第7章 控制器 作业参考题解

7.1 名词术语解释

PC	IR	AR	指令译码
指令预取	指令周期	机器周期	时钟周期
微操作	微命令	微指令	微地址
微程序入口	控制存储器	字段直接编译法	直接控制法
字段间接控制法	水平型微指令	垂直型微指令	毫微程序
硬布线控制器	微程序控制器	流水线	

参考答案：

1. PC：程序计数器，用于存放指令地址，位于 CPU 内。
2. IR：指令寄存器，用于存放指令代码，位于 CPU 内。
3. AR：地址寄存器，用于存放访问存储器的地址。
4. 指令译码：从主存中取出指令码后，按照指令格式对其译码，以确定指令的功能、寻址方式、操作数等等。
5. 指令预取：为实现并行执行指令的流水线，在执行本条指令时，就同时预先从主存中取出下一条指令的代码，减少等待访存的时间，这种操作叫指令预取。
6. 指令周期：是指计算机从主存取出一条指令并完成该指令的执行所需要的时间。
7. 机器周期：又称为 CPU 周期，用于完成相对完整的一个操作，一般为 1 次内存的操作（读或写访问）或者 1 次 ALU 的运算，或者 1 次总线传送。
8. 时钟周期：又称为节拍，是指 CPU 执行一个微操作命令（即控制信号）的最小时间单位，也即 T 周期。
9. 微操作：指令执行时必须完成的基本操作。
10. 微命令：是组成微指令的最小单位，也就是控制实现微操作的控制信号。一般用于控制数据通路上门的打开/关闭，或者功能选择。
11. 微指令：是一组微命令的集合，用于完成一个功能相对完整的操作。
12. 微地址：微指令存放在控制存储器中的地址。
13. 微程序入口：机器指令的功能是由一段微程序解释执行而完成的，由机器指令的操作码来找到该指令对应的微程序在控制存储器中的首地址，即微程序入口。
14. 控制存储器：简称控存，用于存放所有指令的微程序，其中一个存储单元存放一条微指令。一般为 ROM。
15. 字段直接编译法：将微指令的控制字段分成若干段，每段通过编码/译码对应到各个控制信号，从而提高微指令的并行操作能力，缩短微指令字长。

16. 直接控制法：微指令的控制字段中，每一位代表一个微命令（控制信号），直接控制部件动作。
17. 字段间接控制法：将微指令的控制字段分段，某一字段所产生的微命令，还需要和另一字段的代码联合译码得到。
18. 水平型微指令：指能同时发送许多个微命令、同时控制数据通路中多个功能部件并行操作的微指令。特点是：微指令执行效率高、速度快、灵活，并行操作能力强，编制的微程序比较短。
19. 垂直型微指令：采用完全编码的方法，将一套微命令代码化构成微指令。因此，一条微指令只能控制 1～2 种微操作，并行操作能力弱，执行效率低。
20. 毫微程序：在两级微程序设计中，第一级是垂直微程序，用来解释机器指令；第二级是水平微程序，又称毫微程序，用来解释垂直微指令。
21. 硬布线控制器：是指采用组合逻辑电路即硬布线设计的方法构成计算机控制器的主要部件——操作控制信号形成部件。
22. 微程序控制器：是指采用存储逻辑电路即微程序设计的方法构成计算机控制器的主要部件——操作控制信号形成部件。
23. 流水线：就是是将一个较复杂的处理过程分成若干个复杂程度相当、处理时间大致相等的子过程，每个子过程由一个独立的功能部件来完成，处理对象在各子过程连成的线路上连续流动，各功能部件并行工作，从而处理速度成倍提高。

7.2　什么叫指令？什么叫微指令？它们有什么关系？指令包括哪些部分？微指令又包括哪些部分？程序由什么部件确定顺序执行？怎样实现转移？微程序中的顺序执行和转移可以用什么方法实现？

参考答案：

- 指令：即机器指令，指能被计算机硬件识别并直接执行的 0、1 代码串。
- 微指令：指一组微命令的集合，用于完成功能相对完整的一个操作。
- 指令与微指令关系：一条机器指令是由一段微指令构成的微程序解释执行的。
- 指令包括：操作码和地址码
- 微指令包括：控制字段和下址字段
- 程序由 PC 部件自增 1 来确定顺序执行，通过修改 PC 的值来实现转移。
- 微程序中的顺序执行和转移可以用使用一个 μPC 来实现，也可以由下址字段来实现，或者由判别测试字段与下址字段联合控制。

7.3　控制器的基本功能是什么？它由哪些基本部件组成？

参考答案：

- 控制器的基本功能：从存储器中取指令、对指令译码、产生控制信号并控制计算机

系统各部件有序地执行，从而实现这条指令的功能。（取指令、分析指令、执行指令）

- 控制器的组成：专用寄存器（PC、IR、AR、DR）、指令译码器、操作控制信号形成部件、时序系统

7.4　控制器有哪几种控制方式？它们各由什么特点？

参考答案：

控制器有 3 三种控制方式：

- 同步控制方式：控制器产生统一的、顺序固定的、周而复始的节拍电位（机器周期信号）和节拍脉冲（时钟周期信号），即采用相同的机器周期数和相同的节拍脉冲来形成每条指令的操作控制信号序列，因此每条指令的执行所用的时间都是相同的。同步控制方式的优点是电路简单，缺点是运行速度慢、效率低。
- 异步控制方式：每条指令需要多少节拍，就产生多少节拍；当指令执行完毕，发出回答信号；控制器收到回答信号时，才开始下条指令的执行。因此，执行不同指令所需的时间完全由实际需要确定，不尽相同。异步控制方式的优点是指令的运行效率高；缺点是控制器的电路比较复杂。
- 联合控制方式：把同步控制方式和异步控制方式结合使用，大部分指令安排在统一的机器周期内完成，即同步控制；而将较少数特殊指令，或微操作序列过长或过短，或微操作时间难以确定的，采用异步控制来完成。联合控制方式的优点是能保证一定的运行速度，其缺点是控制电路设计相对比较复杂。

7.5　指令周期、机器周期和时钟周期三者之间的关系怎样？

参考答案：

- 指令周期：是指计算机从主存取出一条指令并完成该指令的执行所需要的时间。
- 关系：一条指令的指令周期是由若干个机器周期组成的，而一个机器周期是由若干个时钟周期组成的。

7.6　设某机平均执行一条指令需要两次访问内存，平均需要三个机器周期，每个机器周期包含 4 个节拍周期。若机器主频为 25MHz，试回答：

（1）　若访问主存不需要插入等待周期，则平均执行一条指令的时间为多少？

（2）　若每次访问内存需要插入 2 个等待节拍周期，则平均执行一条指令的时间是又多少？

参考答案：

（1）　时钟周期（节拍周期）=1÷25MHz=40ns，一条指令平均需要 3×4=12 个节拍周期，则平均执行一条指令的时间 = 40ns×12 = 480ns

（2） 若每次访问内存需要插入 2 个等待节拍周期，又一条指令需要访问内存 2 次，则一条指令需要 12+2×2=16 个节拍周期，由此得：平均执行一条指令的时间=40ns×16 = 640ns

7.7 设某机主频为 8MHz，每个机器周期包含 4 个节拍周期，该机平均指令执行速度为 1MIPS。试回答：

（1） 该机的平均指令周期是多少时间？

（2） 平均每条指令周期包含几个机器周期？

参考答案：

（1） 因为平均指令执行速度为 1MIPS，即每秒钟执行 10^6 条指令，则平均指令周期为 $1 \div 10^6$ 秒=1μs

（2） 主频为 8MHz，则时钟周期（节拍周期）=1÷8MHz = 125ns，机器周期 = 4×节拍周期 = 4×125ns = 500ns，又因为指令周期=1μs，所以平均每条指令周期包含 1μs÷500ns =2 个机器周期

7.8 如图 7.27 所示的单周期 CPU 数据通路，在其上执行下列 MIPS 指令，描述执行过程，并据此总结 ALU 必须具备的运算功能，：

（1） xor rd,rs,rt； 位异或：rs⊕rt→rd

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 xor 的 OP 和 func，发送出 w_r_s=00，rt_imm_s=0，wr_data_s=00，ALU_OP 为异或的运算码，Write_Reg=1，Mem_Write=0，PC_s=00；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，读出数据送 ALU 的 A 端和 B 端（因为 rt_imm_s=0），ALU 执行异或操作（ALU_OP 指定）；**
- **在 clk 的下跳沿，运算结果写入 rd 字段指定的寄存器（因为 w_r_s=00、wr_data_s=00 和 Write_Reg=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（2） sltu rd,rs,rt； 无符号数小于则置位：if (rs < rt) rd=1 else rd=0

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 sltu 的 OP 和 func，发送出 w_r_s=00，rt_imm_s=0，wr_data_s=00，ALU_OP 为小于置位操作的运算码，Write_Reg=1，Mem_Write=0，PC_s=00；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，读出数据送 ALU**

的 A 端和 B 端（因为 rt_imm_s=0），ALU 执行小于置位操作（ALU_OP 指定）；在 clk 的下跳沿，运算结果写入 rd 字段指定的寄存器（因为 w_r_s=00、wr_data_s=00 和 Write_Reg=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。

（3） sllv rd,rt,rs； 逻辑左移：(rt << rs)→rd

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 sllv 的 OP 和 func，发送出 w_r_s=00，rt_imm_s=0，wr_data_s=00，ALU_OP 为逻辑左移操作的运算码，Write_Reg=1，Mem_Write=0，PC_s=00；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，读出数据送 ALU 的 A 端和 B 端（因为 rt_imm_s=0），ALU 执行逻辑左移操作（ALU_OP 指定）；**
- **在 clk 的下跳沿，运算结果写入 rd 字段指定的寄存器（因为 w_r_s=00、wr_data_s=00 和 Write_Reg=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（4） andi rt, rs, imm； 逻辑与：rs & imm→rt

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 andi 的 OP 字段，发送出 w_r_s=01，imm_s=0，rt_imm_s=1，wr_data_s=00，ALU_OP 为逻辑与操作的运算码，Write_Reg=1，Mem_Write=0，PC_s=00；指令码的 rs 字段送寄存器堆的 A 口地址，rt 字段送寄存器堆的写端口地址（因为 w_r_s=01），A 口读出数据送 ALU 的 A 端，操作码的低 16 位经过无符号扩展（因为 imm_s=0）后的 32 位数据送 ALU 的 B 端（因为 rt_imm_s=1），ALU 执行逻辑与操作（ALU_OP 指定）；**
- **在 clk 的下跳沿，运算结果写入 rt 字段指定的寄存器（因为 w_r_s=01、wr_data_s=00 和 Write_Reg=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（5） sw rt, offset(rs)； 存数：rt →（rs +offset）

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 sw 的 OP 字段，发送出 imm_s=1，rt_imm_s=1，ALU_OP 为算术加操作的运算码，Write_Reg=0，Mem_Write=1，PC_s=00；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，A 口读出数据送 ALU 的 A 端，操作码的低 16 位经过符号扩展（因为 imm_s=1）后的 32 位数据送 ALU 的 B 端（因为 rt_imm_s=1），ALU 执行逻辑加操作（ALU_OP 指定）计算的地址直接送数据存储器的地址端；**
- **在 clk 的下跳沿，寄存器堆的 B 口数据（即 rt 寄存器数据）写入数据存储器指定的单元（因为 Mem_Write=1）；并且 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

（6） bne rs, rt, label； 不相等转移：if(rs≠rt) PC+4+offset*4→PC

- **在时钟周期上跳沿，将 PC 内容作为指令存储器地址，执行读操作，读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；**
- **“译码及控制单元”依据 bne 的 OP 字段，发送出 imm_s=1，rt_imm_s=0，ALU_OP 为算术减操作的运算码，Write_Reg=0，Mem_Write=0，PC_s=10（ZF=0）或者 PC_s=00（ZF=1）；指令码的 rs 字段和 rt 字段送寄存器堆的 A 口和 B 口地址，读出数据送 ALU 的 A 端和 B 端，进行算术减法操作；同时，操作码的低 16 位经过符号扩展（因为 imm_s=1）后的 32 位数据送相对地址加法器，进行左移 2 位操作并与 PC+4 值进行相加，结果备用；**
- **在 clk 的下跳沿，根据 ALU 减法运算的结果修改 PC：ZF=0 则将相对地址加法器结果置入 PC（因为 PC_s=10），ZF=1 则将 PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。**

7.9’在图 7.27 所示的单周期 CPU 数据通路上，假设 ALU 的功能及 ALU_OP 编码如下：

ALU_OP	操作
000	算术加
001	算术减
010	位异或
011	逻辑左移
100	小于置位

在其上为实现下列 MIPS 指令，写出译码与控制单元所需设置的控制信号，填入下表：

（1） xor rd,rs,rt； 位异或：rs⊕rt→rd

（2） sltu rd,rs,rt； 无符号数小于则置位：if (rs < rt) rd=1 else rd=0

（3） sllv rd,rt,rs； 逻辑左移：(rt << rs)→rd

（4） xori rt, rs, imm； 逻辑与：rs & imm→rt

（5） sw rt, offset(rs)； 存数：rt →（rs +offset）

（6） bne rs, rt, label； 不相等转移：if(rs≠rt) PC+4+offset*4→PC

（7） jal label； 无条件跳转并链接：(PC+4)→$31，{(PC+4) 高 4 位,address,0,0}→PC

指令	w_r_s	imm_s	rt_imm_s	wr_data_s	ALU_OP	Write_Reg	Mem_Write	PC_s
xor rd,rs,rt	00	—	0	00	010	1	0	00
sltu rd,rs,rt	00	—	0	00	100	1	0	00
sllv rd,rt,rs	00	—	0	00	011	1	0	00
xori rt, rs, imm	01	0	1	00	010	1	0	00
sw rt, offset(rs)	—	1	1	—	000	0	1	00

指令	w_r_s	imm_s	rt_imm_s	wr_data_s	ALU_OP	Write_Reg	Mem_Write	PC_s
bne rs, rt, label	—	1	0	—	001	0	0	ZF=0:10 ZF=1:00
jal label	10	—	—	10	—	1	0	11

7.9 伪指令是指机器指令系统中不存在，但是汇编器能将其翻译为某一条或者多条机器指令来实现的符号指令。譬如，移动指令 move rd,rs 是将 rs 内容传送给 rd 寄存器的伪指令，汇编器可以将它翻译为 add rd,rs,$0。MIPS 中有一条取立即数伪指令 li rd,imm，功能是将 16 位立即数 imm 存入 rd 寄存器（高 16 位清零）。

（1） 结合 MIPS 的核心指令集，思考：这条伪指令会被 编译器翻译成哪一条或哪几条指令？

可以被编译器翻译成以下任何一条指令：

ori rd, $zero, imm

xori rd, $zero, imm

（2） 假设现在要直接在 MIPS CPU（单周期）指令系统中加入这条指令（不是伪指令），那么能否在图 7-27 所示的单周期 CPU 数据通路上实现？如果不能，那么需要添加哪些其他部件和数据通路才能实现？请画出并说明执行过程。

可以实现，两种方法：

一种方法可以直接在 li 指令编码时，规定其 rs 字段一定为 00000，即指定 rs 字段为$0，这种方法无须添加部件；另一种方法是：在寄存器堆的 A 地址端口前添加一个 5 位的二选一电路，由 rA_s 信号控制，rA_s=0，则选择指令码的 rs 字段送寄存器的 A 口地址，rA_s =1，则选择 00000 送寄存器的 A 口地址；在执行 li 指令时，发送 rA_s =1，其他指令时，发送 rA_s =0。

7.10 假设要实现一条 lw_inc rt,offset(rs)指令，其功能是在普通的取数指令 lw 的功能基础上，增加递增功能，即取数后，递增地址索引寄存器 rs 的值。lw_inc 相当于执行了两条指令：

lw rt,offset(rs)

addi rs,rs,1

在图 7-27 所示的单周期 CPU 数据通路上添加一些部件和控制信号，实现该指令，说明你的设计方案和指令执行过程。

设计方案：可以强化寄存器堆的功能，使其能对 A 口地址指明的寄存器执行+1 功能，这要求寄存器堆中每一个寄存器都有+1 计数功能，寄存器堆添加一个控制信号 rA_1，当该信号=1 时，在 clk 的下跳沿，将 A 口地址选中的寄存器内容+1。

执行过程：在时钟周期上跳沿，根据 PC 从指令存储器读出 32 位指令机器码，OP 和 func 字段送“译码及控制单元”；同时 PC 送自增加法器，执行+4 操作；“译码及控制单元”依据 lw+的 OP 字段，发送出 w_r_s=01，imm_s=1，rt_imm_s=1，wr_data_s=01，ALU_OP 为算术加操作的运算码，Write_Reg=1，Mem_Write=0，PC_s=00，rA_1=1；指令码的 rs 字段送寄存器堆的 A 口地址，rt 字段送寄存器堆的写端口地址（因为 w_r_s=01），A 口读出数据送 ALU 的 A 端，操作码的低 16 位经过符号扩展（因为 imm_s=1）后的 32 位数据送 ALU 的 B 端（因为 rt_imm_s=1），ALU 执行逻辑加操作（ALU_OP 指定），计算的地址直接送数据存储器的地址端；在 clk 的下跳沿，数据存储器读出的数据写入 rt 指定的寄存器（因为 w_r_s=01，wr_data_s=01 和 Mem_Write=0），且 rs 指定的寄存器执行+1 操作（因为 rA_1=1），PC 的自增值 PC+4 置入 PC（因为 PC_s=00）。

7.11 在图 7-27 所示的单周期 CPU 数据通路上，假设多路复用器的下列选择控制信号发生了恒零错误（即始终为 0），考虑：哪些指令可以正常工作，哪些指令不可以正常工作？

（1） w_r_s=00b;
可以正常工作的指令：R 型、I 型分支类指令、j 指令、sw 指令；
不能正常工作的指令：I 型运算类指令、lw 指令、jal 指令，因为无法写 rt 或者 $31；

（2） imm_s=0b;
可以正常工作的指令：R 型、J 型；I 型无符号运算类指令及逻辑运算类指令
不能正常工作的指令：I 型有符号运算类以及分支指令、sw 和 lw 指令，因为无法符号扩展；

（3） wr_data_s=00b;
可以正常工作的指令：R 型、I 型分支类指令、I 型运算类指令、j 指令、sw 指令；
不能正常工作的指令： lw 指令、jal 指令，因为无法将存储器读出数据或者 PC+4 的值写入寄存器堆；

（4） rt_imm_s=0b;
可以正常工作的指令：R 型、J 型指令、I 型分支指令；
不能正常工作的指令： I 型运算类指令、I 型访存类指令，因为无法将立即数字段送到 ALU 的 B 端；

（5） PC_s=00b;
可以正常工作的指令：R 型、I 型运算类指令、I 型访存指令；
不能正常工作的指令：转移类指令，包括 jr、j、jal、I 型分支类指令，因为无法更新 PC 为其他的值（除 PC+4 外）；

7.12 在图 7.27 所示的单周期 CPU 数据通路上，考虑多路复用器的选择控制信号发生了恒 1 错误对 CPU 的影响。考虑：哪些指令可以正常工作，哪些指令不可以正常工作？

（1） w_r_s=11b;
可以正常工作的指令：I 型分支类指令、jr 指令、sw 指令；
不能正常工作的指令：R 型、I 型运算类指令、lw 指令、jal 指令，因为无法写 rd、rt 或者$31；

（2） imm_s=1b;
可以正常工作的指令：R 型、J 型；I 型有符号运算类指令、以及分支指令、sw 和 lw 指令；
不能正常工作的指令：I 型无符号运算类或者逻辑运算类，因为无法进行无符号扩展；

（3） wr_data_s=11b;
可以正常工作的指令：I 型分支类指令、jr 指令、sw 指令；
不能正常工作的指令：R 型、I 型运算类指令、lw 指令、jal 指令，因为无法将 ALU 运算结果、或者存储器读出数据或者 PC+4 的值写入寄存器堆；

（4） rt_imm_s=1b;
可以正常工作的指令：I 型运算类指令、I 型访存指令、jr、j、jal 指令；
不能正常工作的指令：R 型、I 型分支类指令，因为无法将 rt 的内容送到 ALU 的 B 端；

（5） PC_s=11b;
可以正常工作的指令：j、jal 指令；
不能正常工作的指令：其他指令，因为无法更新 PC 为其他的值（除页面寻址的转移地址之外）；

7.13 在图 7.42 所示的多周期 CPU 的数据通路上，描述下列指令的执行步骤及每个机器周期中需要发送的操作控制信号序列。假设 ALU 的功能及 ALU_OP 编码如下：

ALU_OP	操作
100	位与
101	位或
000	位异或
001	位非
010	算术加
011	算术减
110	小于置位
111	逻辑左移

（1） xor rd,rs,rt；位异或：rs⊕rt→rd

（2） sltu rd,rs,rt；无符号数小于则置位：if (rs < rt) rd=1 else rd=0

（3） sllv rd,rt,rs；逻辑左移：(rt << rs)→rd

（4） andi rt, rs, imm；逻辑与：rs & imm→rt

（5） bne rs, rt, label；不相等转移：if(rs≠rt) PC+4+offset*4→PC

时钟周期	操作	发送控制信号
（1）xor rd,rs,rt		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=010; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B	无
M2	A （op）B→F	ALU_A_s=1, ALU_B_s=01, ALU_OP=000;
M3	F→ Reg[rd]	rd_rt_s=0,alu_mem_s=0, Reg_write;
（2）sltu rd,rs,rt		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=010; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B	无
M2	A （op）B→F	ALU_A_s=1, ALU_B_s=01, ALU_OP=110;
M3	F→ Reg[rd]	rd_rt_s=0,alu_mem_s=0, Reg_write;
（3）sllv rd,rt,rs		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=010; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B	无
M2	A （op）B→F	ALU_A_s=1, ALU_B_s=01, ALU_OP=111;
M3	F→ Reg[rd]	rd_rt_s=0,alu_mem_s=0, Reg_write;
（4）andi rt, rs, imm		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=010; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B	无
M2	A （op）imm→F	ALU_A_s=1, ALU_B_s=10, imm_s=0,ALU_OP=100;
M3	F→ Reg[rt]	rd_rt_s=1,alu_mem_s=0, Reg_write;

时钟周期	操作	发送控制信号
（5）I 型分支指令：beq		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0, Mem_read, IR_write; ALU_OP=010; ALU_A_s=0,ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A,Reg[rt]→B PC+offset*4→F	ALU_A_s=0,ALU_B_s=11,ALU_OP=010,imm_s=1;
M2	A - B ，产生 zero zero=1，则 F→PC zero=0，空操作	ALU_A_s=1, ALU_B_s=01, ALU_OP=011; zero=1:PC_s=01,PC_write

7.14 综合上题中 5 条指令的操作控制信号的逻辑函数，画出其硬布线控制器的电路图。

（1）综合微操作控制信号的函数：

I_D_s = 0

Mem_read = M0

Mem_write = 0

IR_write = M0

PC_write = M0+ beq • M2 • zero

Reg_write = （~beq）• M3

imm_s= beq • M1

ALU_A_s = M2

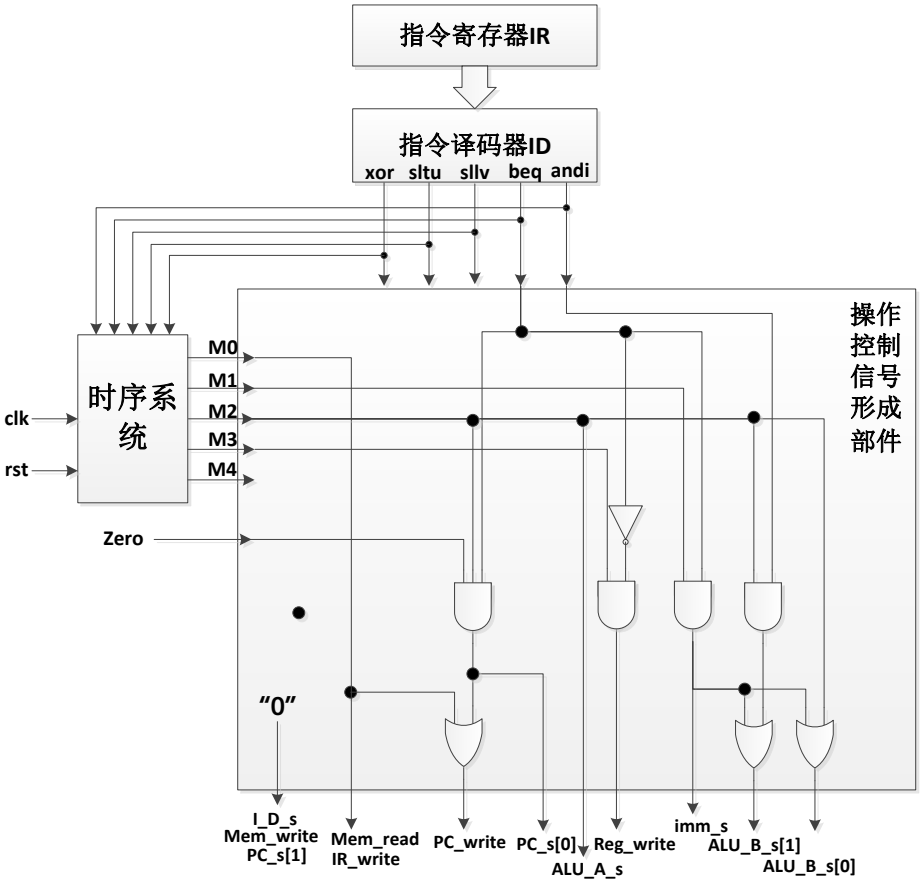
ALU_B_s[1] = andi • M2 + beq • M1

ALU_B_s[0] = M2 + beq • M1

PC_s[1] = 0

PC_s[0] = beq • M2 • zero

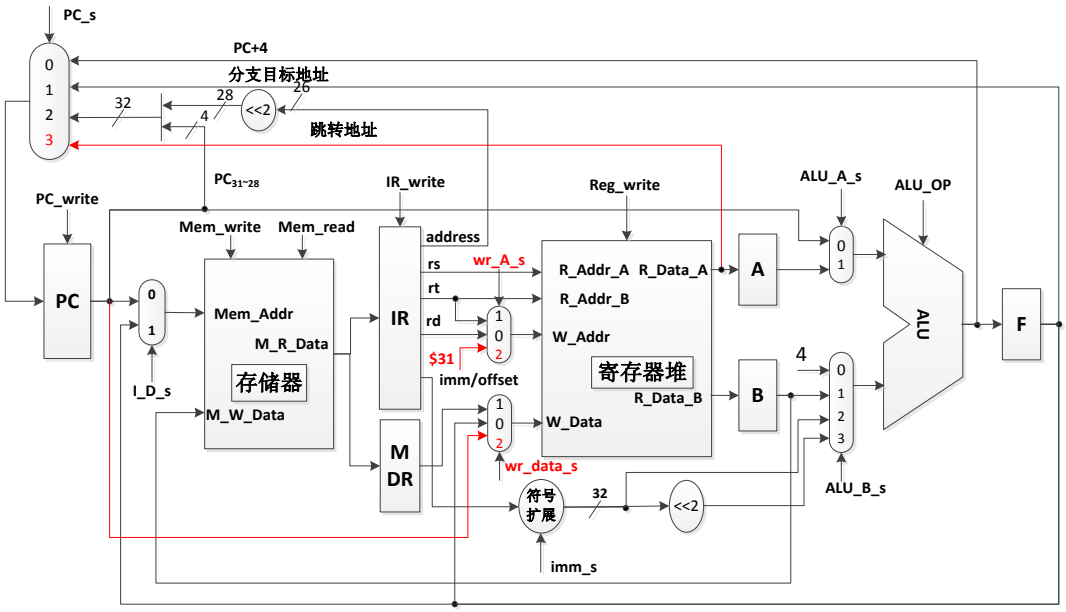
（2）画出电路图：



7.15 假如要在图 7-42 所示的多周期 CPU 的数据通路上，实现以下两条转移指令，请分析是否可行？如果不行，请尝试增加部件或者更改数据通路，以实现以下两条指令；阐述你的方案，画出数据通路图，说明指令执行过程。

（1） jr rs；无条件跳转：rs → PC

（2） jal label；无条件跳转并链接：(PC+4)→$31, {(PC+4)高 4 位,address,0,0}→PC



（1） 为实现 jr rs 指令，可以添加一条链路、在 PC 置位的多路选择器中添加一个选项，变成 4 选 1，当 PC_s=11 时，将寄存器堆的 R_data_A 端口数据送到多路选择器的选项 4 处。

（2） 为实现 jal label 指令，则需要在寄存器写端口的地址和数据端的多路选择器多一个选项，分别是$31 和 PC+4，也即 3 选 1。控制信号改为 wr_A_s 和 wr_data_s。指令执行过程：

时钟周期	操作	发送控制信号
（1）jr rs		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=+; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→PC	PC_s=11, PC_write;
（2）jal Label		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=+; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	PC+4→reg[$31] {(PC+4)高 4 位,add-ress,0,0}→PC	wr_A_s=10, wr_data_s=10, Reg_write PC_s=10, PC_write;

7.16 MIPS 为 RISC CPU，指令集相对简单，实现复杂指令的方法是将其用多条简单指令来代替。思考 MIPS 如何实现交换指令 swap rs,rt（将寄存器$rs 和$rt 中的内容交换）？可以使用多条简单指令实现，也可以以单周期和多周期 CPU 的方式实现，同时考虑允许有另一个寄存器内容被改变和不允许其他寄存器内容被改变两种情况。

（1） 用三条简单指令实现：

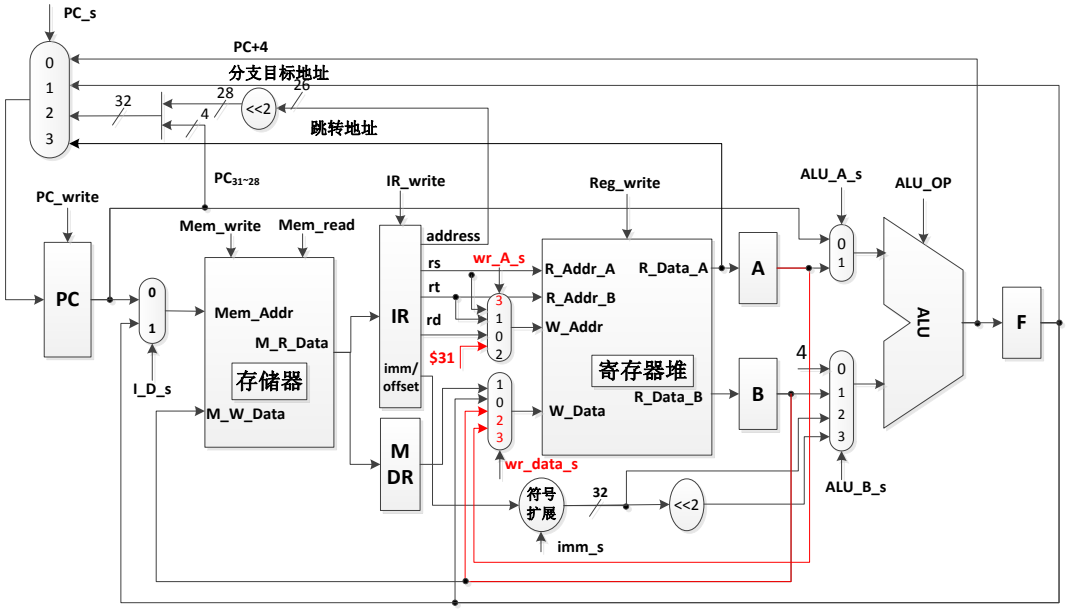
使用汇编器保留的寄存器$1($at)做交换媒介：

```
xor   $1,$zero,$rt        ；rt→$1
xori  rt,rs,0x0000        ；rs⊕0→rt
xor   $rd,$zero, $1       ；  $1⊕0→rd，此处 rd 的编码为 rs
```

（2） 使用多周期 CPU 实现：

不用改变另外一个寄存器的内容；

添加两个数据通路：寄存器写地址端添加一个来源，即指令码的 rs 字段；寄存器写数据端也添加两个来源，即暂存器 A 的输出和暂存器 B 的输出。



时钟周期	操作	发送控制信号
swap rs,rt		
M0	Mem[PC]→IR, PC+4→PC	I_D_s=0,Mem_read,IR_write; ALU_OP=+; ALU_A_s=0, ALU_B_s=00, PC_s=00, PC_write;
M1	Reg[rs]→A, Reg[rt]→B	
M2	A→Reg[rt]	wr_A_s=01, wr_data_s=11,Reg_write;
M3	B→Reg[rs]	wr_A_s=11, wr_data_s=10,Reg_write;

（3） 如果使用单周期 CPU 在一个时钟周期中实现数据交换，那么需要将寄存器堆的写端口修改成双端口写的结构，略。

7.17 参见图 7.60 数据通路，指令“INC R1”将 R1 寄存器的内容加 1，画出其指令周期微程序流程图，并根据表 7.16 和表 7.17 写出每一条微指令码。

表 7.16 模型机微指令格式

$M_{23}M_{22}M_{21}$	$M_{20}M_{19}M_{18}$	$M_{17}M_{16}M_{15}$	M_{14}	M_{13}	M_{12}	M_{11}	M_{10}	M_9	M_8	M_7	$M_6M_5M_4M_3M_2M_1M_0$
BTO(3)	OTB(3)	FUNC(3)	FS	S_3	S_2	S_1	S_0	M	Ci	空	MA_6-MA_0(7)

表 7.17 微指令字段编码表

编码	BTO	OTB	FS=1	FS=0
			FUNC	FUNC
000			PC+1	
001	B-DA1(t4)	$\overline{\mathbf{ALU-B}}$(t2)	$\overline{\mathbf{J1}}$(t2)	$\overline{\mathbf{M-W}}$(t3)
010	B-DA2(t4)	$\overline{\mathbf{299-B}}$(t2)	$\overline{\mathbf{J2}}$(t2)	$\overline{\mathbf{M-R}}$(t2)
011	B-IR(t3)	$\overline{\mathbf{SR-B}}$(t2)	$\overline{\mathbf{J3}}$(t2)	$\overline{\mathbf{IO-W}}$(t3)
100	B-DR(t4)	$\overline{\mathbf{DR-B}}$(t2)	$\overline{\mathbf{J4}}$(t2)	$\overline{\mathbf{IO-R}}$(t2)
101	B-SP(t4)	$\overline{\mathbf{SI-B}}$(t2)	$\overline{\mathbf{J5}}$(t2)	$\overline{\mathbf{INT-R}}$(t2)
110	B-AR(t3)	$\overline{\mathbf{SP-B}}$(t2)	$\overline{\mathbf{CyCn}}$(t2)	$\overline{\mathbf{INT-E}}$(t2)
111	$\overline{\mathbf{B-PC}}$(t4)	$\overline{\mathbf{PC-B}}$(t2)	$\overline{\mathbf{CyNCn}}$(t2)	

参考答案：

将 INC R1 指令抽象为 INC DR 指令，假设其指令操作码为 0000，格式为：

1 1	OP（0000）	DR（2 位）

则 INC 指令的微程序流程图如下图所示，其微代码如下表所示。

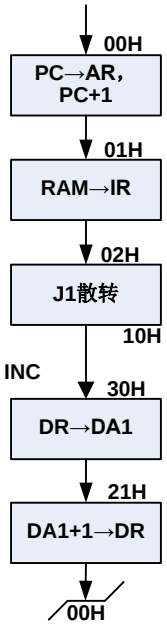


图7.1 习题 7.8 微程序流程图

表7.1 习题 7.8 微码表

微地址	微指令	微指令编码（M_{23}～M_0）					
		BTO	OTB	FUNC	FS	S_3～S_0 MCi	下址
00H	PC→AR，PC+1	110	111	000	1	000000	0000001
01H	RAM→IR	011	000	010	0	000000	0000010
02H	J1#散转	000	000	001	1	000000	0010000
30H	DR→DA_1	001	100	000	0	000000	0100001
21H	DA_1+1→DR	100	001	111	1	000000	0000001

7.18 根据图 7.60 所示的模型机结构和数据通路，写出以下指令从取指到执行的全部微操作序列，说明各条指令需要哪几个机器周期，需要几次访问内存及完成什么操作。指令格式如下：

Opcode（4 位）	寻址方式(2 位)	寄存器编号（2 位）
A 地址（8 位）		

（1） SUB [A]，R ，该指令完成（A）－R→ A，源操作数的寻址方式为寄存器寻址，目标操作数的寻址方式为直接寻址，A 为目标操作数的有效地址。

（2） JMP 偏移量，该指令完成 PC＋偏移量→PC，A 为其偏移量。

参考答案：如表 7.4 所示。

表7.2 习题 7.9 指令执行过程

指令	机器周期	操作	微操作序列	访存

取指令（公操作）	M_0	PC→AR，PC+1	$\overline{\mathbf{PC-B}}$，B-AR，PC+1	
	M_1	RAM→IR	$\overline{\mathbf{M-R}}$，B-IR	取指令第一字
	M_2	$\overline{\mathbf{J1}}$散转	$\overline{\mathbf{J1}}$	
SUB 指令的执行阶段	M_3	PC→AR，PC+1	$\overline{\mathbf{PC-B}}$，B-AR，PC+1	
	M_4	RAM→AR	$\overline{\mathbf{M-R}}$，B-AR	取指令第二字 A
	M_5	RAM→DA_1	$\overline{\mathbf{M-R}}$，B-DA_1	取源操作数（A）
	M_6	DR→DA_2	$\overline{\mathbf{DR-B}}$，B-DA_2	
	M_7	DA_1－DA_2→RAM	$\overline{\mathbf{ALU-B}}$，$S_3$~$S_0MC_i$=0111000，$\overline{\mathbf{M-W}}$	写运算结果
JMP 指令的执行阶段	M_3	PC→AR，PC+1	$\overline{\mathbf{PC-B}}$，B-AR，PC+1	
	M_4	RAM→DA_1	$\overline{\mathbf{M-R}}$，B-DA_1	取指令第二字，即偏移量 A
	M_5	PC→DA_2	$\overline{\mathbf{PC-B}}$，B-DA_2	
	M_6	DA_1 + DA_2→PC	$\overline{\mathbf{ALU-B}}$，$S_3$~$S_0MC_i$=100101，$\overline{\mathbf{B-PC}}$，PC+1	

所以：SUB 指令含 8 个机器周期，共访存 4 次；JMP 指令含 7 个机器周期，访存 2 次。

7.19　假设某机器主要部件有：程序计数器 PC、指令寄存器 IR、通用寄存器 R0～R3、暂存器 DD1 和 DD2、ALU、移位器、存储器地址寄存器 MAR 及存储器 M。

（1）　要求采用单总线结构，画出包含上述部件的逻辑框图，并注明数据流动方向。

（2）　画出 ADD （R1），（R2）指令在取指和执行阶段的操作步骤流程图。R1 寄存器存放目标操作数的存储器地址，R2 寄存器存放源操作数的存储器地址。

（3）　写出各操作步骤所需的全部微操作命令。

参考答案：

（1）　CPU 及存储器的逻辑框图如下图所示，数据流向见图示。控制信号有 in 和 out 两类，其中当 out 类信号有效时数据输出，无效时高阻抗。移位器（SSR）有右移（>>）和左移（<<）两种移位功能。寄存器堆由指令码中的源寄存器（SR）编号或者目的寄存器（DR）编号选择，可以读出或者写入；对于源寄存器，只能读出。CPU 内部总线和外部数据总线合二为一。

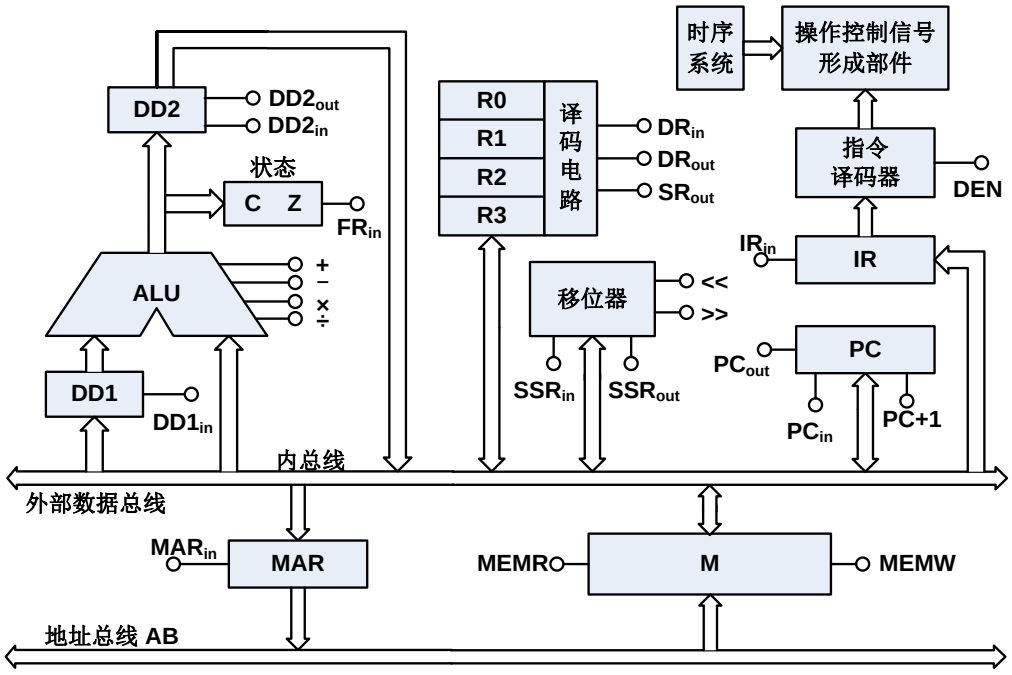


图7.2 习题 7.10 计算机逻辑框图

（2） 由题意知：操作数均为寄存器间接寻址，该 ADD 指令的流程图如图 7.4 所示。

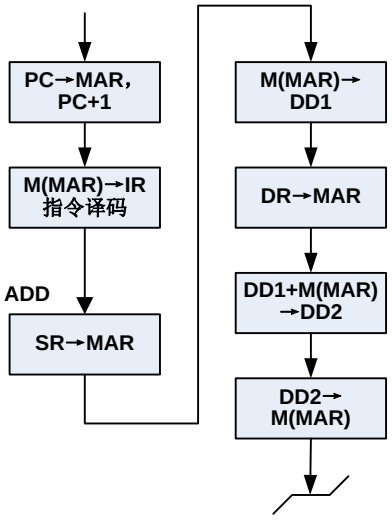


图7.3 习题 7.10 ADD 指令流程图

（3） 微操作命令如表 7.3 所示。

表7.3 习题 7.10 ADD 指令的微命令

序号	微指令	微命令
1	PC→MAR，PC+1	PC_{out}，MAR_{in}，PC+1
2	M(MAR)→IR 指令译码	MEMR，IR_{in}，DEN
3	SR→MAR	SR_{out}，MAR_{in}
4	M(MAR)→ DD1	MEMR，$DD1_{in}$

序号	微指令	微命令
5	DR→MAR	DR_{out}，MAR_{in}
6	DD1+M(MAR)→DD2	MEMR，+，$DD2_{in}$
7	DD2→ M(MAR)	$DD2_{out}$，MEMW

7.20 假设某机共有 76 条指令，平均每一条指令由 11 条微指令组成，其中有一条取指令的微指令是所有指令公共的，该机共有微命令 31 个，微指令的微操作码采用直接控制法。试问：

参考答案：

（1） 该机微指令长度为多少？

- 微指令的条数：76*10+1=761 条，所以控存容量的字数≥761，则控存地址至少 10 位（2^{10}＞761）。
- 微指令的控制字段 31 位，下址 10 位，共 41 位。

（2） 控制存储器的容量应为多少？

- 控存容量 2^{10}×41 位。

7.21 某机采用微程序控制方式，其控制存储器容量为 512×48（位），微程序在整个控制存储器中实现转移，可控制微程序的条件共 4 个，判别测试字段采用编译法。微指令采用水平型格式，后继微指令地址采用判定方式，如图 7.5 所示：

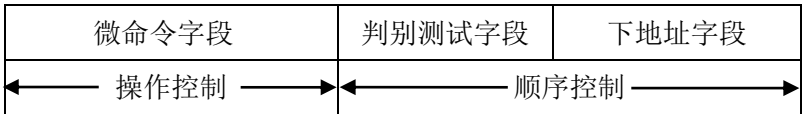


图7.4 习题 7.12 微指令格式图

参考答案：

（1） 微指令中的三个字段分别应多少位？

- 控存容量 512 字，512=2^9，而且微程序在整个控制存储器中实现转移，所以：下址字段为 9 位
- 可控制微程序的条件共 4=2^2 个，判别测试字段采用编译法，所以：判别测试字段=2 位；
- 微命令字段=48-9-2=37 位
- 如果考虑不转移的情况还需一个编码，则判别测试字段需要 3 位，微命令字段需要 36 位

（2） 画出对应这种微指令格式的微程序控制器逻辑框图。

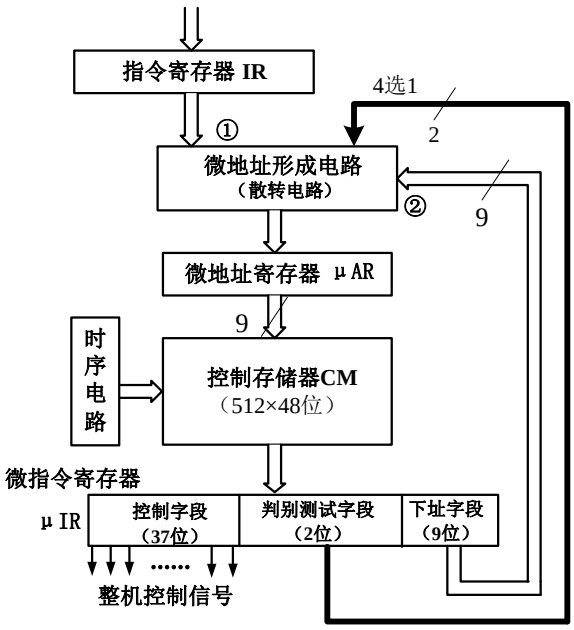


图7.5 习题 7.12 微程序控制器逻辑框图

7.22 什么叫互斥微命令？微指令控制字段主要有哪几种编码方法，各有何特点？

参考答案：

- 互斥微命令：是指在同一个微周期中不可能同时出现的微命令。
- 微指令控制字段主要有 4 种编码方法：
  - 直接控制法：控制字段的每一位代表一个微命令（控制信号），通过直接置某位为 0 或 1 来控制对应的微命令是否发送。优点是无需译码，执行速度快；微程序较短；缺点是微指令字长很长，占用控存容量大。
  - 全译码方式：将所有的控制信号进行编码，作为控制字段。在执行微指令时，译码产生各个微命令。优点是微指令字长很短；缺点是并行操作能力弱，微程序很长，执行速度慢。
  - 字段直接编译法：将控制字段分成若干段，每段通过编码/译码对应到各个控制信号。设计时，将相斥性微命令分在同一字段内，相容性微命令分在不同字段内。优点是并行操作能力较强，字长较短。
  - 字段间接编译法：某字段的编码含意，除了其本身的编码外，还需要由另一字段来加以解释。

7.23 图 7.7 为某模型机的微程序流程图，图中每一个框表示一条微指令。在点 X 处为指令译码后转入指令的微程序入口的多路分支点，由指令寄存器 IR 的 I5I4 两位来决定转入哪一个入口。在点 Y 处根据状态条件 F 实现条件转移，微指令中判断测试位为直接控制法。控存容量能容纳图 7.7 所列出的微指令即可。

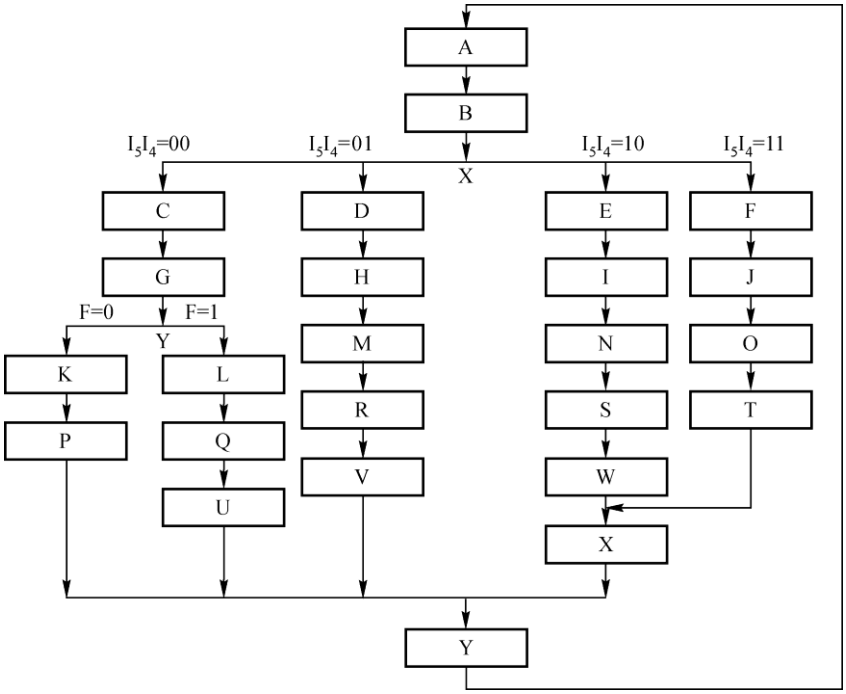


图7.6 模型机微程序流程图

参考答案：

（1） 微指令的判断测试位需要几位二进制？微指令的下址字段需要几位？

- 因为微指令的判断测试位为直接控制法，考虑 X 和 Y 两种条件转移方式及按照微指令的下址字段无条件转移，共有 3 种情况，则判断测试位需要 3 位；
- 图中共 25 条微指令，则控存地址需要 5 位，共有 32 个控存单元，所以微指令的下址字段有 5 位。

（2） 在图中标出每条微指令的微地址。

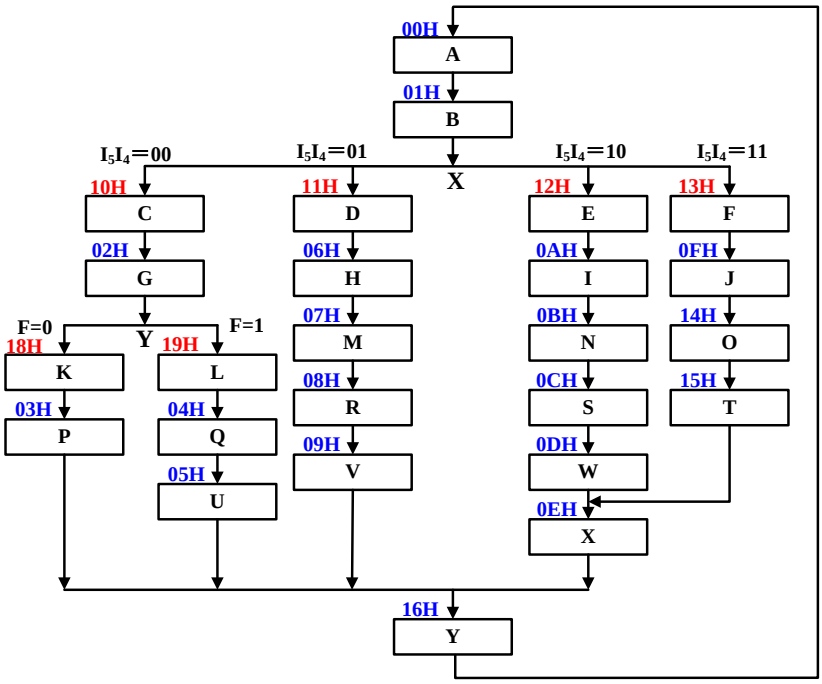


图7.7 习题 7.14 微地址分配

（3） 写出每条微指令的下址字段内容及判断测试字段码。

- 假设 3 位判别测试字段分别为 BX、BY、BZ，定义为：
  - 当 BX=1 时，实现指令散转，即由指令码的 I_5I_4 来产生微地址的低两位，微地址的高 3 位由下址字段的高 3 位产生。
  - 当 BY=1 时，实现微指令的条件转移，即由条件 F 来产生微地址的最低位，微地址的高 4 位由下址字段的高 4 位产生。
  - 当 BZ=1 时，直接由下址字段来产生微地址。

表7.4 习题 7.14 微指令编码

微指令	微地址	判断测试字段			下址字段	微指令	微地址	判断测试字段			下址字段
		BX	BY	BZ				BX	BY	BZ	
A	00H	0	0	1	01H	N	0BH	0	0	1	0CH
B	01H	**1**	0	0	10H	O	14H	0	0	1	15H
C	10H	0	0	1	02H	P	03H	0	0	1	16H
D	11H	0	0	1	06H	Q	04H	0	0	1	05H
E	12H	0	0	1	0AH	R	08H	0	0	1	09H
F	13H	0	0	1	0FH	S	0CH	0	0	1	0DH
G	02H	0	**1**	0	18H	T	15H	0	0	1	0EH
H	06H	0	0	1	07H	U	05H	0	0	1	16H
I	0AH	0	0	1	08H	V	09H	0	0	1	16H

微指令	微地址	判断测试字段			下址字段	微指令	微地址	判断测试字段			下址字段
		BX	BY	BZ				BX	BY	BZ	
J	0FH	0	0	1	14H	W	0DH	0	0	1	0EH
K	18H	0	0	1	03H	X	0E	0	0	1	16H
L	19H	0	0	1	04H	Y	16H	0	0	1	00H
M	07H	0	0	1	08H						

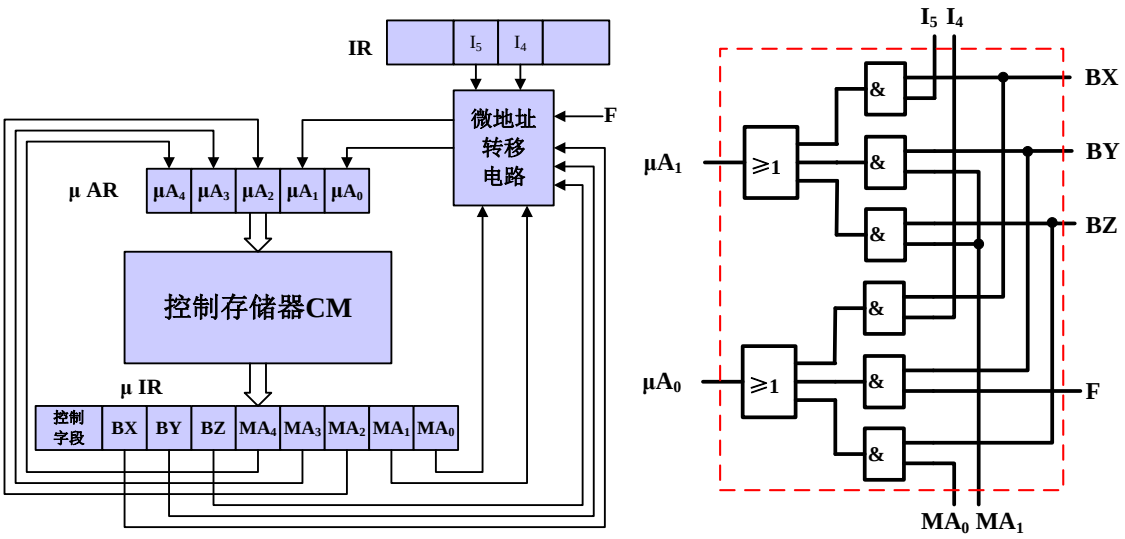
（4） 画出微地址转移逻辑电路图。

微地址转移逻辑的表达式为：

$$\mu A_1 = BX \bullet I_5 + BY \bullet MA_1 + BZ \bullet MA_1$$

$$\mu A_0 = BX \bullet I_4 + BY \bullet F + BZ \bullet MA_0$$

电路图如图 7.9 所示。



（a）微程序控制器框图 （b）微地址转移电路

图7.8 习题 7.14 微程序转移逻辑电路图

7.24 试分别简述硬布线控制器设计和微程序控制器设计的设计步骤和硬件组成，比较其各自的特点。

参考答案：

表7.5 习题 7.15 两种控制器对比

比较内容	硬布线控制器	微程序控制器
设计步骤	1. 确定指令系统，包括每条指令的格式、功能和寻址方式，分配操作码。 2. 确定 CPU 的内部结构，包括	1. 确定指令系统，包括每条指令的格式、功能和寻址方式，分配操作码。 2. 确定 CPU 的内部结构，包括

比较内容		硬布线控制器	微程序控制器
		运算器、控制器、时序系统、连接方式和数据通路。 3． 分析每条指令的执行过程，写出每个机器周期所必需发送的微操作控制信号序列。 4． 综合每个微操作控制信号的逻辑函数，并化简和优化。 5． 用逻辑电路实现。	运算器、控制器、时序系统、连接方式和数据通路。 3． 分析每条指令的执行过程，画出指令系统的微程序流程图。 4． 写出每条微指令所发送的微操作控制信号序列。 5． 设计微指令格式。 6． 分配各微指令的微地址，并编写微指令代码。 7． 将所有的微指令代码装入控制存储器的相应单元。
硬件组成		1、 专用寄存器：PC、IR、AR、DR 2、 指令译码器 3、 操作控制信号形成部件 4、 时序系统	1、 专用寄存器：PC、IR、AR、DR 2、 指令译码器 3、 控制存储器、微地址寄存器、微指令寄存器 4、 时序系统
特点比较	工作原理	微操作控制信号由组合逻辑电路根据当前的指令码、状态和时序，即时产生	微操作控制信号事先以微程序的形式存放在控存中，执行指令时读出即可
	执行速度	快	慢
	规整性	繁琐、不规整	较规整
	应用场合	RISC　CPU	CISC　CPU
	易扩充性	困难	易扩充修改