硬布线控制器又称组合逻辑控制器，以使用最少元件和取得最高操作速度为设计目标。硬布线控制器将控制部件做成产生专门固定时序控制信号的逻辑电路，产生各种控制信号。一旦控制部件构成后，除非重新设计和物理上对它重新布线，否则很难增加新的控制功能。硬布线控制器是计算机中最复杂的逻辑部件之一。当执行不同的机器指令时，通过激活一系列彼此很不相同的控制信号来实现对指令的解释，其结果使得控制器往往很少有明确的结构而变得杂乱无章。结构上的这种缺陷使得硬布线控制器的设计和调试非常复杂且代价很大。正因为如此，历史上有一段时间，硬布线控制器被微程序控制器所取代。但是，与微程序控制器相比，硬布线控制的速度较快。其原因是微程序控制中每条微指令都要从控制存储器中读取一次，影响了速度，而硬布线控制主要取决于电路延迟。另外，随着新一代机器及 VLSI 技术的发展与不断进步，硬布线逻辑设计思想又得到了重视，现代新型计算机体系结构如 RISC 中多采用硬布线控制逻辑。

硬布线控制器逻辑设计中注意的事项

(1) 采用适宜指令格式，合理分配指令操作码；

(2) 确定机器周期、节拍与主频；

(3) 确定机器周期数及一周期内的操作；

(4) 进行指令综合； 综合所有指令的每一个操作命令，写出逻辑表达式，并进行化简。

(5) 明确组合逻辑电路。 将简化后的逻辑表达式用组合逻辑电路来实现。操作命令的控制信号先用逻辑表达式列出，进行化简，考虑各种条件的约束，合理选用逻辑门电路、触发器等器件，采用组合逻辑电路的设计方法产生控制信号。

总之，控制信号的设计与实现，技巧性较强，目前已有一些专门的开发系统或工具供逻辑设计使用，但是，对全局的考虑主要依靠设计人员的智慧和经验实现。

硬布线控制器与微程序控制器的比较

硬布线控制器与微程序控制器相比较，在操作控制信号的形成上有较大的区别外，其它没有本质的区别。对于实现相同的一条指令，不管是采用硬布线控制还是采用微程序控制技术，都可以采用多种逻辑设计方案，导致了各种不同的控制器在具体实现方法和手段上的区别，性能差异。

硬布线控制与微程序控制的主要区别归纳为如下方面：

(1) 实现方式

微程序控制器的控制功能是在存放微程序存储器和存放当前正在执行的微指令的寄存器直接控制下实现的，而硬布线控制的功能则由逻辑门组合实现。微程序控制器的电路比较规整，各条指令信号的差别集中在控制存储器内容上，

因此，无论是增加或修改指令都只要增加或修改控制存储器内容即可，若控制存储器是 ROM，则要更换芯片，在设计阶段可以先用 RAM 或 EPROM 来实现，验证正确后或成批生产时，再用 ROM 代替。硬布线控制器的控制信号先用逻辑式列出，经化简后用电路来实现，因此，显得零乱复杂，当需要修改指令或增加指令时就必须重新设计电路，非常麻烦而且有时甚至无法改变。因此，微操作控制取代了硬布线控制并得到了广泛应用，尤其是指令复杂的计算机，一般都采用微程序来实现控制功能。

(2) 性能方面

在同样的半导体工艺条件下，微程序控制的速度比硬布线控制的速度低，因为执行每条微程序指令都要从控制存储器中读取，影响了速度；而硬布线控制逻辑主要取决于电路延时，因而在超高速机器中，对影响速度的关键部分如核心部件 CPU，往往采用硬布线逻辑实现。近年来，在一些新型计算机系统中，例如，RISC(精简指令系统计算机)中，一般都选用硬布线逻辑电路。

本章通过硬布线单周期 CPU 和多周期 CPU 设计，讲述硬布线控制器设计的方法。

1. 单周期 CPU 逻辑设计

单周期 CPU 的特点是每条指令的执行需要一个时钟周期，一条指令执行完再执行下一条指令。假设该单周期 CPU 能够处理的指令格式如下：

(1) ADD rs,rt,immediate

000000	rs(5 位)	rt(5 位)	rd(5 位)	reserved

完成功能：rd←rs+rt。

(2)LW rt,offset(base)

100011	base(5 位)	rt(5 位)	Offset (16 位)

完成功能：16 位 offset 经符号扩展到 32 位，与寄存器 base 中值相加，结果作为地址，以此地址取字保存到寄存器 rt。

(3)SW rt, offset(base)

101011	base(5 位)	rt(5 位)	Offset (16 位)

完成功能为：16 位 offset 经符号扩展到 32 位，与寄存器 base 中值相加，结果作为地址，寄存器 rt 值保存到此地址内存中。

(4) J target

000010	target (26 位)

完成功能为：26 位地址经零扩展到 32 位成跳转的目标地址。

一般来说，一个 CPU 在处理指令时需要经过以下几个步骤：

1）取指令（IF）：根据程序计数器 PC 中的指令地址，从存储器中取出一条指令，然后转到译码状态。同时，在 PC 中产生取下一条指令需要的指令地址。

2）指令译码（ID）：对取指令操作中得到的指令进行译码，确定这条指令需要完成的操作，从而产生相应的控制信号，驱动执行状态中的各种动作。

3）指令执行（EXE）：根据指令译码得到的控制信号，具体地执行指令动作，然后，转移到结果写回状态。

4）存储器访问（MEM）：所有需要访问存储器的操作都将在这个步骤中执行，该步骤

给出访问存储器的数据地址，把数据写入到存储器中数据地址所指示的位置或者从存储器中的得到数据地址所指示的数据。

5）结果写回（WB）：该步骤负责把指令执行的结果或者访问存储器中得到的数据写回到相应的目的寄存器中。

图 3.1　单周期 CPU 指令处理过程

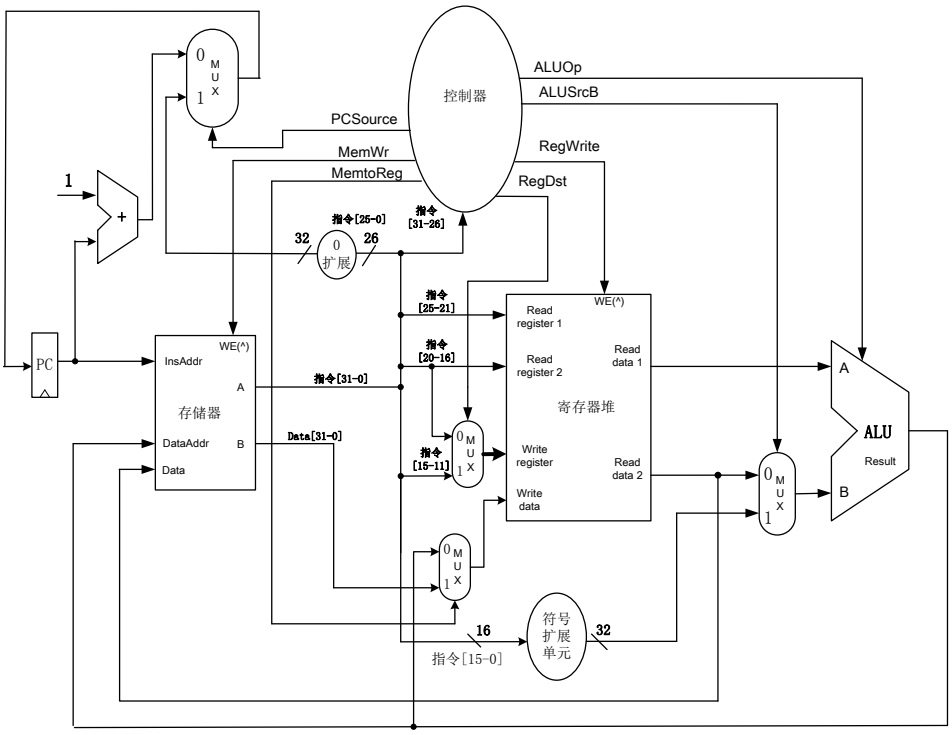

图 1. 单周期 CPU 数据通路和控制线路图

图 1 是一个非常简单的能够单周期完成上述 4 条指令的数据通路和必要的控制线路图。其中存储器和寄存器堆，读操作时，给出地址，输出端就直接输出相应数据；而在写操作时，在 WE 使能信号为 1 时，在时钟上升沿写入。存储器为双端口 RAM，因此当 PC 给出地址 InsAdd 并且不变时，A 端口就输出稳定的指令信号。B 端口地址由 DataAddr 确定。存储器写入地址为 DataAddr，数据为 Data。图中控制信号动作由下表所示。

表 1. 控制信号作用

信号名	无效时作用(0)	有效时作用(1)
RegDst	写寄存器在寄存器堆的地址来自于 rt 字段	写寄存器在寄存器堆的地址来自于 rd 字段
ALUSrcB	寄存器堆 Data2 输出	符号扩展的立即数
ALUOp	无	加法
RegWrite	无	在时钟上升沿时，写寄存器
MemtoReg	送往寄存器堆写数据输入的值来自 ALU	送往寄存器堆写数据输入的值来自存

		储器
MemWr	无	在时钟上升沿时，写存储器
PCSource	PC+1	扩展的立即数

在 CPU 和控制器设计过程中，一个非常重要的工作就是理顺数据在 CPU 各个部分之间的数据通路，从而可以设计控制器。下面从四个图可以看出四条指令执行过程中不同的数据通路，粗线为当前有效通路，控制器上斜体为当前有效控制信号。

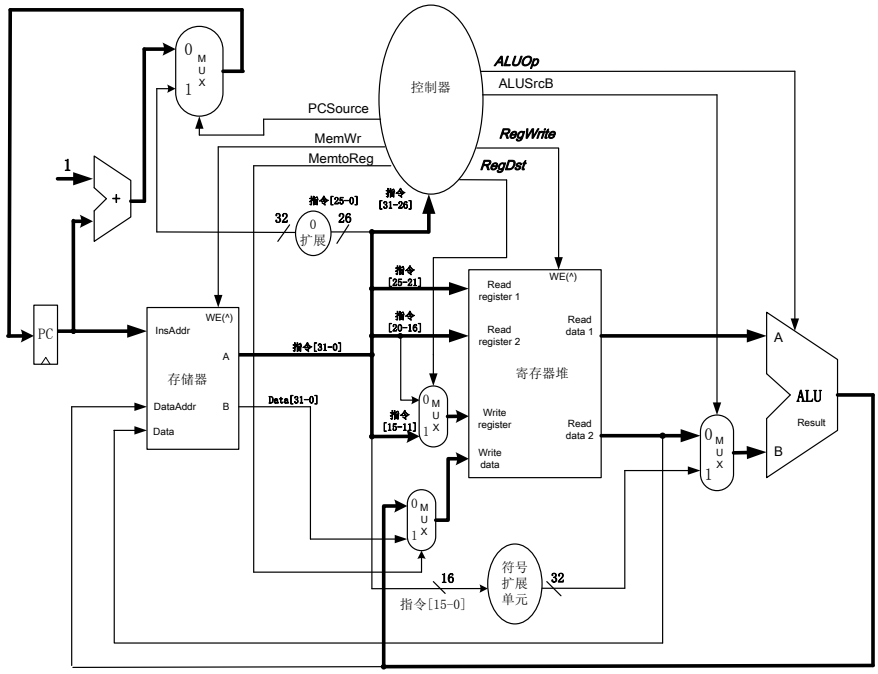

图 2. ADD 指令处理过程图

从图 2 可以看到，经过一个时钟后 PC 自动完成加 1。从存储器取出指令，[31..26]进入控制器译码产生控制信号，[25..21]和[20..16]作为寄存器地址从寄存器堆中取出两加数到 ALU，ALU 运算结果在时钟下一个上升沿写回寄存器地址[15..11]。因此有效控制信号 RegDst、ALUOp、RegWrite 均为 1，其余控制信号为 0。

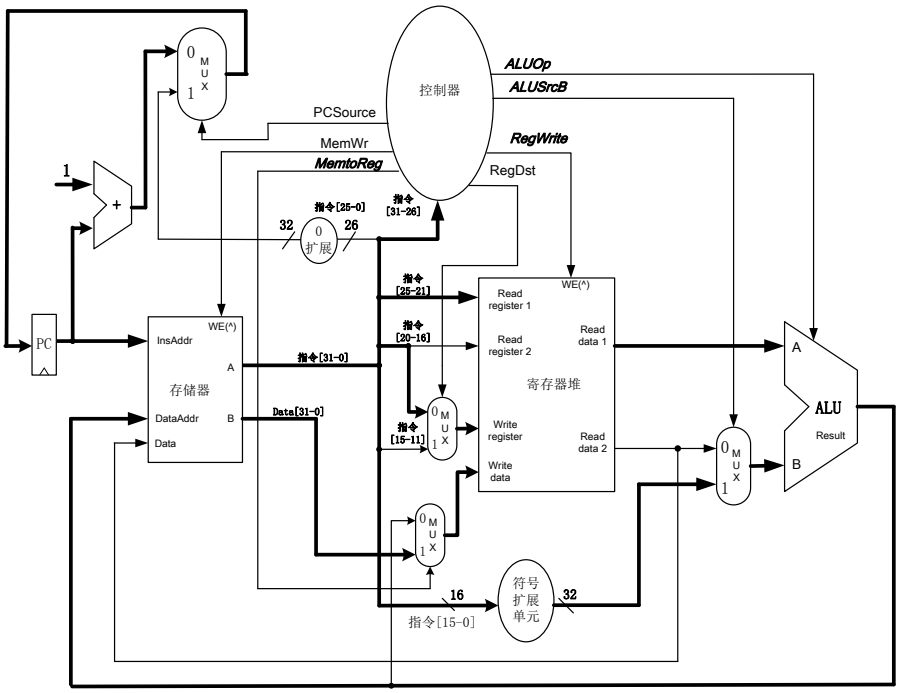

图 3. LW 指令处理过程图

从图 3 可以看到，指令[31..26]进入控制器译码产生控制信号，[25..21]作为寄存器地址读出基地址，[15..0]立即数符号扩展，与基地址在 ALU 相加，作为存储器地址。存储器从 B 口读出数据，在时钟下一个上升沿写到寄存器堆地址[20..16]中。因此有效控制信号 ALUSrcB、ALUOp、RegWrite、MemtoReg 均为 1，其余控制信号为 0。

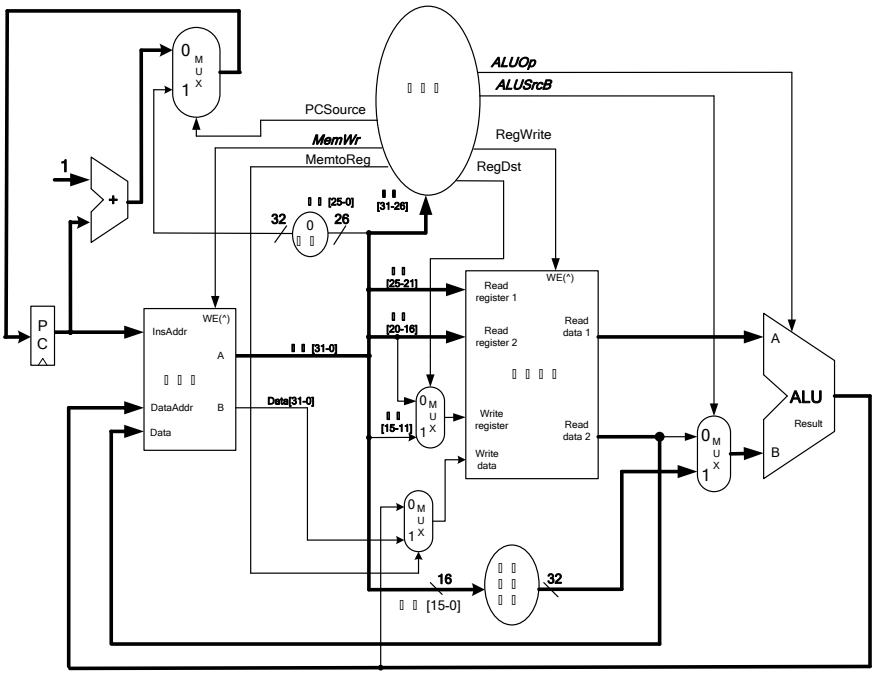

图 4. SW 指令处理过程图

从图 4 可以看到，指令[31..26]进入控制器译码产生控制信号，[25..21]作为寄存器地址读出基地址，[15..0]立即数符号扩展，与基地址在 ALU 相加，作为存储器地址。寄存器堆

地址[20..16]读出数据到存储器的 Data 端，在时钟下一个上升沿保存。因此有效控制信号 ALUSrcB、ALUOp、MemWr 均为 1，其余控制信号为 0。

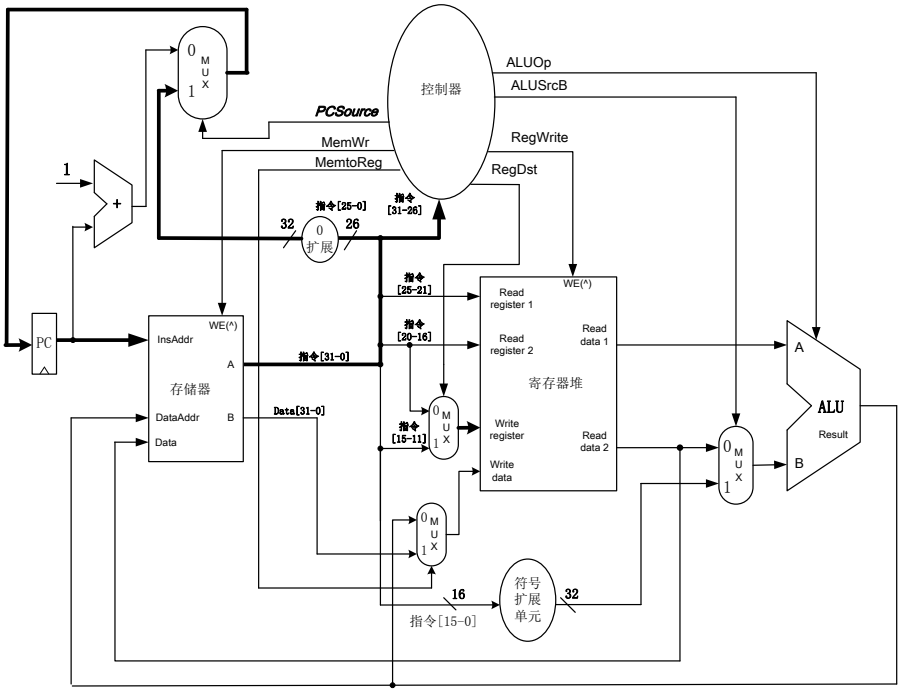

图 5. J 指令处理过程图

从图 5 可以看到，指令[31..26]进入控制器译码产生控制信号，立即数[25..0]扩展为 32 位，在时钟下一个上升沿保存到 PC，完成跳转。因此有效控制信号 PCSource 为 1，其余控制信号为 0。

清楚了每条指令的执行过程，现在就可以设计控制器了。综合前述分析，得到表 2。

控制信号	ADD	LW	SW	J
RegDst	1	0	0	0
ALUSrcB	0	1	1	0
ALUOp	1	1	1	0
RegWrite	1	1	0	0
MemtoReg	0	1	0	0
MemWr	0	0	1	0
PCSource	0	0	0	1

由于 ADD 指令码为 000000，LW 指令码为 100011，SW 指令码为 101011，J 指令码为 000010，因此

RegDst=ADD = $\overline{Op_5}\cdot\overline{Op_4}\cdot\overline{Op_3}\cdot\overline{Op_2}\cdot\overline{Op_1}\cdot\overline{Op_0}$

ALUSrcB =LW+SW= $Op_5\cdot\overline{Op_4}\cdot\overline{Op_3}\cdot\overline{Op_2}\cdot Op_1\cdot Op_0 + Op_5\cdot\overline{Op_4}\cdot Op_3\cdot\overline{Op_2}\cdot Op_1\cdot Op_0$

类似地，可以写出所有控制信号的组合逻辑，经化简后，可以完成控制器。

2. 多周期 CPU 逻辑设计

单周期 CPU 每条指令的执行需要一个时钟周期，而每个时钟周期的时间长短都是一样的，因此在确定时钟周期的时间长度时，要保证每条指令都已经正确完成，因此要考虑指令集中最复杂的指令执行时所需时间。上面四条指令中最复杂的指令是 lw。以 lw $3, 77($2) 为例，来说明在该周期中所做的工作。该指令的任务是从存储器中取来数据，放入寄存器堆的寄存器$3 中；存储器的地址由两个数相加得到，其中的一个数是寄存器$2 中的内容，另一个是指令中的立即数 77。执行该指令需要 5 个步骤：

(1)使用 PC 作为存储器地址，从存储器中把指令取来。

(2)从寄存器堆中寄存器$2 读出数据；把立即数 77 符号扩展为 32 位。

(3)把上述两个数相加。

(4)使用相加的结果作为地址，从存储器中取来数据。

(5)把取来的数据写入寄存器$3 中。

假设以上的每个步骤需要 1ns，则共需 5ns。由此可以确定 CPU 的时钟（clock）最高频率为 $F = \frac{1}{T} = \frac{1}{5\times10^{-9}} = 200MHz$

有些指令，比如 j 指令比 lw 简单得多，执行该指令需要 3 个步骤：

(1)使用 PC 作为存储器地址，从存储器中把指令取来。

(2)把立即数(跳转地址)扩展为 32 位。

(3)更新 PC。

因此 J 指令并不需要 5ns。但是在单周期 CPU 中，简单的 J 指令也需要执行一个时钟周期。因此，所有指令采用同一周期效率是比较低的。

为提高指令执行的效率，可以考虑根据不同的指令，给出不同的执行时间，这就要设计多周期 CPU 控制器了。在单周期 CPU 设计中，每一条指令的执行过程分成了 5 个执行步骤，分别是取指令（IF）、指令译码（ID）、指令执行（EXE）、存储器访问（MEM）和结果写回（WB）。在多周期 CPU 设计中，沿用这 5 个执行步骤。当然，在多周期 CPU 中，这 5 个执行步骤将在 5 个不同的时钟周期中执行。每条指令需要经历的执行步骤不尽相同，这样，指令使用的时钟周期也就各不相同。

在多周期 CPU 中，要解决最大的问题是设计控制部件 CU。在单周期 CPU 设计中，由于每条指令都是在一个周期内运行完成的，每条指令在解码之后，可以使用组合逻辑给出各种控制信号。这些控制信号在指令周期内部不需要改变，直至下一个周期到来，开始执行下

一条指令。然而，在多周期 CPU 中，指令不是在一个周期内运行完成的，这样，就需要控制部件 CU 能够知道在每一个周期要做什么事情。不同的指令，指令执行中需要的周期数量是不同的，每个周期中做的事情也是不一样的。

要做到根据指令的不同，在各个周期内完成不同的任务，这就需要使用有限状态机(Finite State Machine)来实现。一个有限状态机是由一组状态及状态间的装换规则组成。转换规则由一个后继状态函数确定，它将现有状态和输入映射到一个新的状态。当用一个有限状态机描述控制时，每个状态都对应一组控制信号输出。在本章的多周期 CPU 设计采用有限状态机的方式来实现控制部件 CU 的设计。根据不同的指令，控制部件 CU 给出不同的控制时序。在每一个周期内，控制部件 CU 需要给出相应的控制信号。

下面通过一个例子来讲述多周期 CPU 硬布线控制器的设计。这个 CPU 可以处理 5 条指令，其中 4 条指令和上小节讲的一样，第 5 条指令为

5.BEQ rs,rt,offset

000100	rs(5 位)	rt(5 位)	Offset (16 位)

16 位 offset 经符号扩展到 32 位，与 PC+1 相加，结果作为新地址。如果 rs=rt，则 PC 置为新地址，完成跳转。如果 rs≠rt，则顺序执行。

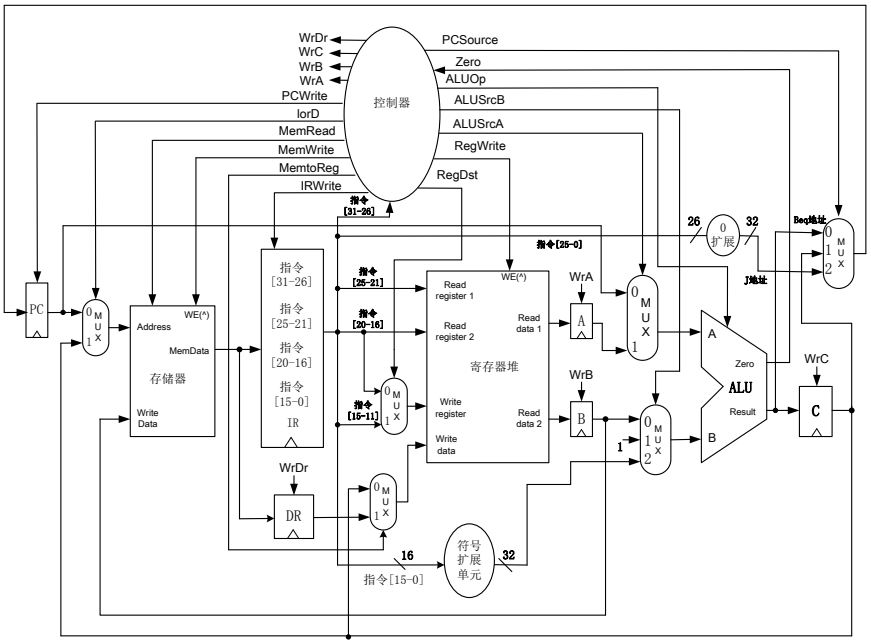

图 6. 多周期 CPU 数据通路和控制线路图

图 6 是一个非常简单的能够多周期完成上述 5 条指令的数据通路和必要的控制线路图。与单周期 CPU 相比，这里只用了一个存储器模块，指令和数据均存于其中。PC+1 和转移地址的计算由 ALU 完成，而不像单周期 CPU 那样使用专用加法器。另外，由于一条指令需要在多个周期内完成，为保存同一条指令的不同周期间的数据，所以图 6 比图 1 多了些寄存器，如 A、B、C、DR 等。存储器和寄存器堆，读操作时，给出地址，输出端就直接输出相应数据；而在写操作时，在 WE 使能信号为 1 时，在时钟上升沿写入。

表 6.1 分析了 5 条指令的周期分配情况，√表示需要执行的步骤。

表 6.1　指令周期分配表

指令	IF	ID	EXE	MEM	WB
add	√	√	√		√
lw	√	√	√	√	√
sw	√	√	√	√	
beq	√	√	√		
j	√	√	√		

从表中可以看出，这 5 条指令的执行至少需要 5 种状态，而且不同指令的同一状态有可能操作也不一样，如 add 和 j 指令的 exe 状态，其操作肯定不一样。状态的转移有些是无条件的，如 IF 状态转到 ID 状态。但有些转移要根据条件来确定的，如 ID 状态转到 add 的 exe 状态还是到 j 的 exe 状态，这是不同的。这要根据指令码和当前状态来确定。

多周期的 CPU 有限状态机需要的状态如表 6.2 所示。

表 6.2　不同指令在每个周期要完成的动作

步骤	ADD	LW	SW	BEQ	J
取指令（IF）	IR = Memory[PC] PC = PC + 1				
取寄存器操作数及计算转移地址（ID）	A = Reg[rs] B = Reg[rt] C = PC +符号扩展的指令[15..0]				
执行（EXE）	C = A add B	C = A +符号扩展的指令[15..0]		若(A-B) = =0 则 PC = C	PC = 0 扩展的指令[25..0]
存储器访问(MEM)或R指令写回(WB）	Reg[rd] = C	DR = Memory[C]	Memory[C] = B		
存储器数据写回（WB）		Reg[rt] = DR			

根据表 6.2 和图 6 数据通路，可以画出图 7 的有限状态机图。

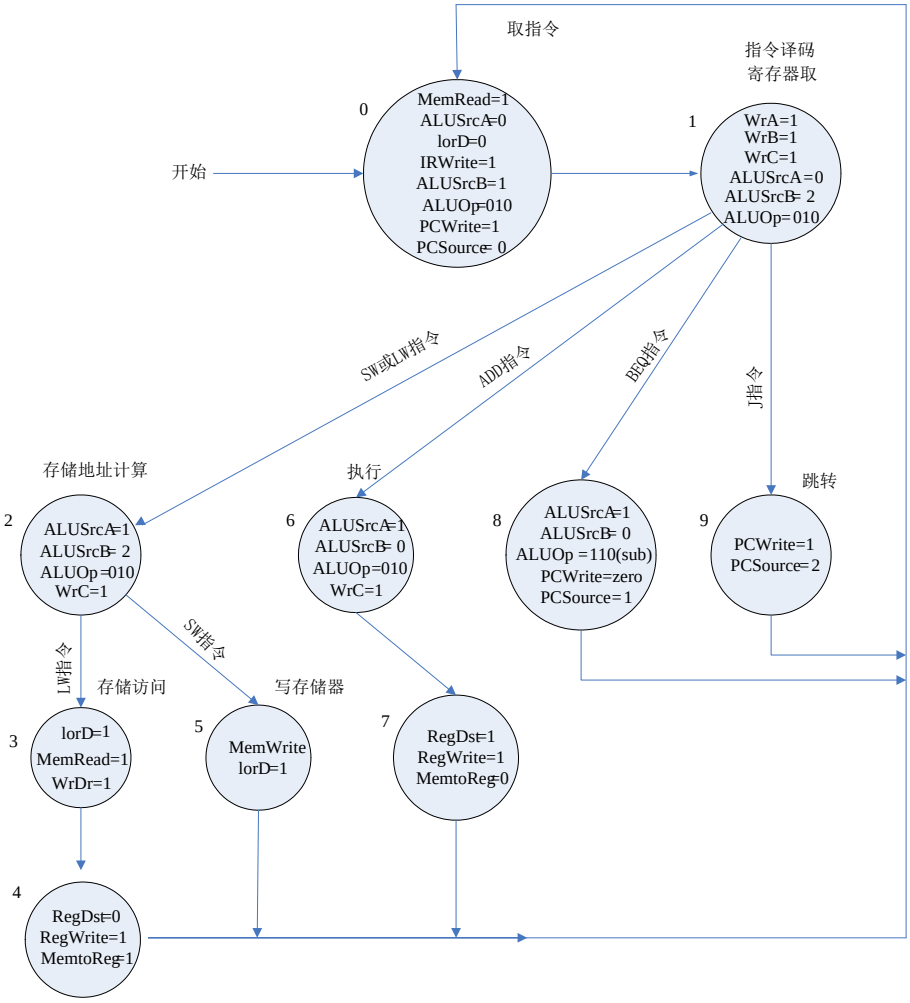

图 7. 完整的有限状态机

下面分析图 7 中 10 个不同状态的数据通路和工作过程，粗线为当前有效通路，控制器上斜体为当前有效控制信号。

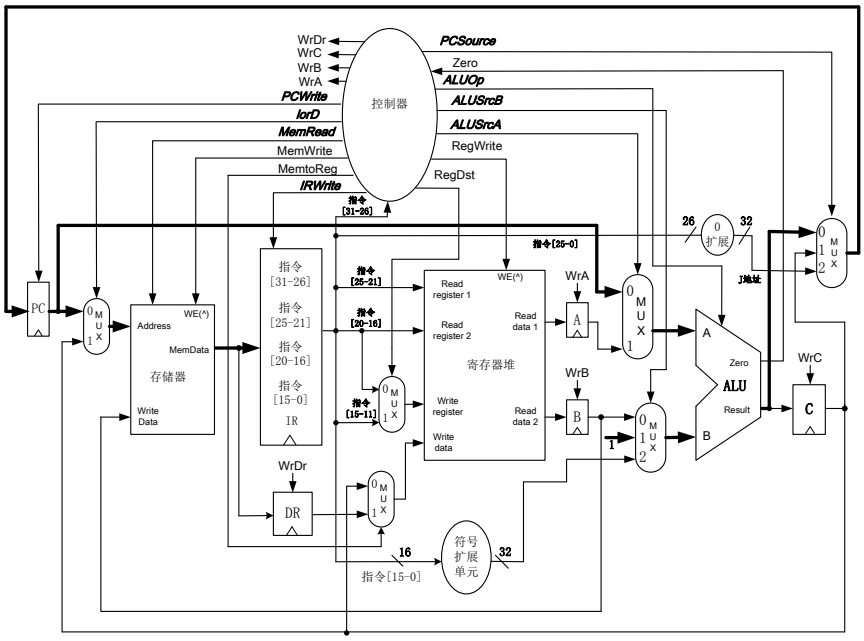

图 8. 取指状态(状态 0)

从图 8 可以看到，在该周期中，以 PC 为地址从存储器取出指令到指令寄存器 IR 的输入端，PC 与 1 相加结果返回 PC 输入端。当下一个时钟上升沿到来时，IR 和 PC 均做更新。

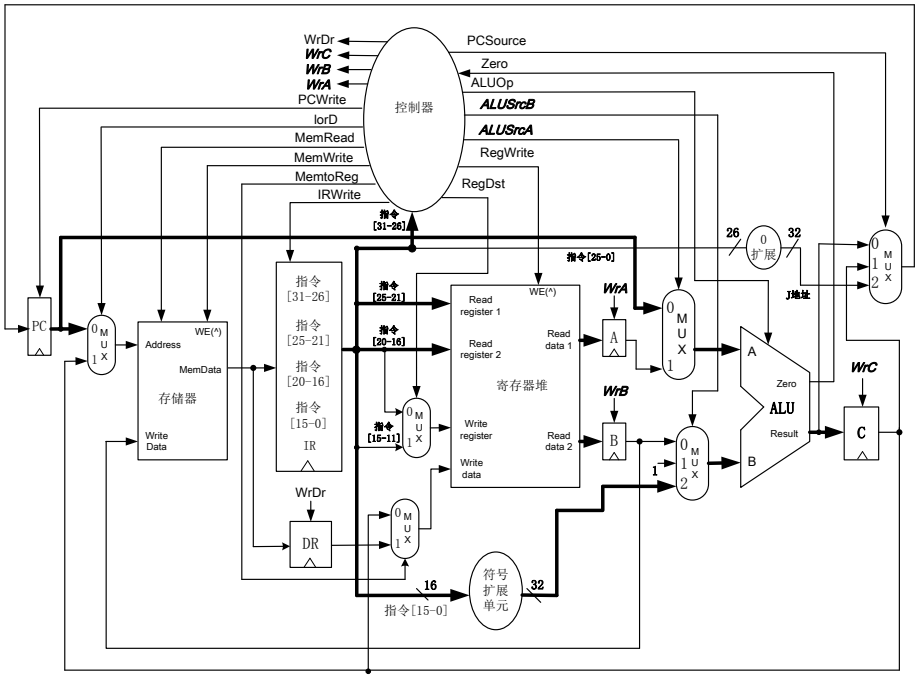

图 9. 译码状态(状态 1)

指令的[31..26]进入控制器进行译码。同时，指令[25..21]、[20..16]作为地址读出两寄存器数到寄存器 A 和 B 的输入端。PC 加上指令的[15..0]，结果到寄存器 C 的输入端，为 BEQ 指令准备。

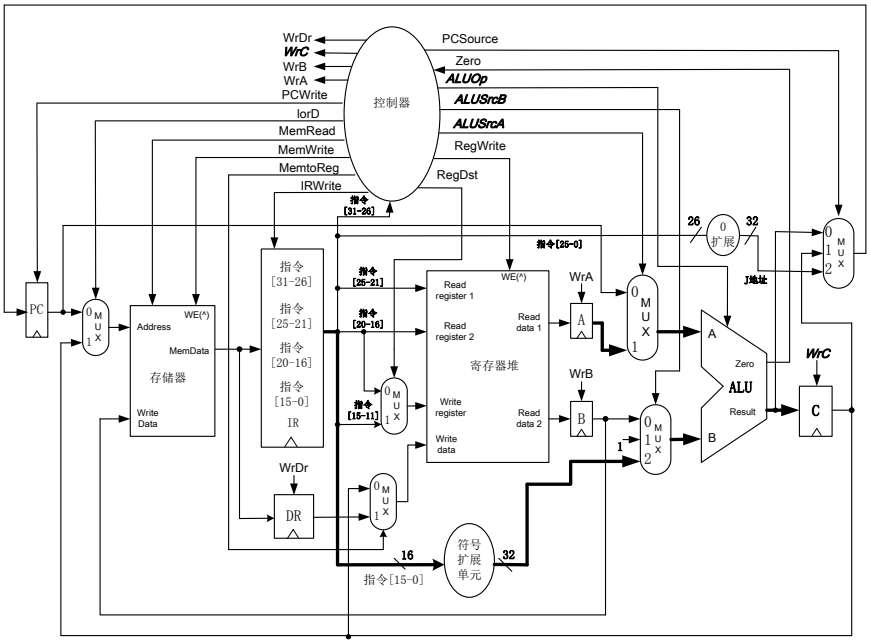

图 10. 存储地址计算(状态 2)

状态 2 完成 LW 和 SW 指令中存储地址的计算。基地址在寄存器 A 中，偏移量在指令[15..0]，相加结果作为地址到寄存器 C 的输入端。

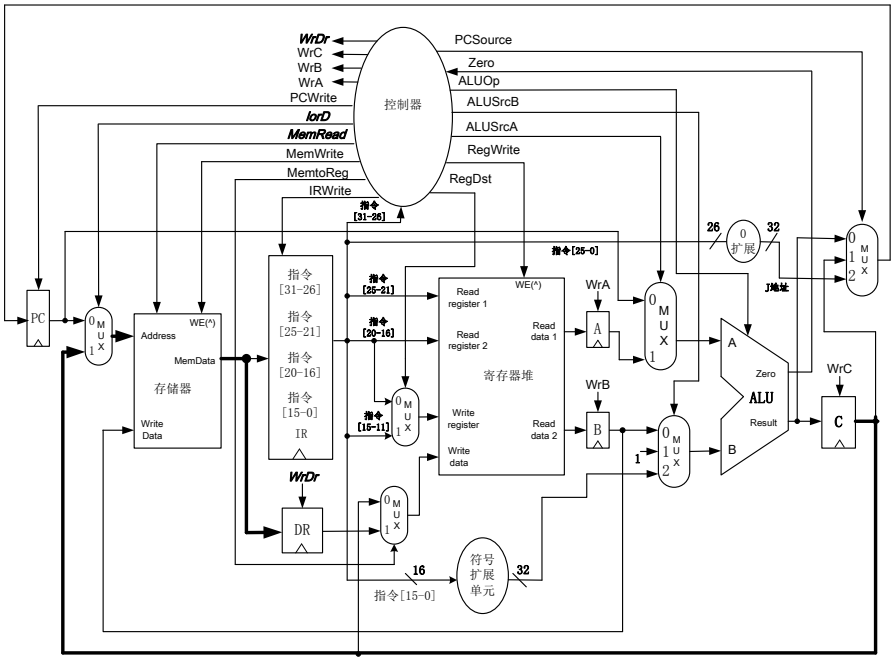

图 11. LW 指令读存储器到 DR(状态 3)

状态 3 完成读存储器到寄存器 DR。寄存器 C 为存储器地址，存储器的数据到寄存器 DR 输入端。

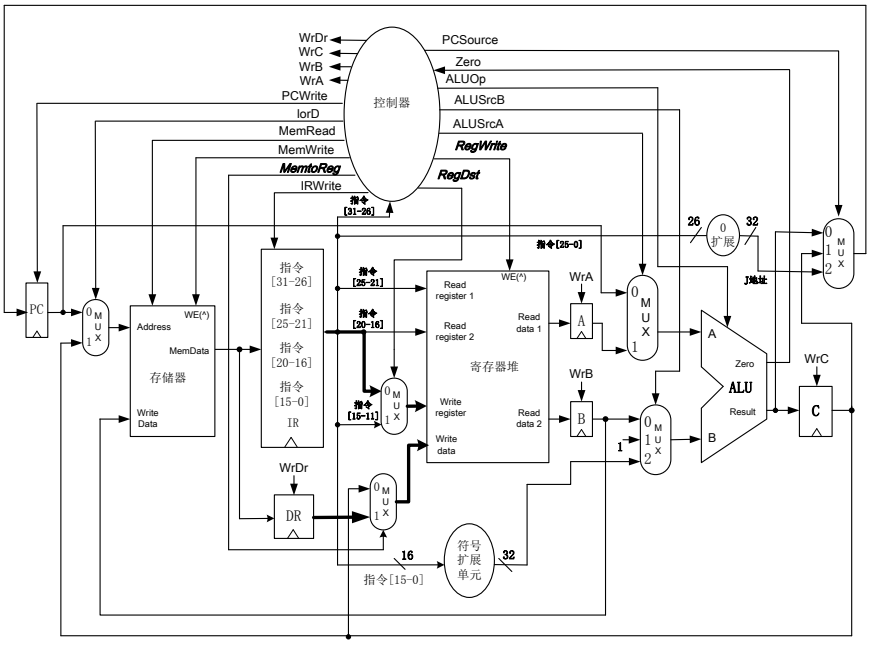

图 12. LW 指令从 DR 到寄存器堆(状态 4)

状态 4 时从存储器读出的数据已经在 DR，现在将此数据到寄存器堆输入端，将指令[20..16]作为寄存器堆的地址，在下一个时钟上升沿可以写入。

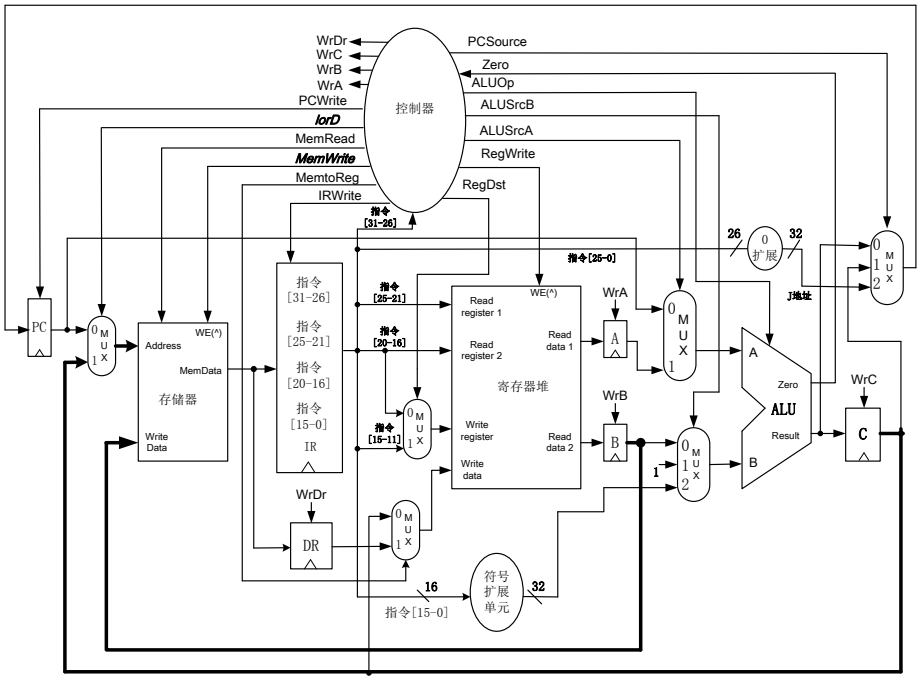

图 13. SW 指令写存储器(状态 5)

状态 5 时，存储器的地址在寄存器 C 中，数据在寄存器 B 中，在下一个时钟周期上升沿写入。

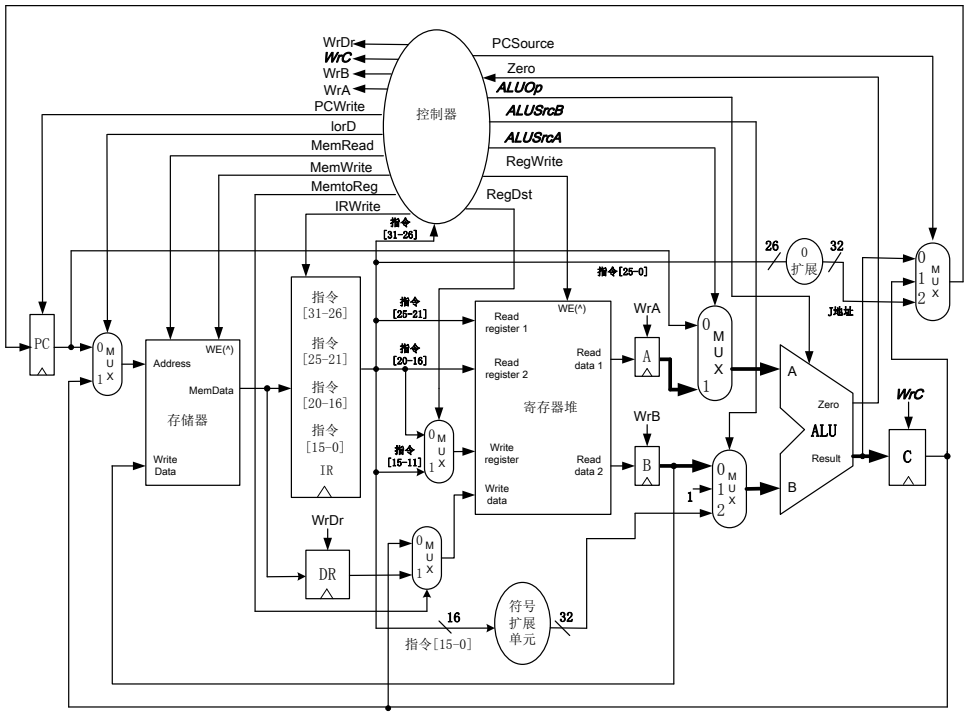

图 13. ADD 指令加法(状态 6)

状态 1 取出的两加数在寄存器 A 和 B 中，在状态 6 执行加法，结果到寄存器 C 的输入端。

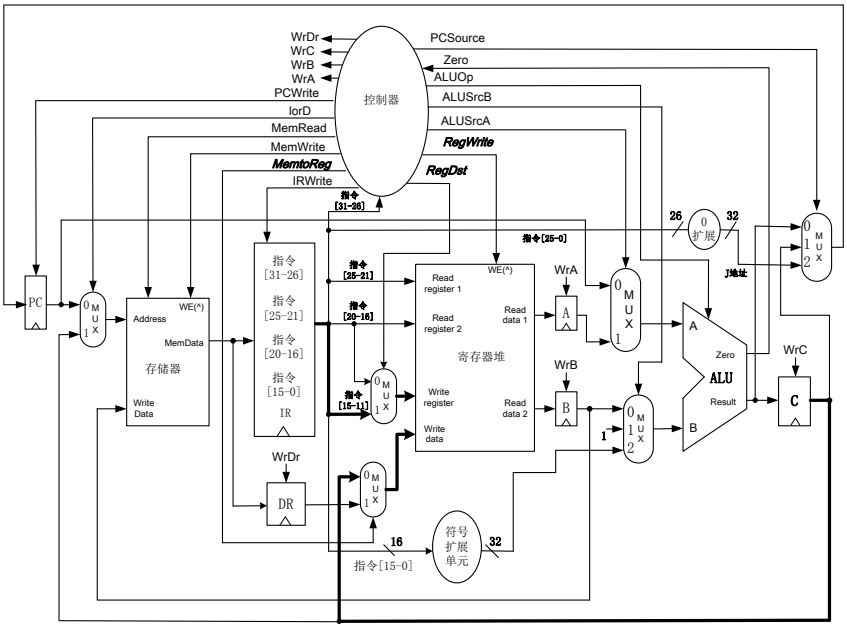

图 14. ADD 指令写回寄存器(状态 7)

状态 7 时，数据在寄存器 C 中，要写入的地址在指令[15..11]中，在下一个时钟上升沿写入。

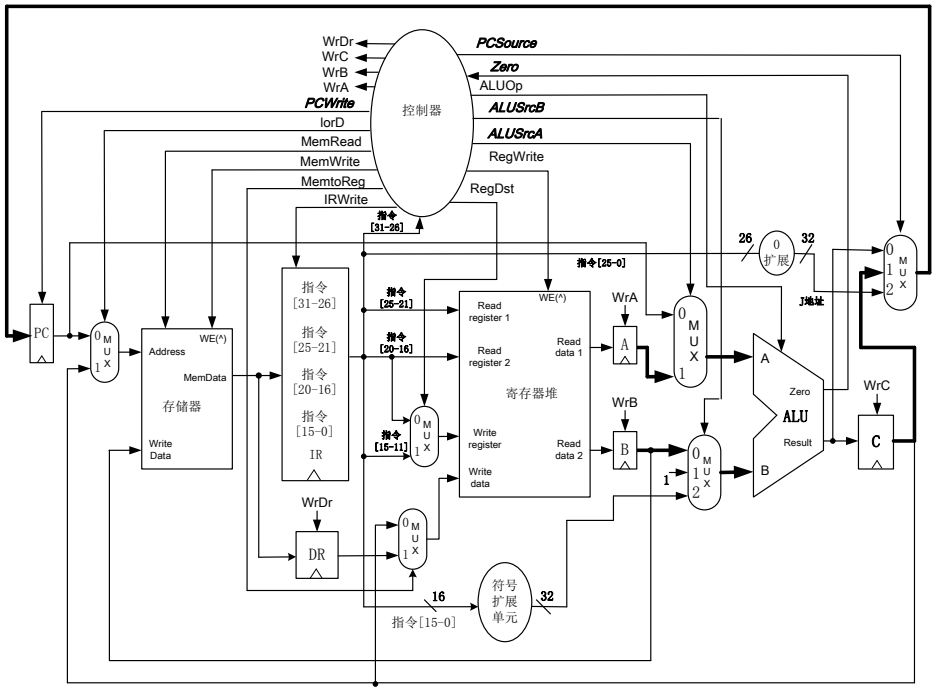

图 15. BEQ 指令判断是否改写 PC(状态 8)

要改写的地址在上一个周期已经算出，存放在寄存器 C 中，并送到 PC 输入端。在状态 8 计算寄存器 A 和 B 中两数差，如为 0，则 PCWrite=1，使 PC 能在下一周期改掉；如为 1，则 PCWrite=0，使 PC 能够保持。

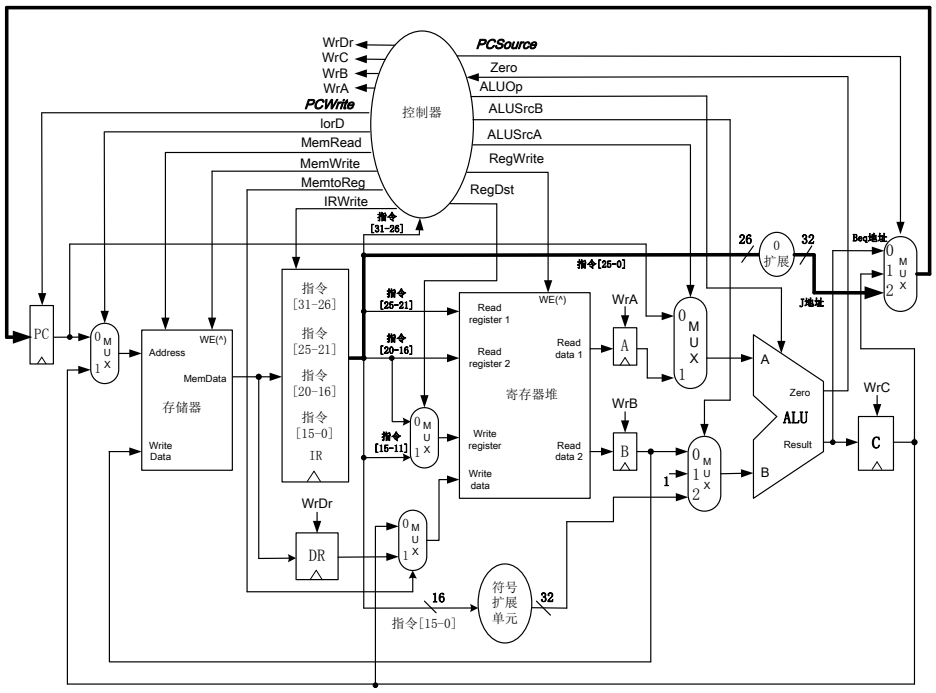

图 16. J 指令改写 PC(状态 9)

状态 9 时，更改的地址为指令[25..0]，在下一时钟上升沿更改。

下面来分析控制器产生电路。根据图 7，显然需要设置一寄存器保存当前状态，根据当前状态和指令码确定下一状态。控制器框图如图 17 所示，其中 Zero 为 ALU 执行减法时的结果输出，以确定 BEQ 指令的执行。

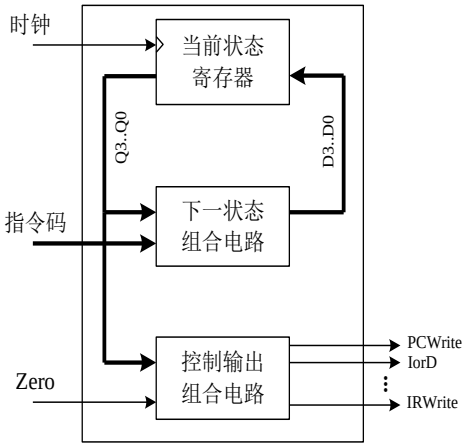

图 17.控制器框图

根据图 7 和图 17，可以得到下一状态转换表。

当前状态	Q3Q2Q1Q0	指令码	下一状态	D3D2D1D0
0	0000	×	1	0001
1	0001	SW 或 LW	2	0010

1	0001	ADD	6	0110
1	0001	BEQ	8	1000
1	0001	J	9	1001
2	0010	LW	3	0011
2	0010	SW	5	0101
3	0011	×	4	0100
4	0100	×	0	0000
5	0101	×	0	0000
6	0110	×	7	0111
7	0111	×	0	0000
8	1000	×	0	0000
9	1001	×	0	0000

对应的可以写出控制器框图中下一状态组合电路组合逻辑，如

D3=状态 1 • BEQ+状态 1 • J

D2=状态 1 • ADD+状态 2 • SW+状态 3 +状态 6

等等

而状态 1= $\overline{Q3}\,\overline{Q2}\,\overline{Q1}Q0$，状态 2= $\overline{Q3}\,\overline{Q2}Q1\overline{Q0}$，BEQ= $\overline{Op_5}\cdot\overline{Op_4}\cdot\overline{Op_3}\cdot Op_2\cdot\overline{Op_1}\cdot\overline{Op_0}$ (由指令码得)，J= $\overline{Op_5}\cdot\overline{Op_4}\cdot\overline{Op_3}\cdot\overline{Op_2}\cdot Op_1\cdot\overline{Op_0}$ 等等。代入可得 D3D2D1D0 组合逻辑。

下面写出图 17 中的控制输出组合逻辑电路的组合逻辑，从图 7，可以得下表

输出	状态 0	状态 1	状态 2	状态 3	状态 4	状态 5	状态 6	状态 7	状态 8	状态 9
PCWrite	1	0	0	0	0	0	0	0	Zero	1
WrC	0	1	1	0	0	0	1	0	0	0
…										

对应的

PCWrite= $\overline{Q3}\,\overline{Q2}\,\overline{Q1}\,\overline{Q0}$ +Zero+ $Q3\overline{Q2}\,\overline{Q1}Q0$

WrC= $\overline{Q3}\,\overline{Q2}\,\overline{Q1}Q0 + \overline{Q3}\,\overline{Q2}Q1\overline{Q0} + Q3\overline{Q2}\,\overline{Q1}Q0$

类似的，可以得到所有控制信号。