4. 多重中断的断点保护

多重中断时,每次中断出现的断点都必须保存起来,如图 8.31 中共出现了 3 次中断,有 3 个断点 $k+1$、$l+1$、$m+1$ 需保存。中断系统对断点的保存都是在中断周期内由中断隐指令实现的,对用户是透明的。

断点可以保存在堆栈中,由于堆栈先进后出的特点,因此图 8.31 中的 $k+1$ 先进栈,接着是 $l+1$进栈,最后是 $m+1$ 进栈。出栈时,按相反顺序便可准确返回到程序间断处。

断点也可保存在特定的存储单元内,例如约定一律将程序断点存至主存的 0 号地址单元内。由于保存断点是由中断隐指令自动完成的,因此 3 次中断的断点都将存入 0 地址单元,这势必造成前两次存入的断点 $k+1$ 和 $l+1$ 被冲掉。为此,在中断服务程序中的开中断指令之前,必须先将 0 地址单元的内容转存至其他地址单元中,才能真正保存每一个断点。读者可自行练习,画出将程序断点保存到 0 号地址单元的多重中断服务程序流程。

思考题与习题

8.1 CPU 有哪些功能?画出其结构框图并简要说明每个部件的作用。

8.2 什么是指令周期?指令周期是否有一个固定值?为什么?

8.3 画出指令周期的流程图,分别说明图中每个子周期的作用。

8.4 设 CPU 内有这些部件:PC、IR、SP、AC、MAR、MDR 和 CU。

(1) 画出完成间接寻址的取数指令"LDA @X"(将主存某地址单元的内容取至 AC 中)的数据流(从取指令开始)。

(2) 画出中断周期的数据流。

8.5 中断周期前是什么阶段?中断周期后又是什么阶段?在中断周期 CPU 应完成什么操作?

8.6 存储器中有若干数据类型:指令代码、运算数据、堆栈数据、字符代码和 BCD 码,计算机如何识别这些代码?

8.7 什么叫系统的并行性?粗粒度并行和细粒度并行有何区别?

8.8 什么是指令流水?画出指令二级流水和四级流水的示意图,它们中哪一个更能提高处理器速度,为什么?

8.9 当遇到什么情况时流水线将受阻?举例说明。

8.10 举例说明流水线中的几种数据相关。

8.11 今有四级流水线,分别完成取指(IF)、译码并取数(ID)、执行(EX)、写结果(WR)4 个步骤。假设完成各步操作的时间依次为 90 ns、90 ns、60 ns、45 ns。

(1) 流水线的时钟周期应取何值?

(2) 若相邻的指令发生数据相关,那么第 2 条指令安排推迟多少时间才能不发生错误?

(3) 若相邻两指令发生数据相关,为了不推迟第 2 条指令的执行,可采取什么措施?

8.12 在 5 个功能段的指令流水线中,假设每段的执行时间分别是 10 ns、8 ns、10 ns、10 ns 和 7 ns。对于完成 12 条指令的流水线而言,其加速比为多少?该流水线的实际吞吐率为多少?

8.13 为什么说超长指令字比超标量更能提高并行处理能力?

8.14　指令流水线和运算流水线在结构上有何共同之处？

8.15　什么是中断？设计中断系统需考虑哪些主要问题？

8.16　计算机为了管理中断，在硬件上通常有哪些设置？各有何作用？对指令系统有何考虑？

8.17　在中断系统中，INTR、INT、EINT 这 3 个触发器各有何作用？

8.18　什么是中断隐指令，有哪些功能？

8.19　中断系统中采用屏蔽技术有何作用？

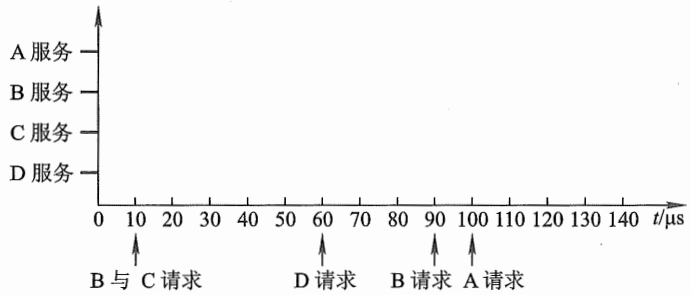
8.20　为实现多重中断，需有哪些硬件支持？

8.21　CPU 在处理中断过程中，有几种方法找到中断服务程序的入口地址？举例说明。

8.22　在中断处理过程中，为什么要进行中断判优？有几种实现方法？若想改变原定的优先顺序，可采取什么措施？

8.23　在中断处理过程中，“保护现场”需要完成哪些任务？如何实现？

8.24　现有 A、B、C、D 4 个中断源，其优先级由高向低按 A→B→C→D 顺序排列。若中断服务程序的执行时间为 20 μs，根据下图所示时间轴给出的中断源请求中断的时刻，画出 CPU 执行程序的轨迹。



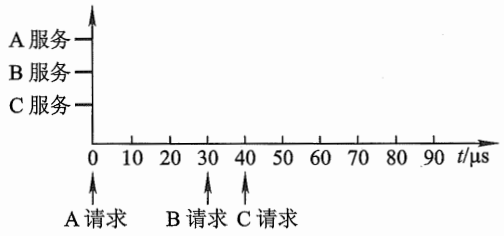
8.25　设某机有 5 个中断源 L_0、L_1、L_2、L_3、L_4，按中断响应的优先次序由高向低排序为 $L_0 \rightarrow L_1 \rightarrow L_2 \rightarrow L_3 \rightarrow L_4$，现要求中断处理次序改为 $L_1 \rightarrow L_4 \rightarrow L_2 \rightarrow L_0 \rightarrow L_3$，根据下面的格式，写出各中断源的屏蔽字。

中断源	屏蔽字				
	0	1	2	3	4
L_0					
L_1					
L_2					
L_3					
L_4					

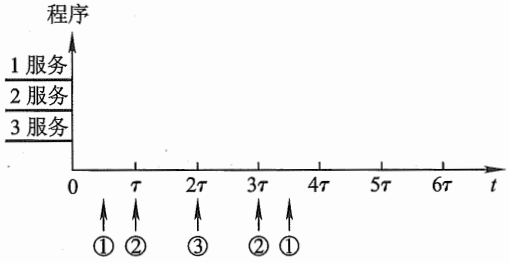
8.26　设某机配有 A、B、C 3 台设备，其优先级按 A→B→C 降序排列，为改变中断处理次序，它们的中断屏蔽字设置如下：

设备	屏蔽字
A	1 1 1
B	0 1 0
C	0 1 1

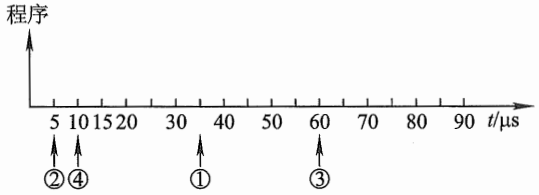
按下图所示时间轴给出的设备请求中断的时刻，画出 CPU 执行程序的轨迹。设 A、B、C 中断服务程序的执行时间均为 20 μs。



8.27　设某机有 3 个中断源，其优先级按 1→2→3 降序排列。假设中断处理时间均为 τ，在下图所示的时间内共发生 5 次中断请求，图中①表示 1 级中断源发出中断请求信号，其余类推，画出 CPU 执行程序的轨迹。



8.28　设某机有 4 个中断源 1、2、3、4，其响应优先级按 1→2→3→4 降序排列，现要求将中断处理次序改为 4→1→3→2。根据下图给出的 4 个中断源的请求时刻，画出 CPU 执行程序的轨迹。设每个中断源的中断服务程序时间均为 20 μs。



第4篇 控制单元

计算机之所以能自动协调地工作,是由于控制单元(CU)的统一指挥。本篇详细分析控制单元的功能及其设计思想。

第9章　控制单元的功能

本章结合指令周期的4个阶段，着重分析控制单元为完成不同指令所发出的各种操作命令——这些命令（又称控制信号）控制计算机的所有部件有次序地完成相应的操作，以达到执行程序的目的——旨在使读者进一步理解指令周期、机器周期、时钟周期（节拍）和控制信号的关系，进一步体会控制单元在机器运行中所起到的核心作用，为下一章控制单元的设计打好基础。

9.1　微操作命令的分析

控制单元具有发出各种微操作命令（即控制信号）序列的功能。

概括地说，计算机的功能就是执行程序。在执行程序的过程中，控制单元要发出各种微操作命令，而且不同的指令对应不同的命令。进一步分析发现，完成不同指令的过程中，有些操作是相同或相似的，如取指令、取操作数地址（当间接寻址时）以及进入中断周期由中断隐指令完成的一系列操作。为更清晰起见，下面按指令周期的4个阶段进一步分析其对应的微操作命令。

9.1.1　取指周期

为了便于讨论，假设CPU内有4个寄存器，如图8.10所示。MAR与地址总线相连，存放欲访问的存储单元地址；MDR与数据总线相连，存放欲写入存储器的信息或最近从存储器中读出的信息；PC存放现行指令的地址，有计数功能；IR存放现行指令。取指令的过程可归纳为以下几个操作。

① 现行指令地址送至存储器地址寄存器，记作PC→MAR。

② 向主存发送读命令，启动主存做读操作，记作1→R。

③ 将MAR（通过地址总线）所指的主存单元中的内容（指令）经数据总线读至MDR内，记作M(MAR)→MDR。

④ 将MDR的内容送至IR，记作MDR→IR。

⑤ 指令的操作码送至CU译码，记作OP(IR)→CU。

⑥ 形成下一条指令的地址，记作(PC)+1→PC。

9.1.2 间址周期

间址周期完成取操作数有效地址的任务,具体操作如下。

① 将指令的地址码部分(形式地址)送至存储器地址寄存器,记作 Ad(IR)→MAR。

② 向主存发送读命令,启动主存做读操作,记作 1→R。

③ 将 MAR (通过地址总线)所指的主存单元中的内容(有效地址)经数据总线读至 MDR 内,记作 M(MAR)→MDR。

④ 将有效地址送至指令寄存器的地址字段,记作 MDR→Ad(IR)。此操作在有些机器中可省略。

9.1.3 执行周期

不同指令执行周期的微操作是不同的,下面分别讨论非访存指令、访存指令和转移类指令的微操作。

1. 非访存指令

这类指令在执行周期不访问存储器。

(1) 清除累加器指令 CLA

该指令在执行阶段只完成清除累加器操作,记作 0→ACC。

(2) 累加器取反指令 COM

该指令在执行阶段只完成累加器内容取反,结果送累加器的操作,记作$\overline{\text{ACC}}$→ACC。

(3) 算术右移一位指令 SHR

该指令在执行阶段只完成累加器内容算术右移一位的操作,记作 L(ACC)→R(ACC),ACC_0→ACC_0(ACC 的符号位不变)。

(4) 循环左移一位指令 CSL

该指令在执行阶段只完成累加器内容循环左移一位的操作,记作 R(ACC)→L(ACC),ACC_0→ACC_n(或 ρ^{-1}(ACC))。

(5) 停机指令 STP

计算机中有一个运行标志触发器 G,当 G=1 时,表示机器运行;当 G=0 时,表示停机。STP 指令在执行阶段只需将运行标志触发器置“0”,记作 0→G。

2. 访存指令

这类指令在执行阶段都需要访问存储器。为简单起见,这里只考虑直接寻址的情况,不考虑其他寻址方式。

(1) 加法指令 ADD X

该指令在执行阶段需要完成累加器内容与对应于主存 X 地址单元的内容相加,结果送累加

器的操作,具体如下:

① 将指令的地址码部分送至存储器地址寄存器,记作 Ad(IR)→MAR。

② 向主存发读命令,启动主存做读操作,记作 1→R。

③ 将 MAR(通过地址总线)所指的主存单元中的内容(操作数)经数据总线读至 MDR 内,记作 M(MAR)→MDR。

④ 给 ALU 发送加命令,将 ACC 的内容和 MDR 的内容相加,结果存于 ACC,记作 (ACC) + (MDR) →ACC。

当然,也有的加法指令指定两个寄存器的内容相加,如“ADD AX,BX”,该指令在执行阶段无须访存,只需完成 (AX) + (BX) →AX 的操作。

(2) 存数指令 STA X

该指令在执行阶段需将累加器 ACC 的内容存于主存的 X 地址单元中,具体操作如下。

① 将指令的地址码部分送至存储器地址寄存器,记作 Ad(IR)→MAR。

② 向主存发写命令,启动主存做写操作,记作 1→W。

③ 将累加器内容送至 MDR,记作 ACC→MDR。

④ 将 MDR 的内容(通过数据总线)写入 MAR(通过地址总线)所指的主存单元中,记作 MDR→M(MAR)。

(3) 取数指令 LDA X

该指令在执行阶段需将主存 X 地址单元的内容取至累加器 ACC 中,具体操作如下。

① 将指令的地址码部分送至存储器地址寄存器,记作 Ad(IR)→MAR。

② 向主存发读命令,启动主存作读操作,记作 1→R。

③ 将 MAR(通过地址总线)所指的主存单元中的内容(操作数)经数据总线读至 MDR 内,记作 M(MAR)→MDR。

④ 将 MDR 的内容送至 ACC,记作 MDR→ACC。

3. 转移类指令

这类指令在执行阶段也不访问存储器。

(1) 无条件转移指令 JMP X

该指令在执行阶段完成将指令的地址码部分 X 送至 PC 的操作,记作 Ad(IR)→PC。

(2) 条件转移(负则转)指令 BAN X

该指令根据上一条指令运行的结果决定下一条指令的地址,若结果为负(累加器最高位为 1,即 $A_0=1$),则指令的地址码送至 PC,否则程序按原顺序执行。由于在取指阶段已完成了(PC)+1→PC,所以当累加器结果不为负(即 $A_0=0$)时,就按取指阶段形成的 PC 执行,记作 $A_0 \cdot Ad(IR) + \overline{A}_0 \cdot (PC) \rightarrow PC$。

由此可见,不同指令在执行阶段所完成的操作是不同的。如果将访存指令分为直接访存和间接访存两种,则上述三类指令的指令周期如图 9.1 所示。

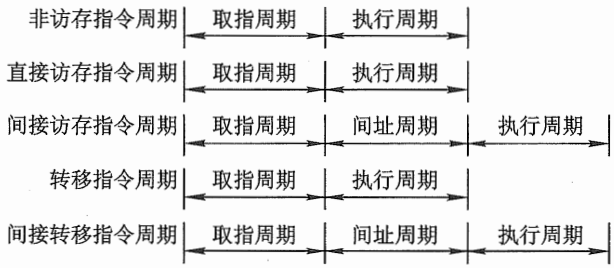


图 9.1 三类指令的指令周期

9.1.4 中断周期

在执行周期结束时刻,CPU 要查询是否有请求中断的事件发生,如果有则进入中断周期。由 8.4.4 节可知,在中断周期,由中断隐指令自动完成保护断点、寻找中断服务程序入口地址以及硬件关中断的操作。假设程序断点存至主存的 0 地址单元,且采用硬件向量法寻找入口地址,则在中断周期需完成如下操作。

① 将特定地址"0"送至存储器地址寄存器,记作 0→MAR。

② 向主存发写命令,启动存储器作写操作,记作 1→W。

③ 将 PC 的内容(程序断点)送至 MDR,记作 PC→MDR。

④ 将 MDR 的内容(程序断点)通过数据总线写入 MAR(通过地址总线)所指示的主存单元(0 地址单元)中,记作 MDR→M(MAR)。

⑤ 将向量地址形成部件的输出送至 PC,记作向量地址→PC,为下一条指令的取指周期做准备。

⑥ 关中断,将允许中断触发器清零,记作 0→EINT(该操作可直接由硬件线路完成,参见图 8.30)。

如果程序断点存入堆栈,而且进栈操作是先修改栈指针,后存入数据(参见图 7.18),只需将上述① 改为(SP)-1→SP,且 SP→MAR。

上述所有操作都是在控制单元发出的控制信号(即微操作命令)控制下完成的。

9.2　控制单元的功能

9.2.1　控制单元的外特性

图 9.2 是反映控制单元外特性的框图。

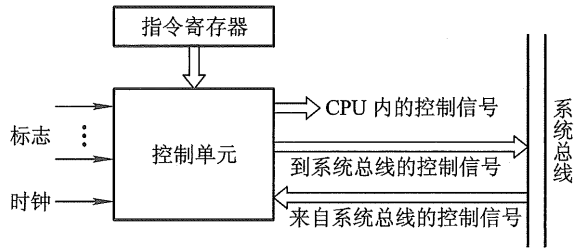


图 9.2　控制单元外特性

1. 输入信号

(1) 时钟

上述各种操作有以下两点应特别注意。

① 完成每个操作都需占用一定的时间。

② 各个操作是有先后顺序的。例如,存储器读操作要用到 MAR 中的地址,故 PC→MAR 应先于 M(MAR)→MDR。

为了使控制单元按一定的先后顺序、一定的节奏发出各个控制信号,控制单元必须受时钟控制,即每一个时钟脉冲使控制单元发送一个操作命令,或发送一组需要同时执行的操作命令。

(2) 指令寄存器

现行指令的操作码决定了不同指令在执行周期所需完成的不同操作,故指令的操作码字段是控制单元的输入信号,它与时钟配合可产生不同的控制信号。

(3) 标志

控制单元有时需依赖 CPU 当前所处的状态(如 ALU 操作的结果)产生控制信号,如 BAN 指令,控制单元要根据上条指令的结果是否为负而产生不同的控制信号。因此“标志”也是控制单元的输入信号。

(4) 来自系统总线(控制总线)的控制信号

例如,中断请求、DMA 请求。

2. 输出信号

(1) CPU 内的控制信号

主要用于 CPU 内的寄存器之间的传送和控制 ALU 实现不同的操作。

(2) 送至系统总线(控制总线)的信号

例如,命令主存或 I/O 读/写、中断响应等。

9.2.2 控制信号举例

控制单元的主要功能就是能发出各种不同的控制信号。下面以间接寻址的加法指令“ADD @X”为例,进一步理解控制信号在完成一条指令的过程中所起的作用。

1. 不采用 CPU 内部总线的方式

图 9.3 示意了未采用 CPU 内部总线方式的数据通路和控制信号的关系。图中未画出每个寄存器的输入或输出控制门,但标出了控制这些门电路的控制信号 C_i,考虑到从存储器取出的指令或有效地址都先送至 MDR 再送至 IR,故这里省去了 IR 送至 MAR 的数据通路,凡是需要从 IR 送至 MAR 的操作均可由 MDR 送至 MAR 代替。

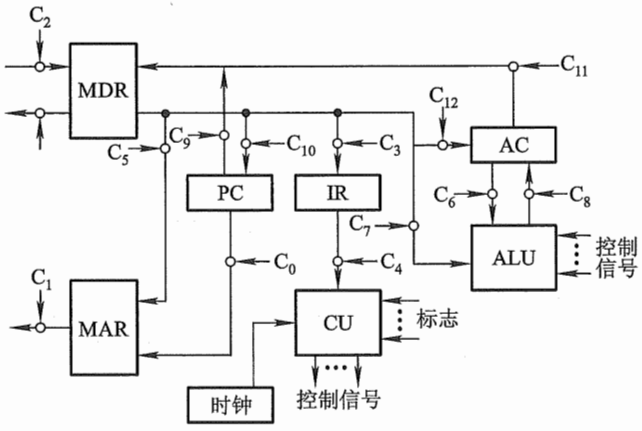


图 9.3 未采用 CPU 内部总线方式的数据通路和控制信号

(1) 取指周期

① 控制信号 C_0 有效,打开 PC 送往 MAR 的控制门。

② 控制信号 C_1 有效,打开 MAR 送往地址总线的输出门。

③ 通过控制总线向主存发读命令。

④ C_2 有效,打开数据总线送至 MDR 的输入门。

⑤ C_3 有效,打开 MDR 和 IR 之间的控制门,至此指令送至 IR。

⑥ C_4 有效,打开指令操作码送至 CU 的输出门。CU 在操作码和时钟的控制下,可产生各种

控制信号。

⑦ 使 PC 内容加 1(图中未标出)。

(2) 间址周期

① C_5 有效,打开 MDR 和 MAR 之间的控制门,将指令的形式地址送至 MAR。

② C_1 有效,打开 MAR 送往地址总线的输出门。

③ 通过控制总线向主存发读命令。

④ C_2 有效,打开数据总线送至 MDR 的输入门,至此,有效地址存入 MDR。

⑤ C_3 有效,打开 MDR 和 IR 之间的控制门,将有效地址送至 IR 的地址码字段。

(3) 执行周期

① C_5 有效,打开 MDR 和 MAR 之间的控制门,将有效地址送至 MAR。

② C_1 有效,打开 MAR 送往地址总线的输出门。

③ 通过控制总线向主存发读命令。

④ C_2 有效,打开数据总线送至 MDR 的输入门,至此,操作数存入 MDR。

⑤ C_6、C_7 同时有效,打开 AC 和 MDR 通往 ALU 的控制门。

⑥ 通过 CPU 内部控制总线对 ALU 发“ADD”加控制信号,完成 AC 的内容和 MDR 的内容相加。

⑦ C_8 有效,打开 ALU 通往 AC 的控制门,至此将求和结果存入 AC。

图中 C_9 和 C_{10}分别是控制 PC 的输出和输入的控制信号,C_{11}和 C_{12}分别是控制 AC 的输出和输入的控制信号。

2. 采用 CPU 内部总线的方式

图 9.4 示意了采用 CPU 内部总线方式的数据通路和控制信号的关系,图中每一个小圈处都有一个控制信号,它控制寄存器到总线或总线到寄存器之间的传送。例如,IR_i 表示控制从内部总线到指令寄存器的输入控制门;PC_o 表示控制从程序计数器到内部总线的输出控制门。下标为 i 表示输入控制,下标为 o 表示输出控制,以此类推。与图 9.3 相比,图 9.4 多了两个寄存器 Y 和 Z,这是由于 ALU 是一个组合逻辑电路,在其运算过程中必须保持两个输入端不变,其中一个输入可以从 Y 寄存器中获得,另一个输入可以从内部总线上获得。当 CPU 内有多个通用寄存器时,由于设置了寄存器 Y,可实现任意两个寄存器之间的算逻运算。此外,ALU 的输出不能直接与内部总线相连,因为其输出又会通过总线反馈到 ALU 的输入,影响运算的正确性,故用寄存器 Z 暂存运算结果,再根据需要送至指定的目标。

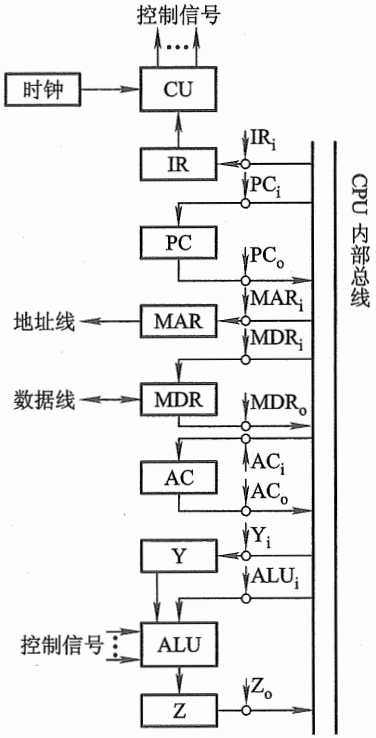


图 9.4　采用 CPU 内部总线方式的数据通路和控制信号

下面仍以完成间接寻址的加法指令“ADD @X”为例,分析控制单元发出的控制信号。

(1) 取指周期

① PC_o 和 MAR_i 有效,完成 PC 经内部总线送至 MAR 的操作,即 PC→MAR。

② 通过控制总线(图中未画出)向主存发读命令,即 1→R。

③ 存储器通过数据总线将 MAR 所指单元的内容(指令)送至 MDR。

④ MDR_o 和 IR_i 有效,将 MDR 的内容送至 IR,即 MDR→IR,至此,指令送至 IR,其操作码字段开始控制 CU。

⑤ 使 PC 内容加 1(图中未标出)。

(2) 间址周期

① MDR_o 和 MAR_i 有效,将指令的形式地址经内部总线送至 MAR,即 MDR→MAR。

② 通过控制总线向主存发读命令,即 1→R。

③ 存储器通过数据总线将 MAR 所指单元的内容(有效地址)送至 MDR。

④ MDR_o 和 IR_i 有效,将 MDR 中的有效地址送至 IR 的地址码字段,即 MDR→Ad(IR)。

(3) 执行周期

① MDR_o 和 MAR_i 有效,将有效地址经内部总线送至 MAR,即 MDR→MAR。

② 通过控制总线向主存发读命令,即 1→R。

③ 存储器通过数据总线将 MAR 所指单元的内容(操作数)送至 MDR。

④ MDR_o 和 Y_i 有效,将操作数送至 Y,即 MDR→Y。

⑤ AC_o 和 ALU_i 有效,同时 CU 向 ALU 发“ADD”加控制信号,使 AC 的内容和 Y 的内容相加(Y 的内容送至 ALU 不必通过总线),结果送寄存器 Z,即 (AC) + (Y) →Z。

⑥ Z_o 和 AC_i 有效,将运算结果存入 AC,即 Z→AC。

现代计算机的 CPU 都集成在一个硅片内,在芯片内采用内部总线的方式可大大节省芯片内部寄存器之间的连线,使芯片内各部件布局更合理。

例 9.1 设 CPU 内部采用非总线结构,如图 9.3 所示。

(1) 写出取指周期的全部微操作。

(2) 写出取数指令“LDA M”、存数指令“STA M”、加法指令“ADD M”(M 均为主存地址)在执行阶段所需的全部微操作。

(3) 当上述指令均为间接寻址时,写出执行这些指令所需的全部微操作。

(4) 写出无条件转移指令“JMP Y”和结果为零则转指令“BAZ Y”在执行阶段所需的全部微操作。

解:(1) 取指周期的全部微操作如下:

PC→MAR	现行指令地址→MAR
1→R	命令存储器读
M(MAR)→MDR	现行指令从存储器中读至 MDR
MDR→IR	现行指令→IR

OP(IR)→CU　　指令的操作码→CU 译码

(PC) +1→PC　　形成下一条指令的地址

(2) ① 取数指令“LDA M”执行阶段所需的全部微操作如下：

Ad(IR)→MAR　　指令的地址码字段→MAR

1→R　　命令存储器读

M(MAR)→MDR　　操作数从存储器中读至 MDR

MDR→ACC　　操作数→ACC

② 存数指令“STA M”执行阶段所需的全部微操作如下：

Ad(IR)→MAR　　指令的地址码字段→MAR

1→W　　命令存储器写

ACC→MDR　　欲写入的数据→MDR

MDR→M(MAR)　　数据写至存储器中

③ 加法指令“ADD M”执行阶段所需的全部微操作如下：

Ad(IR)→MAR　　指令的地址码字段→MAR

1→R　　命令存储器读

M(MAR)→MDR　　操作数从存储器中读至 MDR

(ACC) + (MDR)→ACC　　两数相加结果送 ACC

(3) 当上述指令为间接寻址时，需增加间址周期的微操作。这 3 条指令在间址周期的微操作是相同的，即

Ad(IR)→MAR　　指令的地址码字段→MAR

1→R　　命令存储器读

M(MAR)→MDR　　有效地址从存储器中读至 MDR

进入执行周期，3 条指令的第一个微操作均为 MDR→MAR(有效地址送 MAR)，其余微操作不变。

(4) ① 无条件转移指令“JMP Y”执行阶段的微操作如下：

Ad(IR)→PC　　转移(目标)地址 Y→PC

② 结果为零则转指令“BAZ Y”执行阶段的微操作如下：

Z · Ad(IR)→PC　　当 Z=1 时，转移(目标)地址 Y→PC

(Z 为标记触发器，结果为 0 时 Z=1)

例 9.2　已知单总线计算机结构如图 9.5 所示，其中 M 为主存，XR 为变址寄存器，EAR 为有效地址寄存器，LATCH 为锁存器。图中各寄存器的输入和输出均受控制信号控制，例如，PC_i 表示 PC 的输入控制信号，MDR_o 表示 MDR 的输出控制信号。假设指令地址已存于 PC 中，画出“ADD X,D”(X 为变址寄存器 XR，D 为形式地址)和“STA * D”(* 表示相对寻址，D 为相对位移量)两条指令的指令周期信息流程图，并列出相应的控制信号序列。

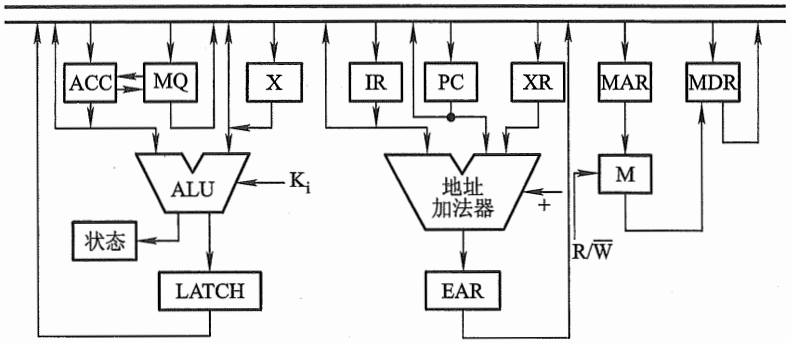


图 9.5　单总线计算机结构

解:(1)“ADD X,D”指令取指周期和执行周期的信息流程及相应的控制信号如图 9.6 所示,图中 Ad(IR)为形式地址。

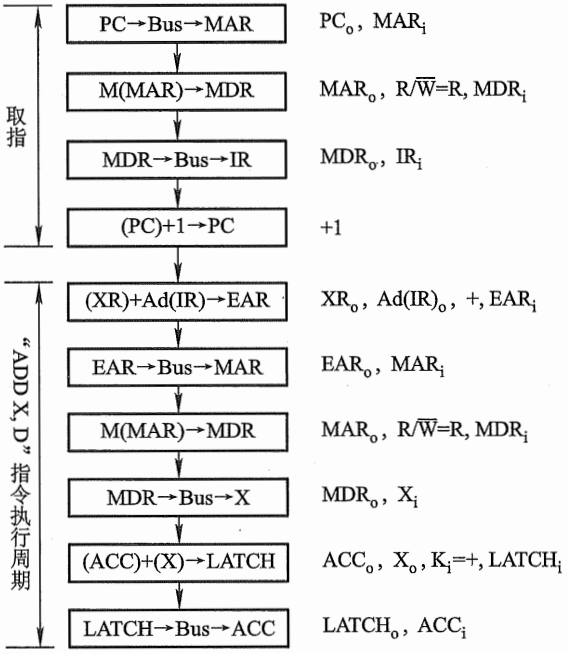


图 9.6　“ADD X,D”指令周期的信息流程及相应的控制信号

(2)“STA ＊D”指令取指周期和执行周期的信息流程及相应的控制信号如图 9.7 所示,图中 Ad(IR)为相对位移量的机器代码。