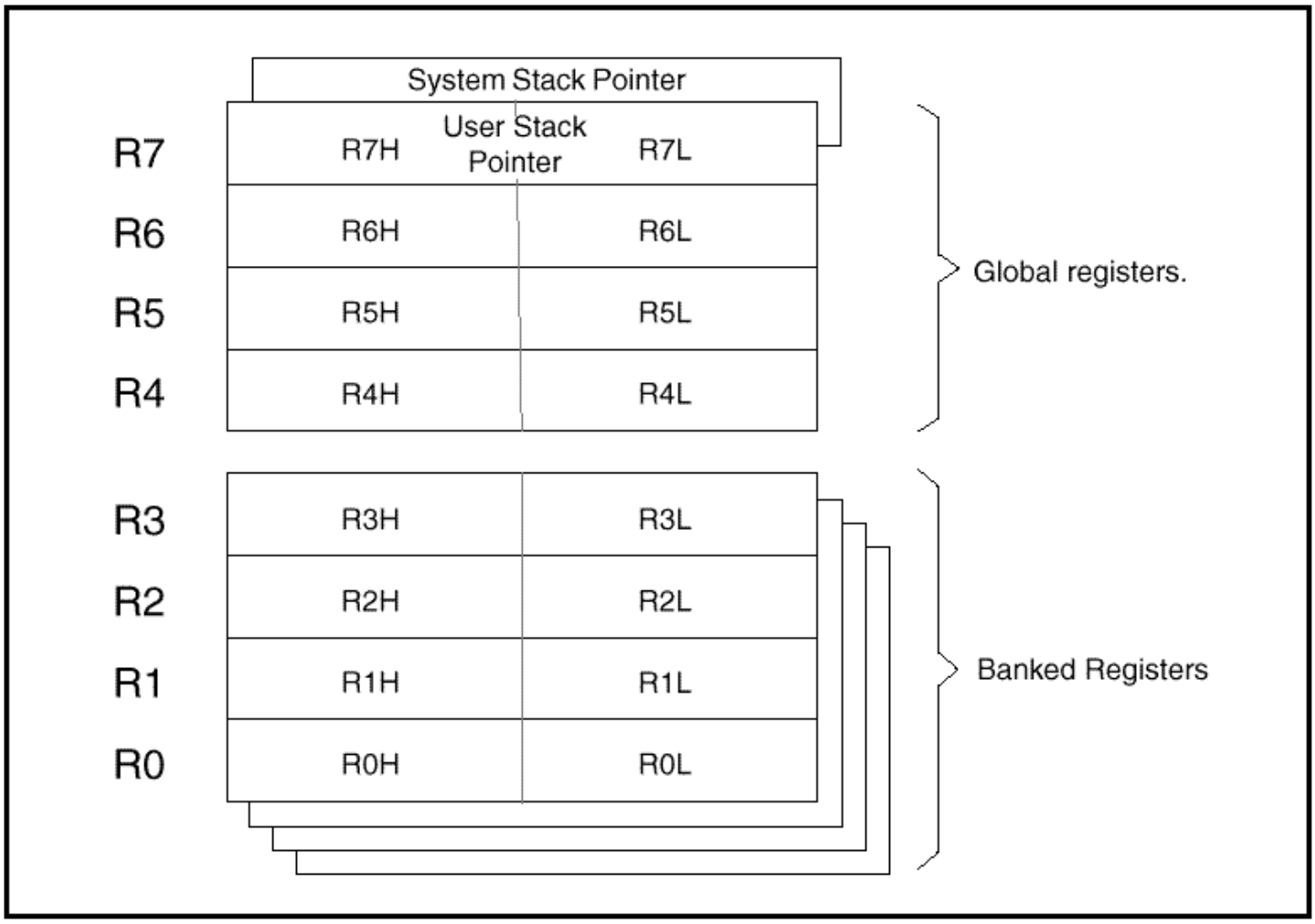

图2.1

2.2.2 数据区域

XA的结构提供24比特的地址线,支持16M比特的数据存储空间.一些派生的器件则有较少的地址线,提供小一些的范围.数据空间由0开始,通常在片内,可以延伸到指定XA系列的片内上限.在这个地址以上,XA自动延伸到外部数据区域.

XA的数据区域被分成多个64K比特的段(图2.2),能为多任务系统提供内在的保护机制和性能改善. 如果需要巨大的存储区域而必须使用全24位的地址线,则可以使用段寄存器提供的高端8位地址线.(图2.3)

XA提供2个段寄存器用来对数据区域进行操作,数据段寄存器(DS),附加段寄存器(ES).每个指针寄存器同两个段寄存器的之一相配合,段寄存器由段选择(SSEL)决定.指针寄存器保持这种配合直到被程序改变.

XA提供了灵活的寻址模式.大部分的算法,逻辑,数据移动指令支持下面的数据区域寻址模式.

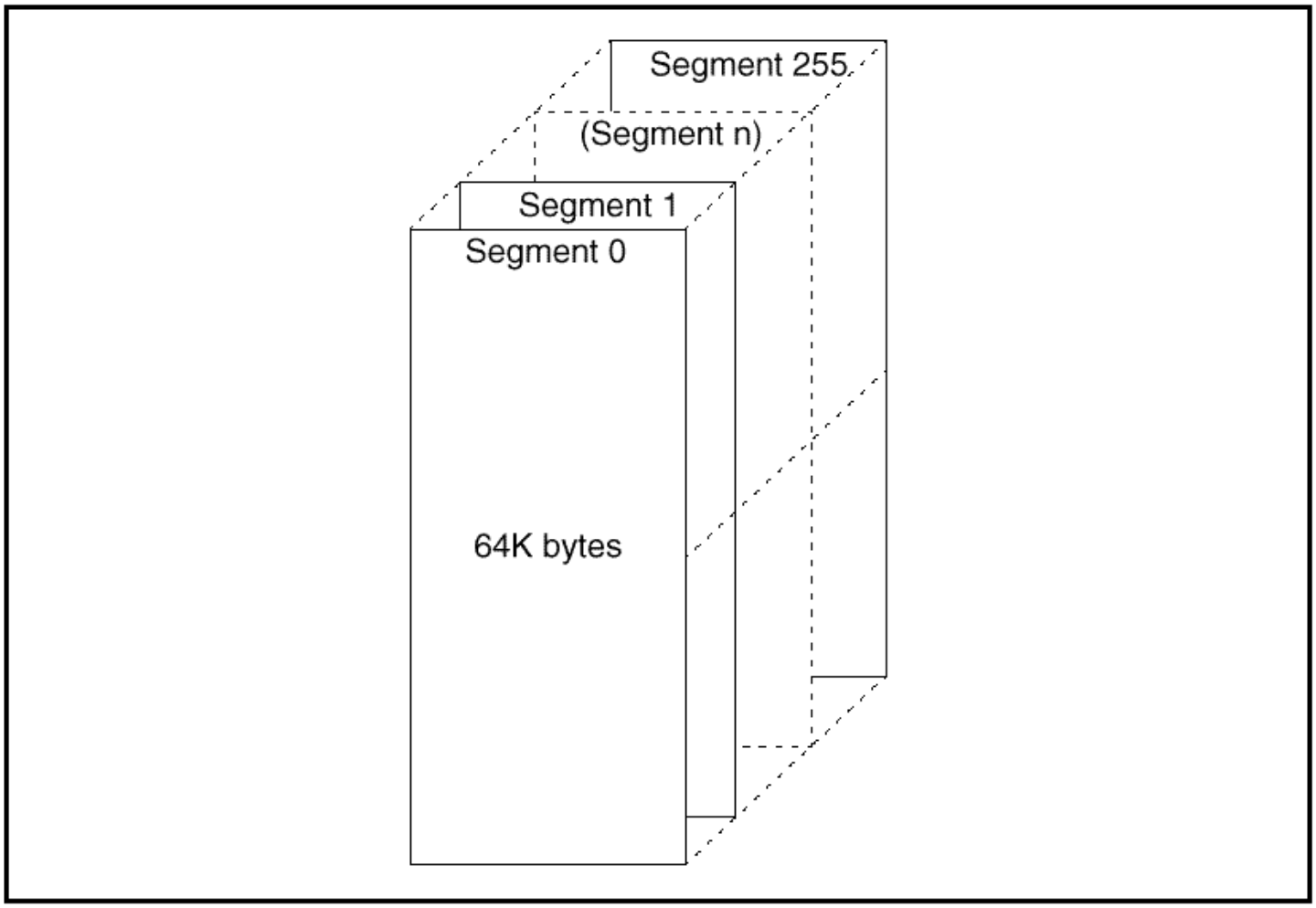
Segment 255
(Segment n)
Segment 1
Segment 0
64K bytes

图2.2

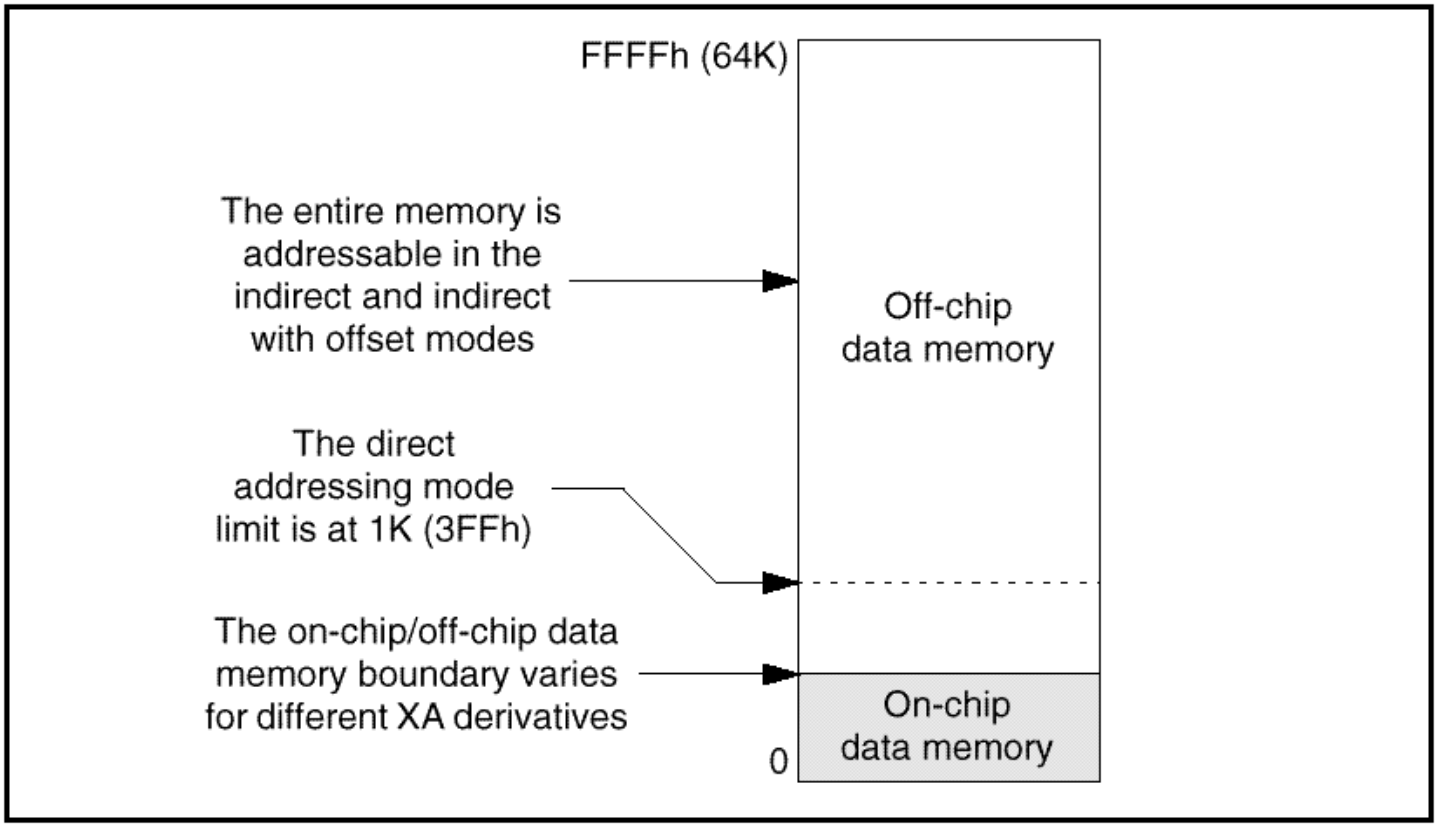
FFFFh (64K)
The entire memory is addressable in the indirect and indirect with offset modes
Off-chip data memory
The direct addressing mode limit is at 1K (3FFh)
The on-chip/off-chip data memory boundary varies for different XA derivatives
On-chip data memory
0

图2.3

直接寻址: 每个段起始的1K比特的数据可以由包含地址的指令直接存取.
间接寻址: 8比特的段寄存器和16Bit的指针寄存器共同连接成完整的24比特数据区域地址.
间接偏移寻址: 指令中包含8比特或16比特的有符号的地址偏移量,它叠加到指针寄存器的内容数值上,然后同段寄存器一起构成完整的24比特地址.如果把指针寄存器的内容作为起始地址,使用这种方法可以读取一个数据组内的数据.它也允许子程序通过堆栈获取参数.
间接增量寻址: 地址的形成同间接寻址类似,但地址寄存器的内容在操作完成后递增.
其它寻址方法: 数据移动指令和其它的一些特殊用途的指令也有一些附加的寻址方式.
XA的数据空间的寻址方式同80C51向上兼容.详细的内容请参考第9章.

2.2.3 代码区域

XA是哈佛结构,数据空间和代码空间是分开的.XA能提供16M连续的,不分段的代码空间(图2.4).在XA系列中,对于具有ROM或EPROM的型号,片内的代码空间总是从0开始,延伸到片内代码空间的顶部.在这之外,将从外部获取代码.多数XA支持数据和代码的外部总线操作.也可以在无ROM的模式下使用,这时不使用片内代码.

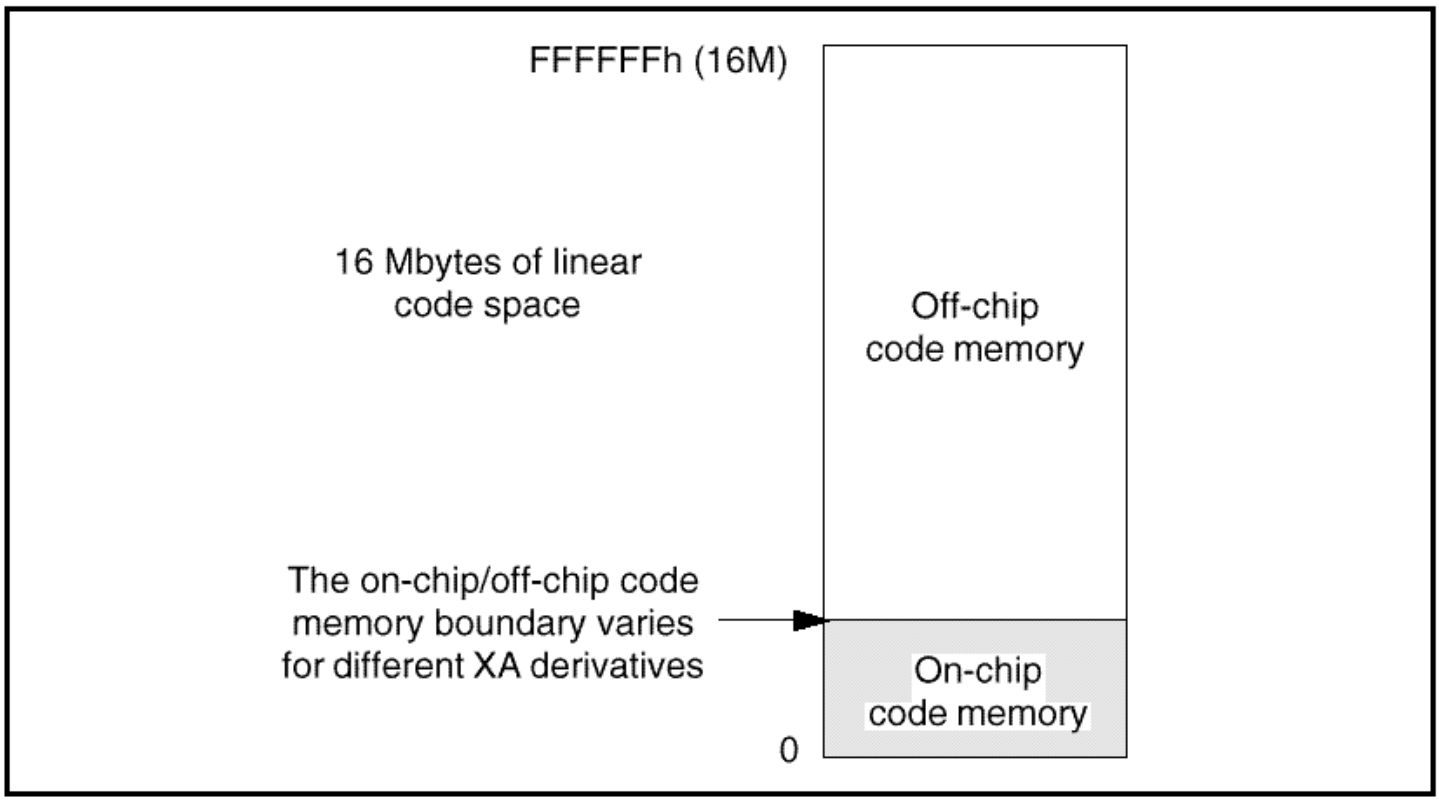

图**2.4**

在某些情况下,代码空间可能放置的是数据.特殊的指令可以通过指针对全部的代码空间进行操作.指定的段寄存器(CS或代码段),或程序计数器的高8位,同指针一起,可以确定代码的地址.

2.2.4特殊功能寄存器SFR

特殊功能寄存器(SFR)提供的功能是:操作寄存器,内部控制寄存器,外围设备,和I/O口.任何SFR可以由程序在任何时间操作,不必顾及任何指针和段.SFR的地址总是全部包含在指令中.见图2.5
全部SFR的空间为1K比特,进一步分成2个512比特的空间区域.低半部分配给内部SFR,高半部分配给片外SFR.这样允许把片外的I/O口映射成XA的SFR.片外的SFR并不是所有的XA型号都支持.
片内的SFR可以根据需要实现对外设的控制,或控制CPU的功能和特性.每种型号的XA可能有不同数量的SFR,因为有不同的外设功能.在某些XA型号中,有一些地址的SFR不可用.
片内的SFR,开始的64字节可以按位寻址.需要位寻址的CPU或外设寄存器可以定位在这个区域.

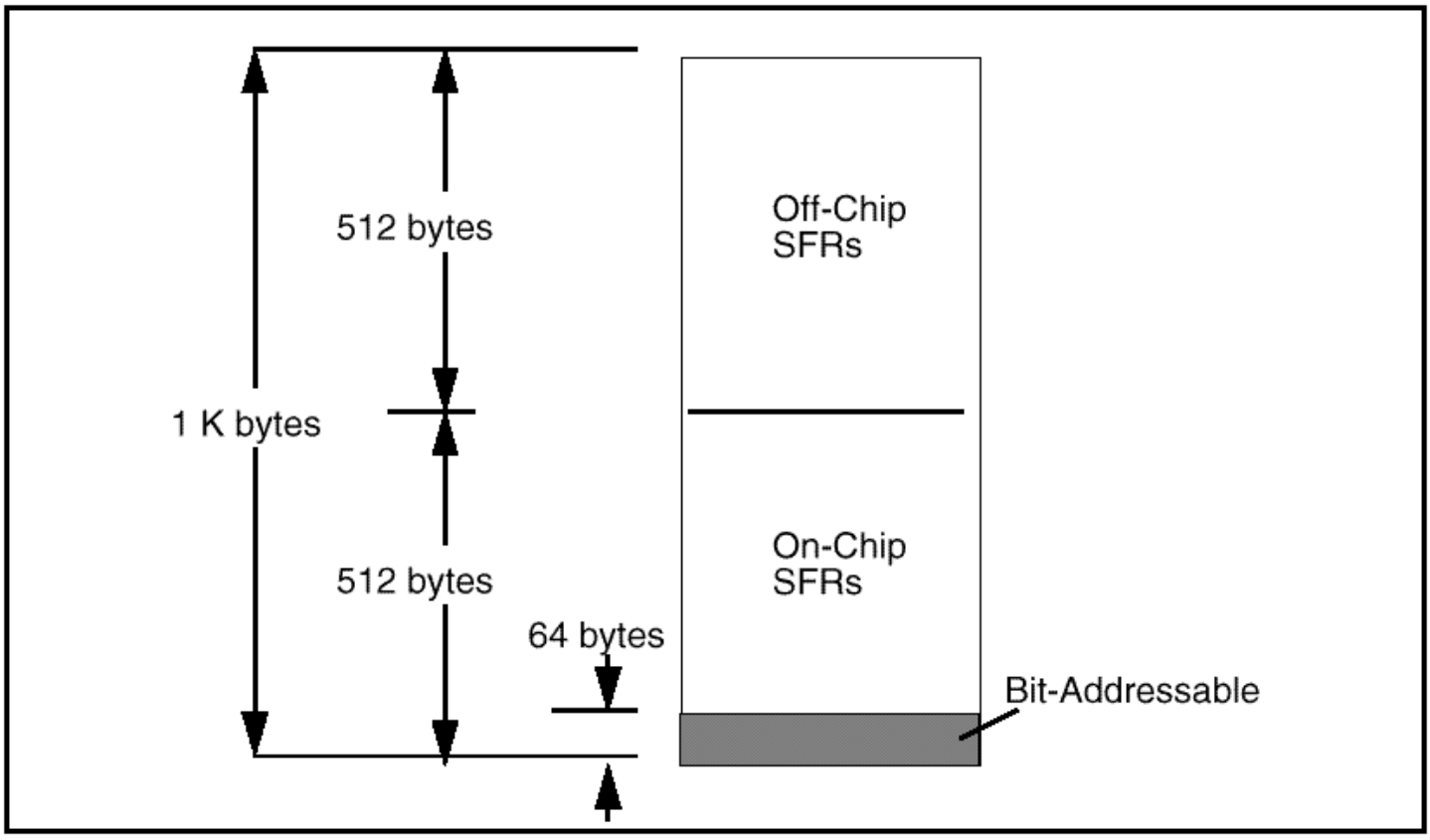

图**2.5** 特殊寄存器地址空间

2.3 CPU

图2.6是XA的全部结构.这一节将对每一部分进行描述.

2.3.1 CPU 功能块

XA处理器包含多个功能块:指令获取和译码;执行单元;算术运算;故障处理控制部分;中断控制;寄存器阵列和核寄存器;程序存储器(ROM,EPROM),数据存储器(RAM);SFR和外部总线接口;震荡器,片上外设和I/O口.

在XA系列中,绝大多数型号所具有的功能块可能在一些型号中没有.这些功能块是:

外部总线接口,特殊功能寄存器总线接口,特殊外设,I/O口,程序和数据存储空间,中断控制器.

CPU 性能特点

XA核部分是流水操作,而一些CPU功能的执行是并行的.例如,指令的读取和译码,一些情况下的数据回写,都是同指令操作同步执行的.这种部分流水操作能在低成本下实现实现很快的指令执行.例如,XA在30MHz的时钟频率下,大多数寄存器-寄存器的指令执行时间为3个CPU周期,100纳秒.

ALU

在XA中数据操作是通过16-bit操作单元完成的,同时提供8比特和16比特的功能.一部分允许32比特的功能,例如移位,乘法,除法.

核寄存器

XA核包含多个关键的特殊功能存储器,由程序进行控制.

系统配置寄存器(SCR)设定XA的基本操作模式.

程序状态字(PSW)包含状态标志(它显示了运算器ALU的操作结果),寄存器组的选择位,中断屏蔽位,和其他的系统标志.

数据段(DS),扩展段(ES),代码段(CS)寄存器寄存器包含当前数据段的段码.

段选择寄存器(SSEL)包含的BITs位决定指针寄存器使用哪个段寄存器的内容.

功率控制寄存器PCON控制处理器的功率减小模式.

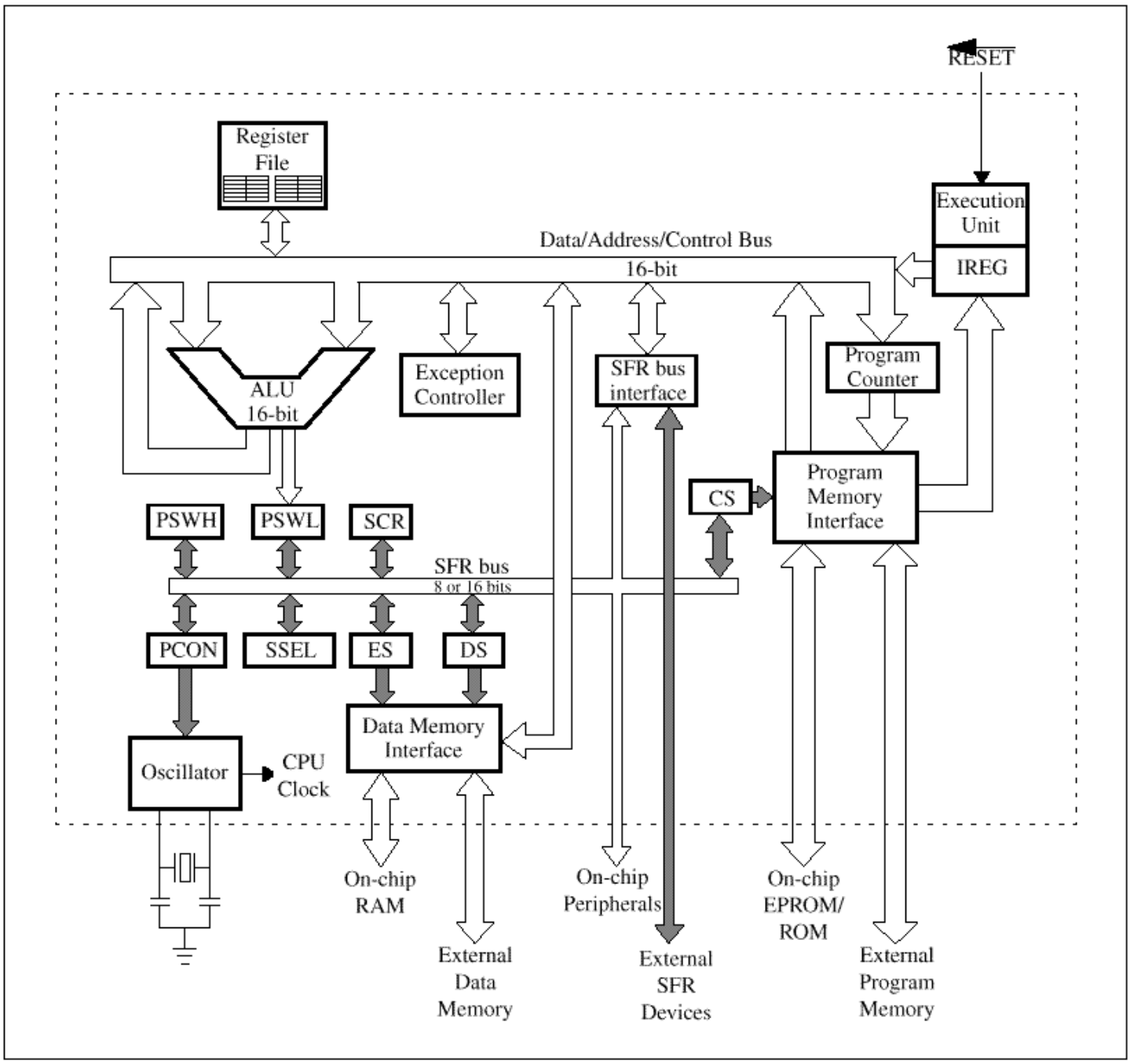

图2.6 XA的结构

执行和控制单元

执行和控制单元读取代码区的指令,译码后执行.XA通常是预先读取现在立即要执行的指令.这些预先读取的指令放在读取和译码单元的7字节队列中.如果队列中包含要执行的指令,执行单元可以不必等待指令的读取,直接执行新的指令.如果程序出现分支,则队列被刷新,从新的位置读取指令,此功能块也决定从片内读取指令,还是从片外读取指令.

队列头部的指令被译码成不同的功能区域,告诉CPU的其它部分在指令执行时如何动作.这些区域存储在阶段寄存器中,保持到下一个指令的执行.

执行单元

执行单元在指令执行时,要控制很多CPU的功能块.它发送地址信息,发送读写命令到寄存器阵列和存储器控制块,通知读取和译码单元何时跳转,控制堆栈,并保证所有的操作按照一定的顺序正确执行.执行单元从ROM中获取每个指令的控制信息.

中断控制器

XA中断控制器可以接受任何中断源的申请,并根据可编程中断优先寄存器保存的优先级信息对中断源进行分级.当中断控制器从XA器件的中断源中接受到中断申请后.它根据可编程的优先寄存器对他们进行优先级进行分类.然后再判断PSW的中断屏蔽位.如果中断比当前运行代码的优先级别高,则中断控制器就会响应并执行中断程序.
中断控制器也包含附加的寄存器以处理软件中断,软件中断的优先级别是固定的,按顺序排列,不会出现"优先级倒置"的现象.
虽然中断控制器不是XA核的一部分,但所有型号的XA都通过一定的方式加以支持.

异常处理控制器
异常处理同中断类似,只不过它处理的是CPU的异常事件,而不是硬件和软件的中断申请.异常中断源是:堆栈溢出,除以0,用户执行RETI指令,硬件断点,跟踪模式,无屏蔽中断(NMI).

异常处理的执行按照固定的优先级队列.通常异常处理必须立即执行,因为它表明了重要事情的发生,或者是问题必须处理完后系统才能恢复正常的操作.
XA核总是包含异常处理控制器.

中断和异常处理
中断和异常处理均使用矢量表,它驻留在代码区的地址低端.每个中断和异常处理在矢量表中有一个入口,它包含服务程序的起始地址和服务程序开始使用的新的PSW值.服务程序的起始地址必须在第一个64K代码区域内.

当XA执行一个异常处理或中断时,它首先在堆栈中保存返回地址,接着是PSW的内容.下一步是从矢量表中调入相应的程序开始地址,以及要使用的PSW.

当中断程序结束后,通过执行RETI指令返回.这个指令从堆栈中调入原来的PSW值,PC计数器,返回原来的中断点.如果服务程序使用了除PSW和PC以外别的资源,应该存储和恢复占用的寄存器和其它机器状态,一般的是使用堆栈,也经常使用寄存器组切换.

复位
通过对复位脚施加有效的低电平后,就能产生电源复位和其它的外部复位.通常,使用一个简单的电阻,电容电路就能在电源启动时产生一个复位脉冲.复位端是一个施密特触发输入,避免产生复位端的不可靠复位.
XA可以在程序中使用RESET指令来复位.指令复位同外部的复位有相同的效果,但有一些硬件锁存信息,例如EA脚,总线宽度等不再调入.这些差别是必要的,因为驱动这些引脚的外部电路不可能知道在处理中产生了复位.

一些型号的XA可能含有硬件看门狗.如果超出了允许时间,也能产生相同的复位.

震荡器和功率节约模式
XA系列含有一个片上的震荡器,使用晶体或陶瓷,为处理器提供时钟源.
XA支持两种功率节省模式:空闲模式,关闭模式.通过设置功率控制器PCON的相应位,就可以激活.空闲模式关闭所有的处理器功能.但片上的外设和外部中断仍然开启,震荡器也工作.一个中断可以使XA从停止的地方继续运行.关闭模式则全部关闭,包括震荡器.这将把功率消耗减小到CMOS电路泄露电流的水平,当然还包括引脚产生的负载电流.从关闭模式返回到正常操作需要震荡器的启动,大约需要10ms的时间.以下的动作可以退出关闭模式:复位,外部中断.在关闭模式中,片上的RAM的数据仍然保存.如果想进一步的在关闭模式中节省消耗,可以降低Vdd,详情参考数据手册.

堆栈

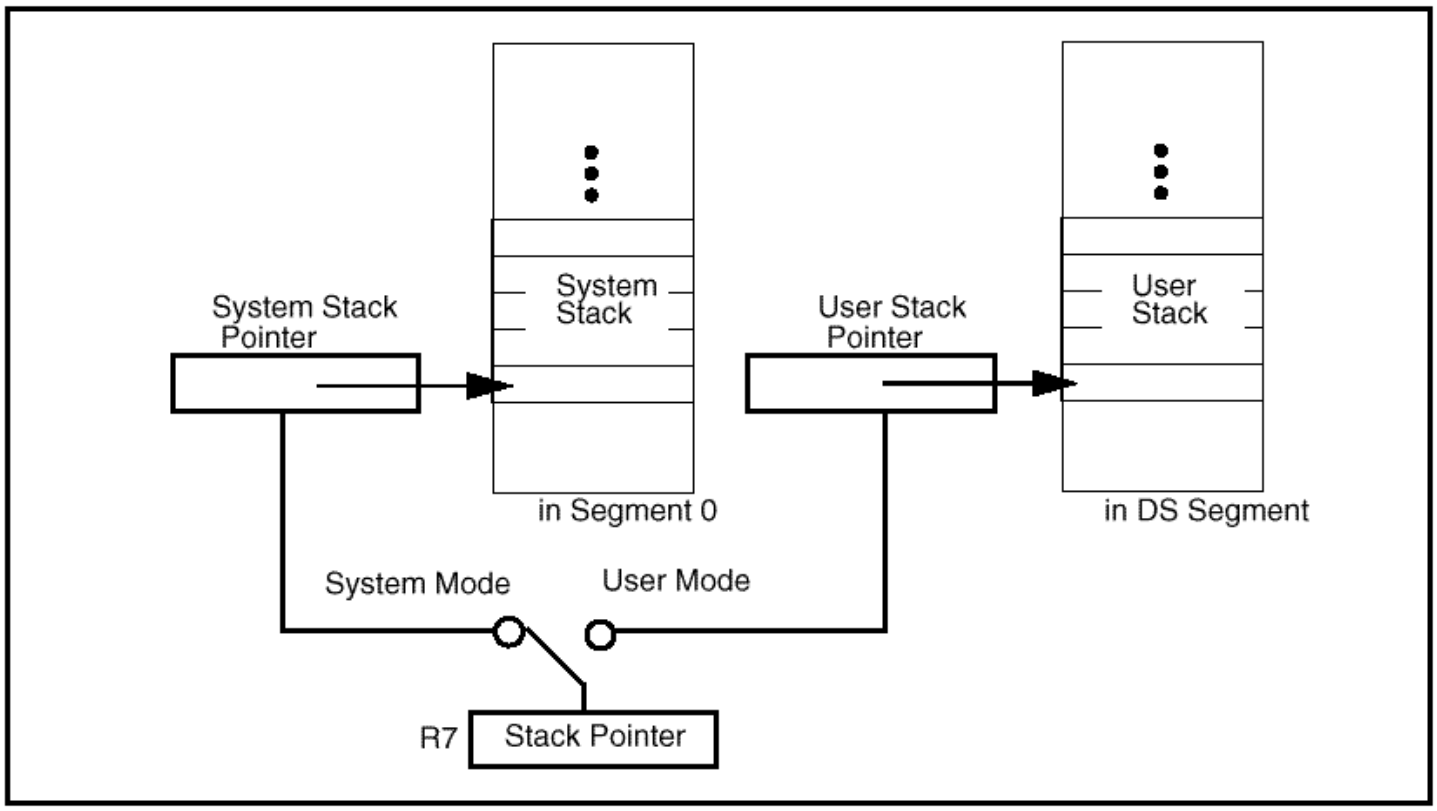

图2.7 XA的堆栈

处理器的堆栈用来保存中断和子程序的返回地址,是一个临时的数据区域.XA有两个堆栈指针,系统堆栈指针SSP,和用户堆栈指针USP,每一个对应不同的堆栈:系统堆栈和用户堆栈.见图2.7.系统堆栈总是驻留在第一段的数据区内.用户堆栈的的区域由当前的DS寄存器决定.
在同一时间内,执行代码只能通过R7对一个区域进行操作.由于每个堆栈驻留在一个数据段内,所以它的最大容量为64K比特.为何使用两个堆栈指针,将在下面的任务管理部分详细讨论.
XA的堆栈向下生长,在数据区内由高端到低端.当前堆栈指针总是指向最后压入堆栈的项,除非堆栈是空的.执行压入堆栈后,堆栈指针减2.当执行弹出堆栈时, 操作正好相反.首先,数据读出,堆栈指针减2.在堆栈中的数据总是偶数个字节,即按字在区域中排列.

调试特性
XA具有一些特殊特性,用于编程和系统调试.软件断点指令可以由调试器插入用户的程序中,程序执行到这里将产生断点,转向断点服务程序,在服务程序中传递CPU参数,用户就可以看到CPU的状态.
跟踪模式同断点类似,但它是在执行完每一步指令后,由硬件实现的.

跟踪服务程序可以跟踪用户的每一条指令,把CPU的状态通过串行口或其它外设传输出去,以便显示和存储.跟踪模式由PSW的一位决定.不管是中断或异常中断发生,XA都可以改变跟踪模式位.这给跟踪模式的使用带来了很大的灵活性,举个例子来说,你可以让中断全速运行以满足系统硬件的需要,而同时单步运行主程序.
使用这两个特性可以作成一个简单的监视调试程序,既可以允许用户单步运行一个程序,又可以全速运行,当达到断点时停止,两种情况下都可以在CPU重新运行前观察CPU的状态.

2.4任务管理

XA具备多种特性用于支持多任务.多任务可以理解成在同一处理器上同时运行多个程序,并用一个管理程序决定该执行那一个程序或任务,执行多长时间.由于多个任务分享CPU的资源,因此每一个任务需求的资源必须隔离,而且由一个任务切换到另一个任务时CPU的状态也必须保存.对于微控制器来讲,问题相对要比微处理器简单,因为微处理器的执行代码总是来自相同的地方:所运行系统的设计者.因此,这个代码基本上可以认为是可信赖的,而施加过分的措施以防止系统的异常是不必要的.在XA设计中重要的是要防止简单的冲突.在XA的多任务设计中,第一步是提供两个可执行的文本:一个是基本的任务,在XA中称为用户

模式;另一个称为管理程序(系统模式).运行在系统模式下的程序可以对所有的处理器资源进行操作,也可以设置和启动程序.

运行在系统模式下的代码使用系统堆栈指针(SSP),而运行在用户模式下的代码使用用户堆栈指针(USP).系统指针总是位于第一个数据段内,这样它可以使用快速的片内RAM.用户堆栈位于每个任务所在的数据段内,由DS存储器决定.用户模式使用独立的用户堆栈指针,而不是使用系统的系统指针,这样可以防止任务在意外情况下破坏系统堆栈内的数据和进入别的任务空间.另外的保护机制是控制位和寄存器只能由系统代码改写.例如DS寄存器,它确定用户模式的当前数据段,只能在系统模式下改写.虽然任务仍然可以读写其它段寄存器ES寄存器,但是它不能通过ES改写存储器区域,除非系统允许.这样数据分段的方式就可以防止任务通过非法途径操作数据区域.

其它保护特性包括启动跟踪模式和变换中断屏蔽.
四个寄存器组对于小的多任务非常有用,每一组用于不同的任务,其中一个用于系统代码.这样在任务切换时可以减少CPU状态存储的时间.

2.5指令集合

XA的指令集合用来支持通用的控制用途.指令译码经过优化以适应大多数常用的指令:寄存器-寄存器或寄存器-直接地址算法和逻辑操作;短的条件转移和无条件转移.这些指令均译码成2字节.大多数XA指令是2字节或3字节,但也有一些1字节的指令,和4,5,6字节的指令.

为了加速处理,通常指令的执行与指令的读取以及有时候的读-返回是重叠进行的.

2.5.1指令语法

XA的指令语法同8051的语法在很多地方类似.一个典型的XA指令有一个基本的助记符,如"ADD",后面是要操作的运算对象.基本的语法见图2.8.操作流程的方向由运算对象出现的顺序来决定.例如指令:"ADD R1,R2",执行R1加R2后,结果防在R1中.由于在XA中R1和R2是字寄存器,因此这是16位运算.

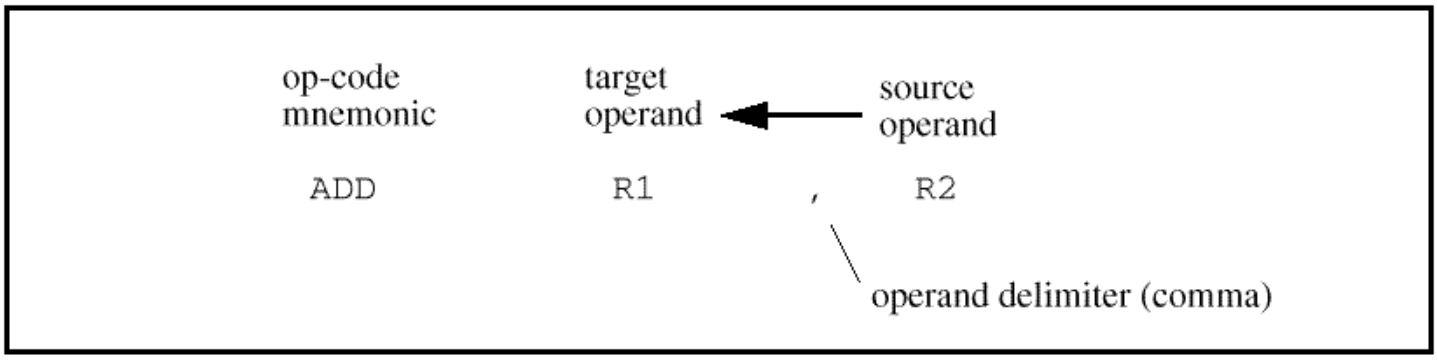

图**2.8**基本的指令语法

间接的引用(用寄存器的内容作为地址引用数据),要把运算对象用方括号包起来,如"ADD R1,[R2]".见图2.9.这个指令的结果是,把R1同数据区的一个单元想加,这个单元的地址由R2的内容决定,当然还要同段寄存器结合,结果放在R1中.变换运算单元的顺序("ADD [R2], R1"),运算结果将防在数据单元中,见图2.10.

大多数的指令支持一个附加的特性,称作为"自动增量",在数据单元操作完成后,作为间接数据地址的存储器数值将自动增加.例如下面的一行源程序,"ADD R1,[R2+]".如图2.11所示,自动增加的数值总是匹配指令中使用的数据尺寸.上面的是字操作,所以R2的数值将增加2.

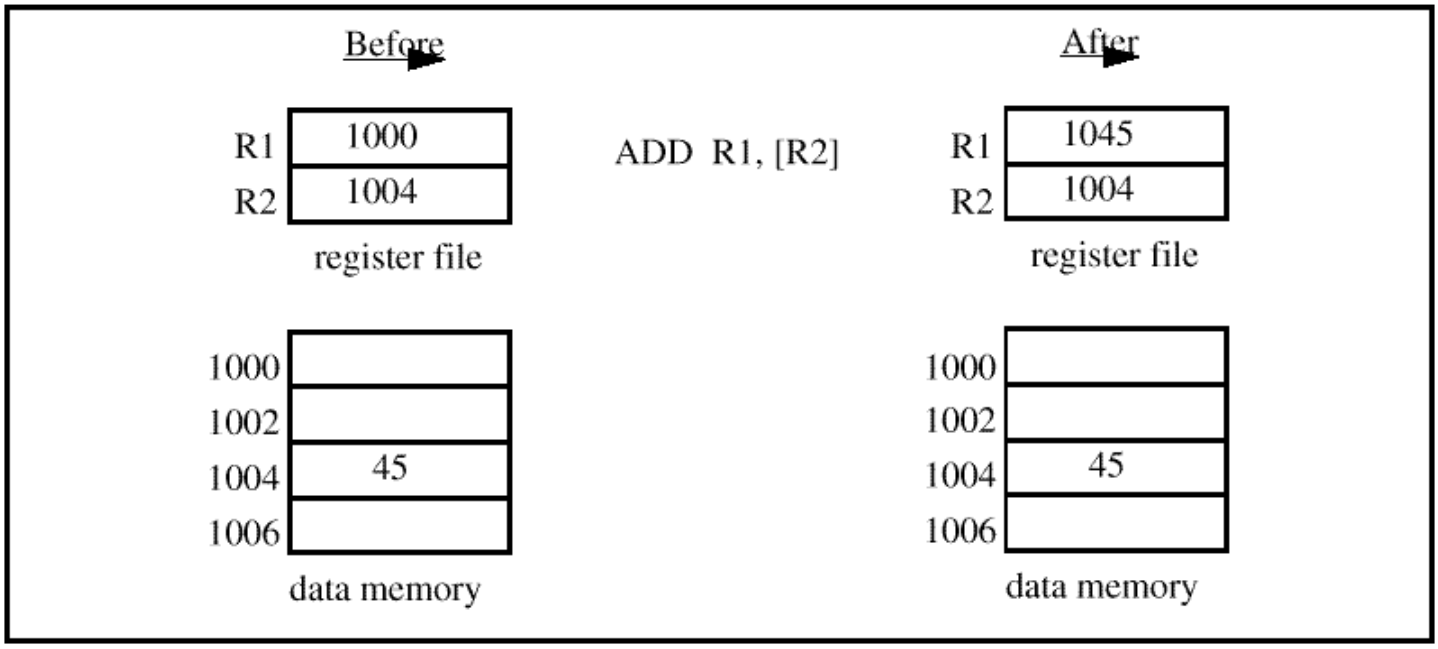

图**2.9**基本的间接寻址语法(到寄存器)

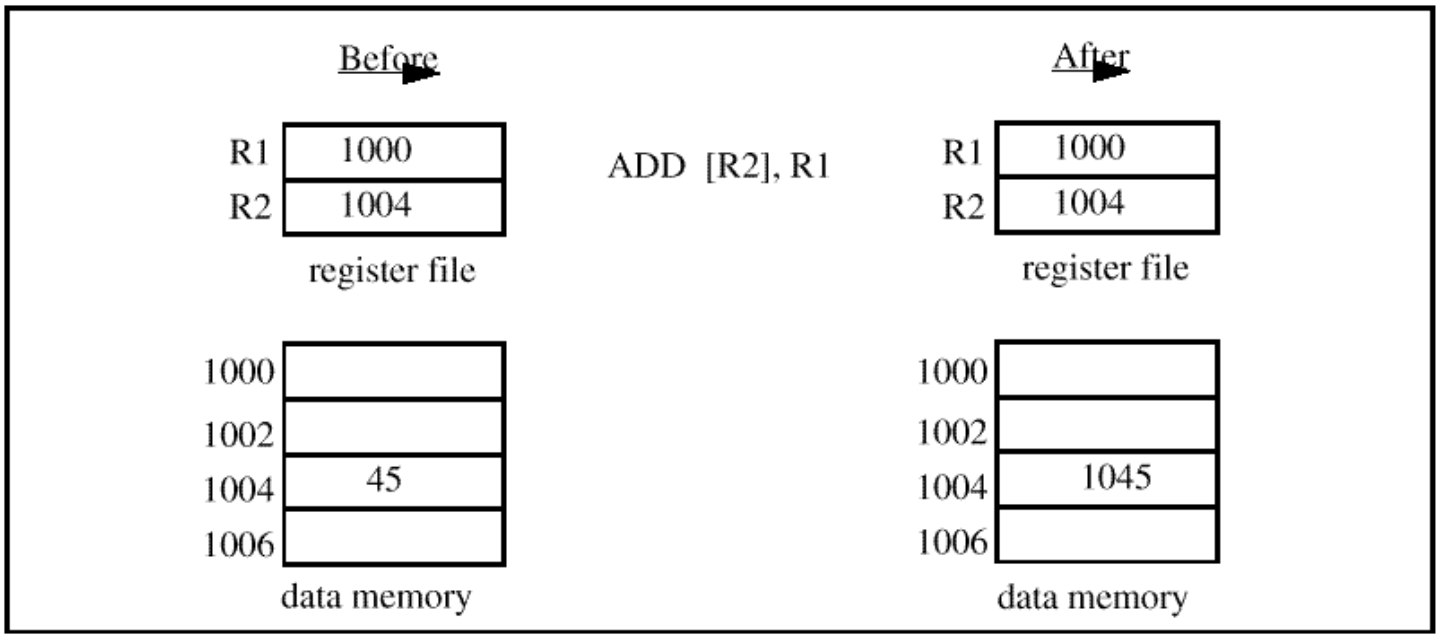

图**2.10**基本的间接寻址语法(从寄存器)

另一种间接寻址方式是间接偏移.在这种方式中,指令中的一个立即数被加到间接寄存器的内容中形成一个真正的地址.结果是16比特的数,再加上段寄存器形成完整的地址.如果偏移计算溢出16比特,溢出将被忽略,这样间接寻址的引用将维持在同一段内.指令中的立即数是一个有符号的8位或16位的偏移量.这样对于8比特将产生+127到-128的偏移;对于16比特将产生+32767到-32768范围内的偏移.注意:由于地址计算被限制在16比特范围内,所以16比特的偏移允许操作整个数据段.

当一个指令需要一个立即数(包含在指令中的数值),将使用"#"符号.例如:"ADD R1, #12"说明把R1的数值加上12.

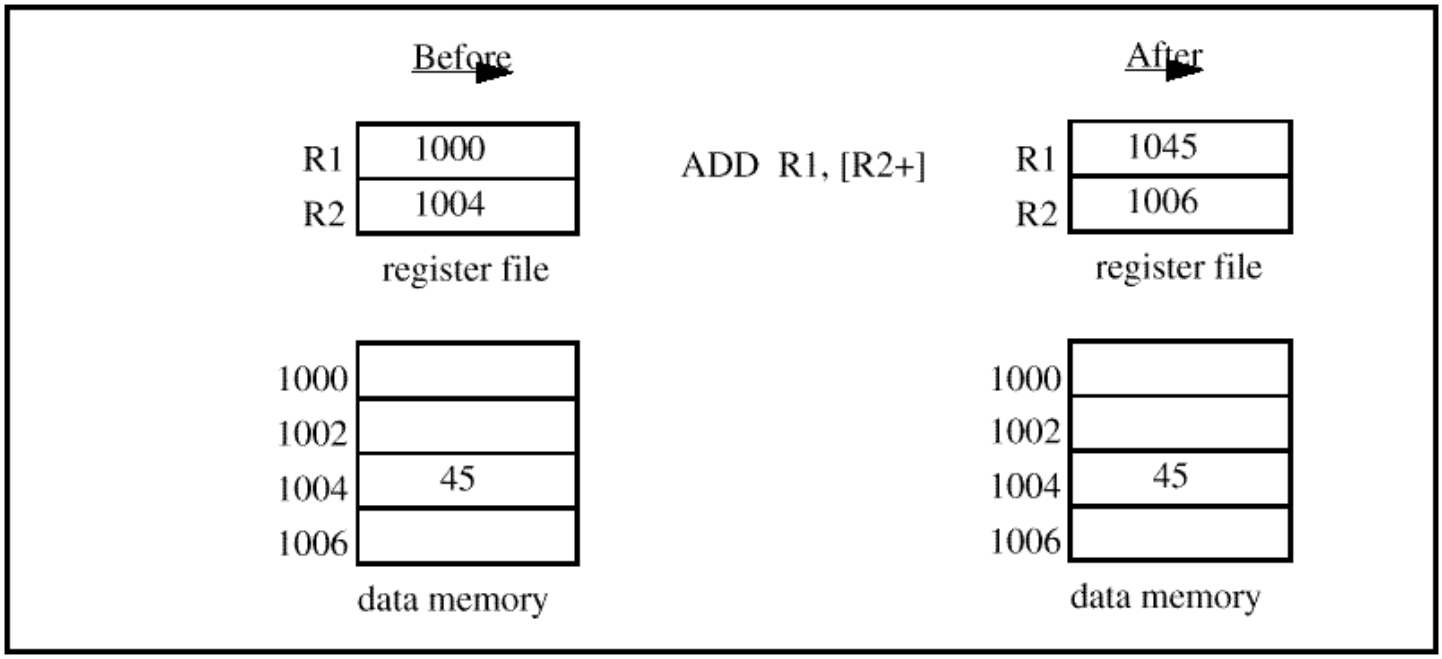

图**2.11**间接寻址的自动增量

由于间接数据引用和立即数不会表明操作对象的尺寸,所以一些XA指令必须表明运算尺寸.例如以下指令,"MOV [R1], #1".立即数不会表明运算尺寸,储存到R1所指向的地址单元可能是字,或字节.为了表明类似指令的意图,尺寸的标识必须加到助记符中,如下"MOV.b [R1], #1".它告诉我们是字节操作.如果是"MOV.w [R1], #1"则是字操作.

如果指令中使用了直接地址,可以直接写成:"ADD 123, R1",表示R1和数据区地址123的内容相加,和保存在地址123中.在实际的程序中,直接地址可以给出一个名称,使程序可读性强,例如:"ADD Count, R1"操作SFR类似与直接地址,但通常是使用它们的名字:"MOV PSW,#12".如果使用它们的实际地址而不是他们的名字将使程序难以读懂而且在XA系列中难以通用.

在指令中指定bit位的地址可以用几种方法.bit位可以给出一个唯一的名称,或者是它在存储器或单元中的位置。例如，一个bit位是PSW中的一位，例如进位标志”C”.如果想清除进位标志,可以使用以下指令”CLR C”,也可以由它在PSW中的位置来寻址:”CLR PSW.7,”,这里句点”.”这是一个位引用.在程序也可以使用位的名称来标识位.

最后,程序中的地址可以使用名称,也可以使用数字.但是如果程序中使用名称,将容易阅读和验证.例如,"JMP Loop" 和"JMP 124".

2.5.2 指令集概述

下面的章节给出XA系列的指令概述.详情请见第6章.

基本的算术,逻辑和数据移动指令

程序中最常用的指令是算术,逻辑指令,加上移动指令.XA支持如下基本的操作.

ADD 简单加
ADDC 带进位加.
SUB 减.
SUBB 带借位减
CMP 比较
AND 逻辑与.
OR 逻辑或
XOR 逻辑与或.

这些指令支持下面标准的XA寻址组合:

运算对象	解释
R, R	源和目标都是寄存器.
R, [R]	源是间接地址,目标是寄存器.
[R], R	源是寄存器,目标是间接地址.
R, [R+]	源是间接地址并自动增加,目标是寄存器.
[R+], R	源是寄存器,目标是间接地址并自动增加.
R, [R+offset]	源是间接地址并带有8-16bit的字节偏移,目标是寄存器.
[R+offset], R	源是寄存器,目标是间接地址并带有8-16bit的字节偏移.
direct, R	源是寄存器,目标是直接地址.
R, direct	源是直接地址,目标是寄存器.
R, #data	源是8或16bit的立即数,目标是寄存器.
[R], #data	源是8或16bit的立即数,目标是间接地址.
[R+], #data	源是8或16bit的立即数,目标是间接地址并自动增加.
[R+offset], #data	源是8或16bit的立即数,目标是一个间接地址并带8或16bit的偏移量.
direct, #data	源是8或16bit的立即数,目标是直接地址.

XA另外一些指令使用了不同的运算对象组合.全部的指令细节见指令集章节.以下只是一个概述.

附加运算指令

ADDS	加带符号立即数的低4位.
NEG	取反.
SEXT	根据以前运算标志位N的结果,决定现在运算对象是00,或者FFH.
MUL	乘.
DIV	除.
DA	十进制调整..
ASL	左移位.
ASR	右移位
LEA	调入实际的地址.

附加的逻辑指令

CPL	逻辑取反..
LSR	逻辑右移.
NORM	移位标准化.
RL	循环左移.
RLC	带进位循环左移.
RR	循环右移.
RRC	带进位循环右移

其它数据移动指令

MOVS	有符号低4位数值移动.
MOVC	移至或移自代码空间.
MOVX	移至或移自外部数据空间.
PUSH	数据压入堆栈.
POP	数据弹出堆栈.
XCH	交换数据.

位处理指令

SETB	指定位置1.
CLR	指定位清零.
MOV	移到或移自进指定位.
ANL	指定位同进位C逻辑与.

ORL	指定位同进位C逻辑或.

跳转,分支,调用指令.

BR	分支到代码地址(加或减256个字节).
JMP	跳转到代码地址(地址由指定的JMP变量决定).
CALL	调用子程序(范围由调用变量决定).
RET	由子程序或中断返回.
BCC	条件分支,由15个可能的条件变量决定.
JB, JNB	由bit位的状态决定是否跳转.
CJNE	比较两个操作数,如果不等则跳转.
DJNZ	减1,如果不等于0则跳转.
JZ, JNZ	为0或不为0则跳转(同8051兼容).

其它指令

NOP	空操作指令.
BKPT	断点(用于调试).
TRAP	软件陷阱(在多任务系统中用于调用系统程序).
RESET	复位整个芯片.

2.6 外部总线

多数XA型号能够通过外部总线操作外部数据和代码.外部总线为外部器件提供地址信息,并开启代码读,数据读,和数据写的信号.标准的XA外部总线提供了灵活,简单的连接,对外部指令的读取进行了优化.如4.4.4节所述,外部总线宽度是可以硬件设置的,在复位过程中,XA检查P15的电平,决定总线宽度是8位还是16位.

2.6.1 外部总线信号

XA的外部总线支持8位或16的数据传输,多达24条地址线.地址线的具体数目在不同的型号上有所不同.下面是外部总线的控制信号和它们的功能.

信号名称	信号功能
ALE	地址锁存使能信号.它引导锁存部分外部地址,以备下一个总线的操作.锁存的地址可以是数据地址,也可以是代码地址..
PSEN	程序空间使能信号.表明XA通过外部总线进行读代码操作.一般是连接到外部ROM的输出使能端.
RD	读信号.外部数据读控制.一般连接到外设器件的RD端.
WRL	写信号.外部数据的低字节写控制.一般连接到外设的WR端.对于8bit的总线,它是唯一的写控制信号.对于16bit总线,此信号只加到低字节的数据上.
WRH	高字节写信号.在使用16bit总线时,它是外部总线高字节的写信号控制.
WAIT	等待信号.允许加大任何类型的外部总线周期.如果使用WAIT信号,在总线操作时操作将根据该信号相应的电平进行等待.

2.6.2 总线配置

用户可以把标准的XA总线配置成多种方式.首先,总线宽度可以配置成8或16bit.这可以通过设置一个引脚在复位时的逻辑电平来实现.如果初始代码是从片内执行,而不是先进行外部总线操作,还可以通过软件改变总线宽度.如同80C51,由EA引脚确定初始代码是从片内或片外读取.

另外,可以配置总线的位数,这样就能更加合理的使用I/O脚.由于通常外部总线分享I/O脚和I/O脚功能.因此在一些设计中,我们可以按照需要设置需要的地址线的数目,省下的I/O脚可以另作它用.

2.6.3 总线时序

标准的XA能提供高级的总线时序结构.ALE的宽度,PSEN的宽度,RD和WRL/WRH的宽度,对WRL/WRH的数据保持时间的宽度,这些参数均可以单独控制.即便是在一个很宽的晶振频率范围内,通过在一定范围内改变上述时间参数,可以适应大多数的RAM,ROM,EPROM,而不必使用附加的锁存器,缓冲器,等待产生器.下图显示了基本的XA总线操作时序.详情请见第7节和器件数据手册.

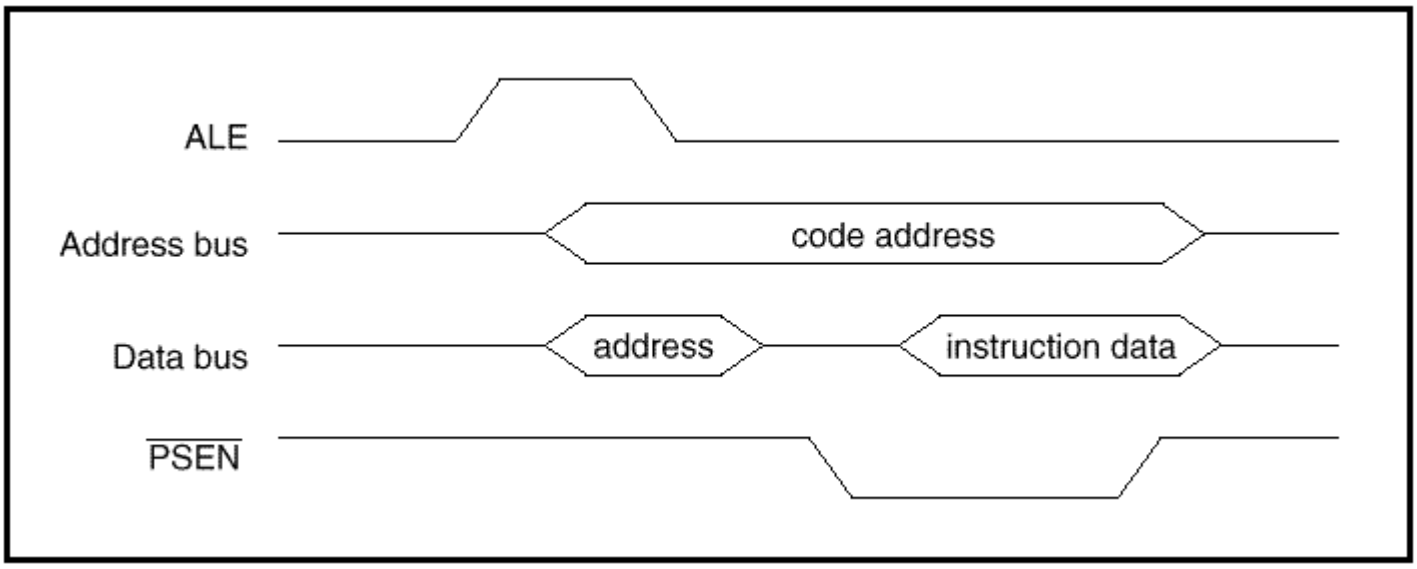

图**2.12** 典型的外部代码读.

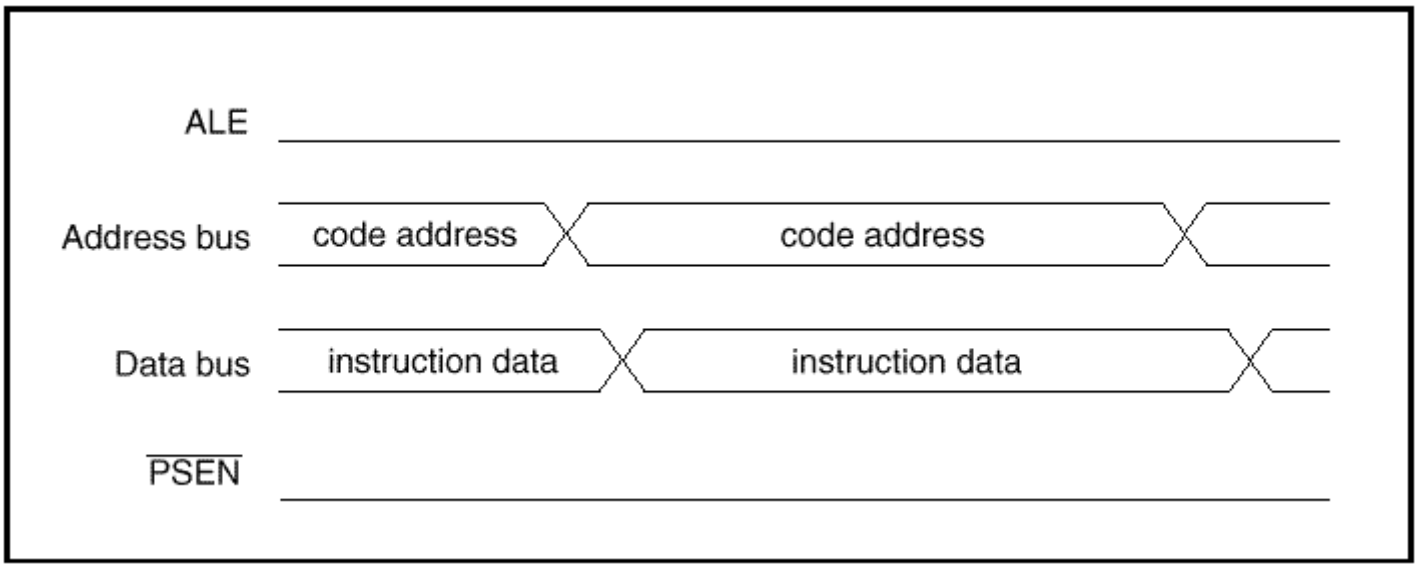

图**2.13** 优化的(连续)外部代码读

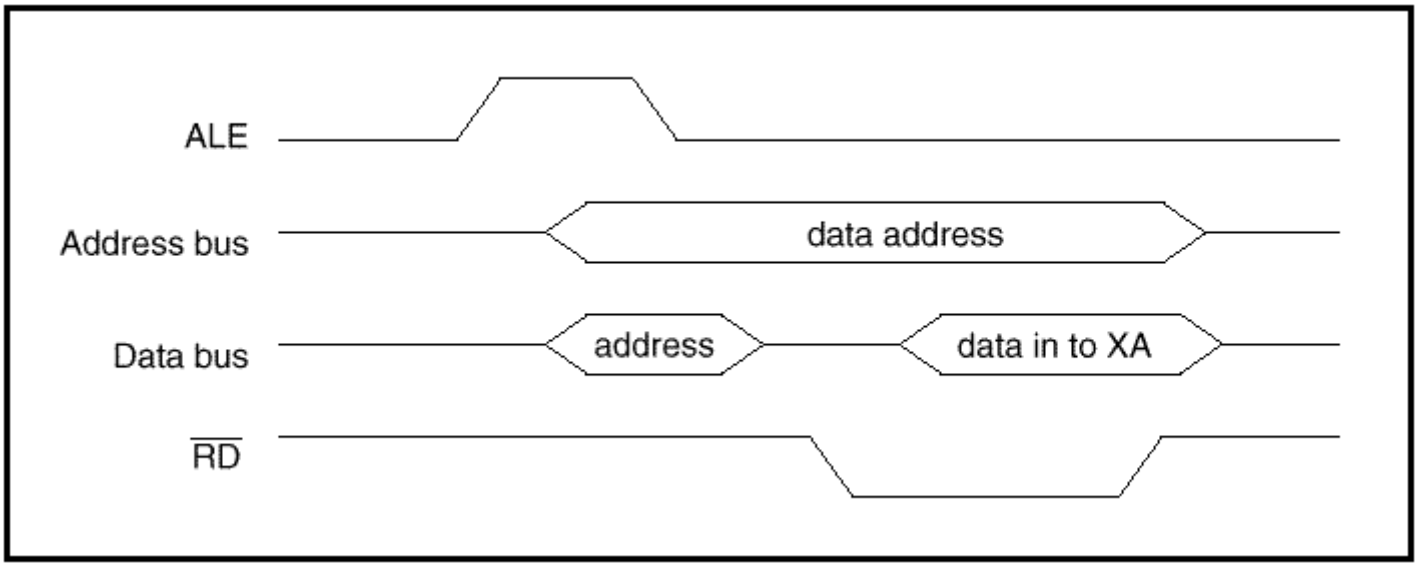

图**2.14** 典型的外部数据读.

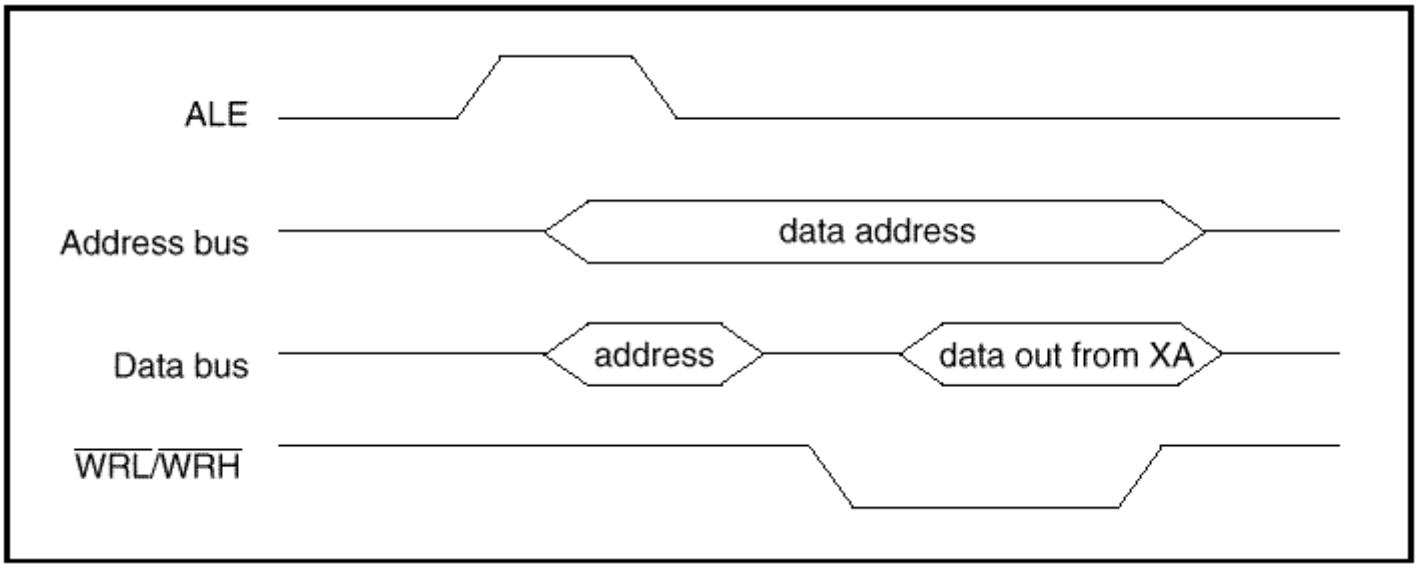

图**2.15** 典型的外部数据写.

2.7 I/O口

同先前的80C51和它的衍生系列相比,XA的I/O口的灵活性能和可编程性能得到加强.在程序中可以通过分配给I/O口的SFR地址来对它进行操作.I/O可以同其它SFR一样进行读或写.

XA赋予了I/O口更加灵活的使用方法,允许不同的输出配置.见图2.16.I/O口可以编程为准双向(80C51型),集开型,推拉型,和高阻型(只能输入).

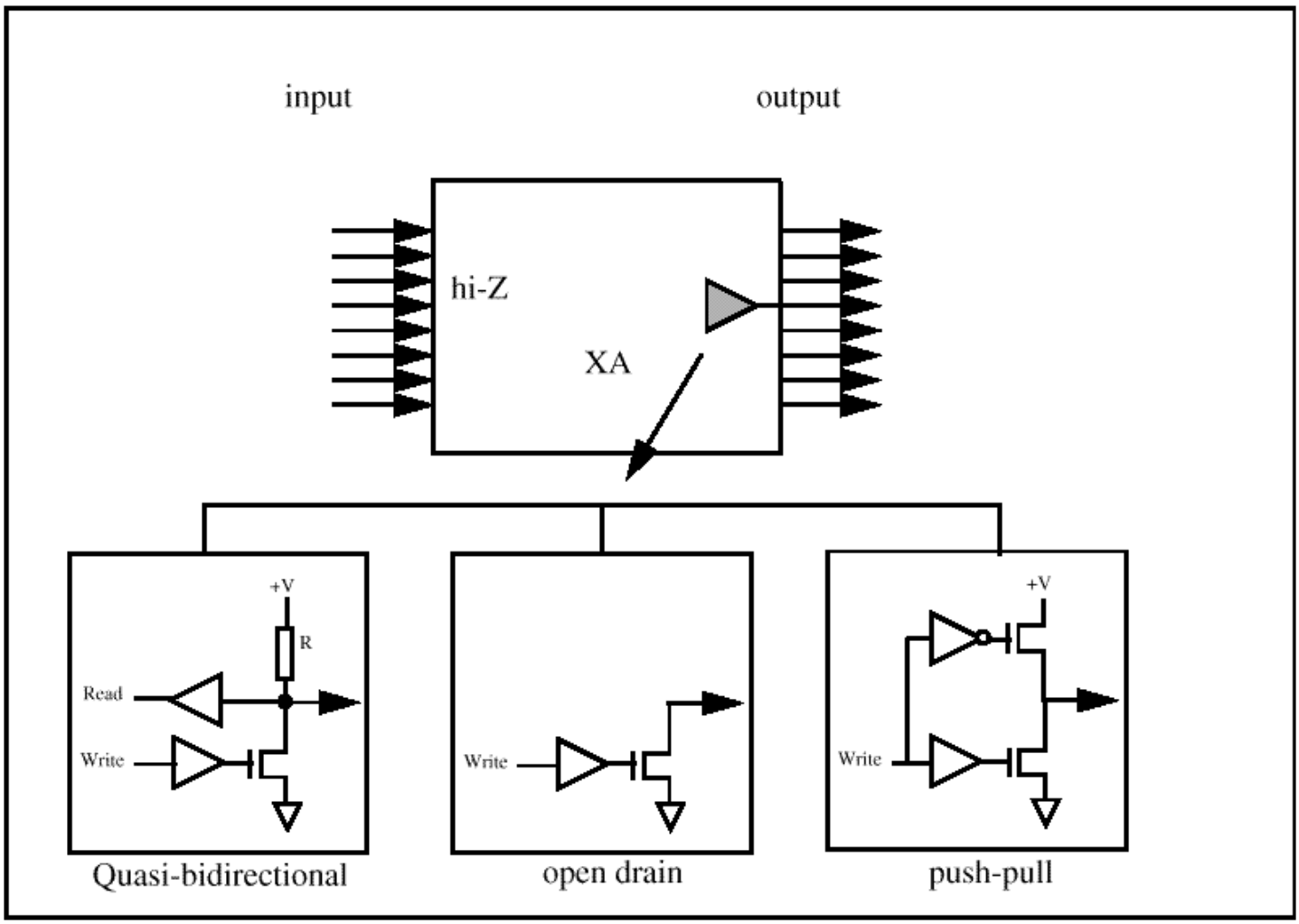

图 **2.16 XA I/O**口的驱动选择

2.8 外设

为了使XA核的系列快速而简单,XA核不包括外设器件,但是它可以通过的特殊功能寄存器总线(又叫SFR总线,它同CPU内部总线不同)附加上.于是,XA系列可以设计一个外设功能块附加到SFR总线上,如果该功能块不存在,那么该地址仍属于XA核.

2.9 80C51兼容性

80C51是当今世界上应用最广泛的8bit微处理器结构,存在巨大的为该系列所写的公用和私用代码.对用户来讲,保护他们在开发代码上的利益是很重要的.通过简单的源代码传输,XA允许现存的80C51代码重新使用在高性能的16位控制器上.同时,XA系列在硬件上向上兼容.80C51的设计,只要在软件和硬件上做极小的改动,就可以移植到XA上.
XA提供了一个80C51的兼容模式,从根本上仿照80C51的寄存器结构,能提供最好的兼容性.运行在变通模式中,XA象一个具有80C51结构的经过优化的16bit微处理器.

在建立XA结构时,已经考虑到大量的兼容性问题.最重要的是在80C51到XA的汇编代码的转换上能实现1:1的转换原则,即一个80C51的指令对应一个XA的指令.

一些特殊的兼容性将在下面的两节中讨论.关于完整的兼容性描述,请见第9章.

2.9.1 软件兼容性

在设计80C51兼容的XA软件时,为了避免过于复杂的XA设计,几个问题应该注意.以下是一些主要问题:

■指令分布.每一条80C51指令将被翻译成一条XA指令.复杂的组合指令尽量避免,因为它们如果被中断分割,将会产生问题.只有一条80C51指令,XCHD(但很少用到),在XA中没有一对一的指令代换.

■相同-相近指令.绝大多数XA指令是80C51的扩展指令集.但是80C51源指令不可能全部相同或相近的包含在XA的指令集中

■时序.为了改善性能,时序必须变化.XA不会试图在时序上同80C51保持兼容;而宁愿通过简单的设计加快指令的执行速度.当80C51的源代码对时序有严格的要求时,用户在代码转换后必须进行时序分析和调整.

■SFR操作.传输SFR的操作相对简单,这是由于SFR通常都是用名称引用的.这样的引用在XA传输时得到保持.假如80C51的源代码来自于一个特殊的型号并用地址来引用SFR,传输器能用XA中相应的SFR来替代,提供80C51和XA均能识别的标识来传递.

2.9.2 硬件兼容性

硬件的目的是制作一个向上兼容的相似结构.

■存储器分布.在XA同80C51硬件兼容问题上,最重要的就是存储器分布.由于XA的每个存储区域(寄存器,数据区域,代码区域)是80C51的扩展集合,因此可以在一定程度上兼容.

■堆栈.功能的变化必然无法避免改变处理器对堆栈的使用发生变化.由于XA支持16位的堆栈操作空间,XA将堆栈的生长方向变为向下,这也是16位处理器的标准,这样才能满足内存中16位变量的有效操作.这对于支持高级语言,例如C也是重要的.

■脚-脚兼容性.XA系列不准备同80C51系列脚-脚的兼容.很多片上的XA外设,性能已经改善,如果保持脚-脚的兼容将限制获得这些性能.一般来讲,外设是同80C51器件向上兼容的,所有的增强性能的添加也是透明的.在这种情况下,80C51代码在80C51功能子集中能够正确工作.

■总线接口.XA的外部总线接口协议同80C51类似.但为了为了提高灵活性,和增强性能,XA总线做了一些改动.在外部总线章节将对二者的差别进行描述.