一. 课程设计步骤

1. 确定模型计算机功能及用途

完成一个较简单的计算机主机系统的设计，加深对微程序控制的计算机主机的基本构成、部件详设计、部件间的连接、微程序的编制与调试等全过程的体验和认识。所设计的模型机应具备：在自行设计的模型机指令集基础上，通过微程序实现对相应硬件的控制，实现模型机的特定功能，具体来说，应支持以下功能的用户程序实现：

(1) 从内存中取两个数，相加后将结果存入第三个内存单元；

(2) 在以的基础上，将相加过程中产生的"产生进位、溢出、结果为负、结果为零"状态存入状态寄存器(PSW)中；

(3) 在以上的基础上，支持条件跳转功能，即"若PSW具有某状态则跳转"；

(4) 实现乘法功能，采用累加方法，乘积16位；

(5) 程序运行结束后停机。

事实上，要设计完成以上任务的模型机，其具备的指令系统应当已经是图灵完备的，可以实现非常多的功能。

2. 指令系统

为实现相应功能，设计的指令系统有如下指令：

(1) 取数指令 LD Ri, AD，微程序入口地址10H：

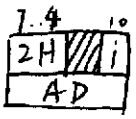
功能：将RAM中地址为AD的单元中数据存入寄存器中，(AD)→Ri；

格式：双字长、双操作数。

第一个字节高四位为操作码1H，低两位为寄存器编号(00.01.10.11)。

第二个字节为AD，即数据来源地址(直接寻址)。

(2) 存数指令 ST Ri, AD，微程序入口地址20H



功能：将Ri寄存器中的内容存入RAM中地址为AD的单元，Ri→(AD)；

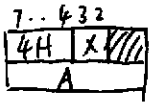
格式：双字长、双操作数。

第一个字节高四位为操作码 2H，低两位为寄存器编号。
第二个字节为 4D 即数据目标地址（直接寻址）。

(3) 停机指令 HALT，微程序入口地址 30H：

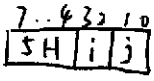
功能：停机，此指令执行完毕后不再执行任何周期；
格式：单字长，无操作数，高四位为操作码 3H。

(4) 条件跳转指令 ~~JXA~~ JX A，微程序入口地址 40H：



功能：含为 JC、JV、JN、JZ 即"进位/溢出/为负/为零时跳转"，跳转意即将程序计数器 PC 的值加上 A。若条件不满足则 PC+1→PC；
格式：双字长，双操作数。
第一个字节高四位为操作码 4H，此后两位代表 X：(00：C，01：V，10：N，11：Z)。第二个字节为偏移量，用补码表示(带立即数)。

(5) 加法指令 ADD Ri, Rj，微程序入口地址 50H：



功能：将寄存器 Ri、Rj 的和存入寄存器 Rj 中，并将产生的状态打入 PSW，(Ri)+(Rj)→Rj；
格式：单字长，双操作数。高四位为操作码 5H，接下来两位编码代表 i，最后两位代表 j。

(6) 无条件跳转指令 JMP A，微程序~~入口~~入口地址 60H：

功能：将 PC 的值加上 A，即相对寻址，PC+A→PC；
格式：双字长，单操作数。首字节高四位为操作码 6H，第二个字节为偏移量(立即数)。

(7) 递增指令 INC Rj，微程序入口地址 70H：

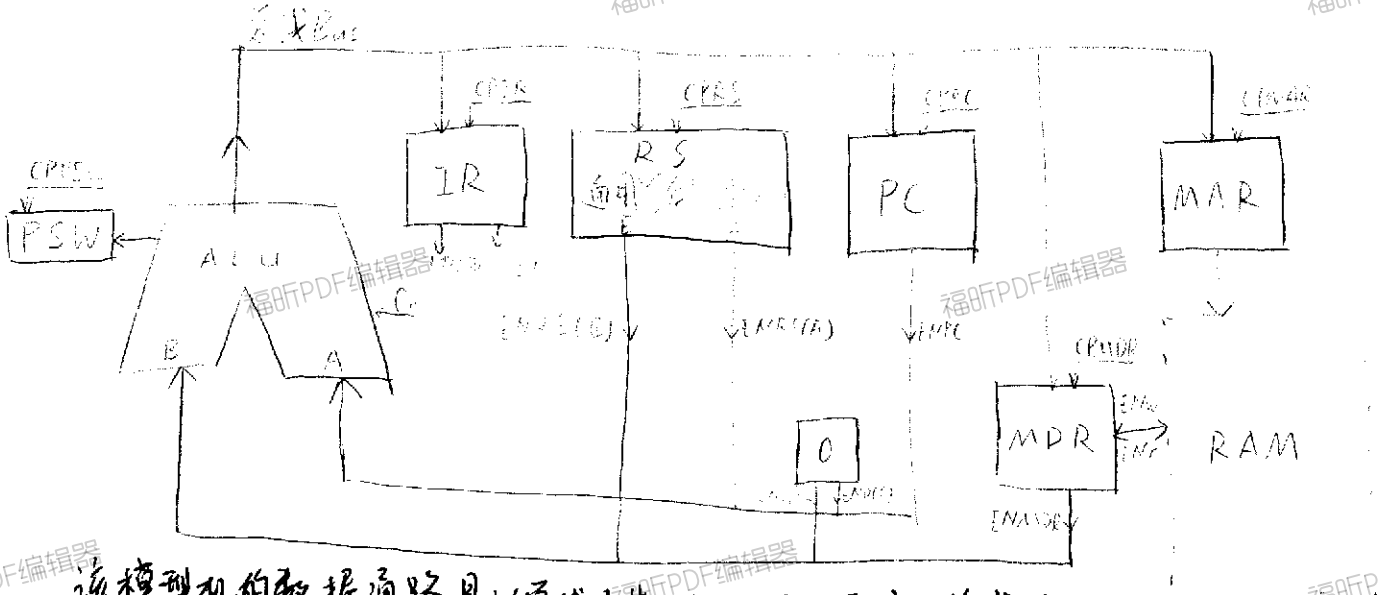
功能：将寄存器 Ri 的值加 1 模 256（按无符号数处理）；
格式：单字长，单操作数，高四位为操作码 7H，低两位代表 i。

(8) 带状态递增指令 INC+ Ri，微程序入口地址 A0H：

功能：同递增指令，但将递增运算中产生的状态打入 PSW；
格式：单字长，单操作数。高四位为操作码 AH，低两位代表 i。

3. 总体结构与数据通路

总体结构图：



该模型机的数据通路是以总线为基础，以ALU为核心构成的。

打入脉冲共6种，他们可以将数据打入到相应的寄存器。
分别为：CPPSW、CPIR、CPRS、CPPC、CPMDR、CPMAR。
其中CPPSW独占于其他五种，另外五种由单一控制信号决定，故互斥。

三态门输出使能信号共6种（扩展要求需额外的一种，见扩展说明部分），他们控制ALU的A、B两端的数据来源。
与A相连的有3种：ENRS(A)、ENO(A)、ENPC，即A端数据来源；
与B相连的有3种：ENRS(B)、ENO(B)、ENMDR，即B端数据来源。

此外还有MDR与RAM交互的双向三态门，由单独的控制信号控制RAM的读写，实现CPU与内存数据的隔离。

4. 指令执行流程

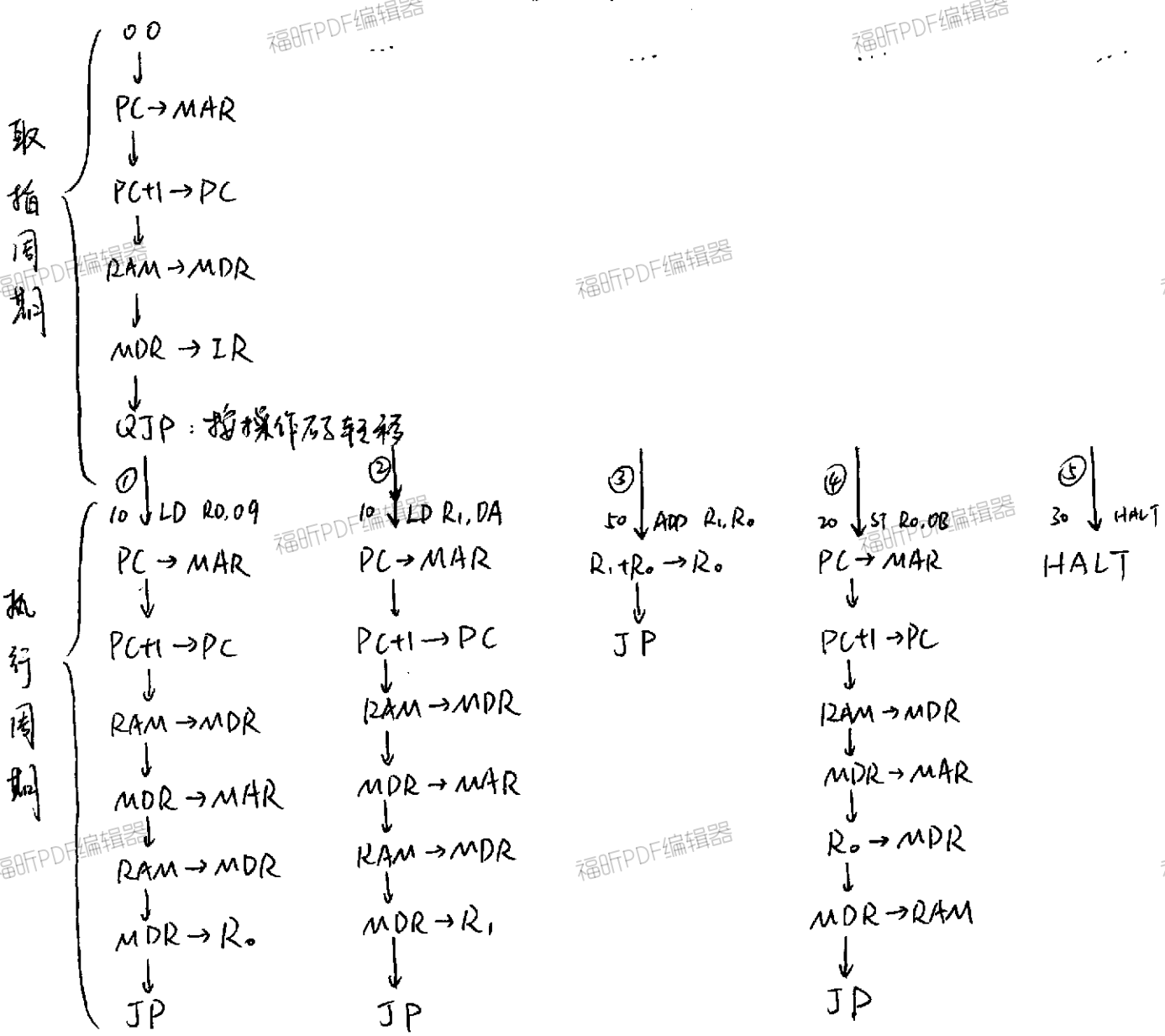
程序包含的指令，按地址顺序，从00单元起存放于RAM中，由程序计数器控制访问。除相对跳转的指令JX与JMP外，指令按地址顺序执行。

每条指令的执行具有两个周期：

(1) 取指周期。此周期由控存内入口地址为00的取指微程序控制（见附录2），该周期开始时，PC指向该指令的地址，由微程序负责将该单元内指令放入IR中，并使用QJP微指令跳转到对应微程序入口，将uPC转移到该指令对应的微程序入口。

(2) 执行周期。指存开始执行微程序，IR的低四位可能有参数（通用寄存器编号，JX指令的CNVZ），在程序执行时起控制作用。微程序末尾通过JP(0)指令，从PC转移到00地址，从而进入下一条指令的取指周期。

如此按规定的顺序执行每个指令周期，直到停机指令的执行周期结束，模型机不再工作。以简单加法（附录表3-2）为例，示意指令执行流程：



说明：1. LD、ST指令均采用双字长、立即数寻址方式；

2. 送指令地址的微操作，安排在取指阶段的第一拍；

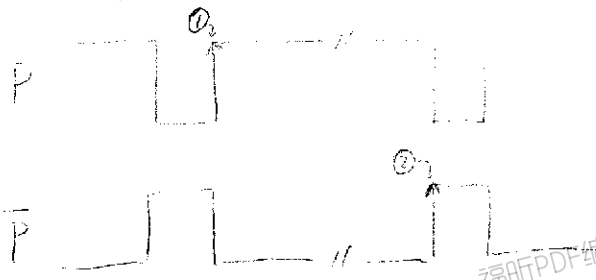
3. 取指微程序入口固定为指存的00H，通过设置μPC的初值为0实现；

4. 第一条机器指令放在00H开始的RAM单元，通过设置PC初值为0实现第一条机器指令的取指数。

5. 微程序流程

模型机采用微程序控制模式，将完成某任务需要的控制命令组合在一起连续地写在控存中形成微程序，供指令通过入口调用。每个控制命令对应一种微操作，这需要一定的时序系统设计得以完成。

(1) 微程序控制器时序



将 $\overline{\mu RD}$ 接地，将控存设为始终可读。

① P脉冲的上升沿：将读出的微指令送往 μIR；

② $\overline{P}$ 脉冲的上升沿：将运算结果送往指定的寄存器，并将脉冲打入 μPC 形成下一条微指令的地址。

(2) 微指令格式

① 微指令字段定义

a) ALU-B端数据来源：μIR_{15} μIR_{14}

0	0	ENO
0	1	ENRS
1	0	ENMDR
1	1	ENXJP (见扩展要求)

b) ALU-A端数据来源：μIR_{13} μIR_{12}

0	0	ENO
0	1	ENRS
1	0	备用
1	1	ENPC

c) 输出分配(打入脉冲): μIR_{11} μIR_{10} μIR_9

0	0	0	CPRS
0	0	1	备用
0	1	0	备用
0	1	1	备用
1	0	0	CPIR
1	0	1	CPMDR
1	1	0	CPMAR
1	1	1	CPPC

d) 低位进位控制: μIR_8

0	$C_0=0$
1	$C_0=1$

e) PSW 打入脉冲: μIR_6

0	无 CPPSW
1	有 CPPSW

f) 存储器读写控制: μIR_5 μIR_4

0	1	$\overline{WR}=0$
1	0	$\overline{RD}=0$

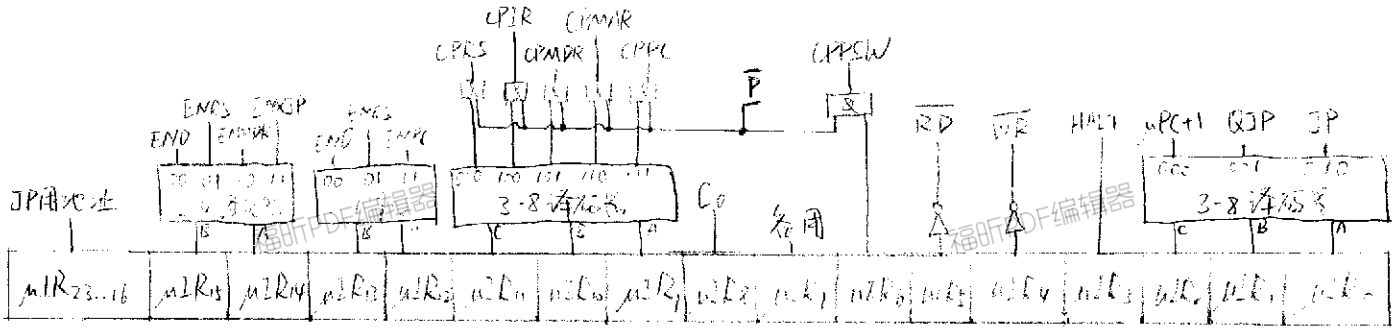
g) 停机控制: μIR_3

0	不停机
1	停机

h) 后继微地址形成方式:

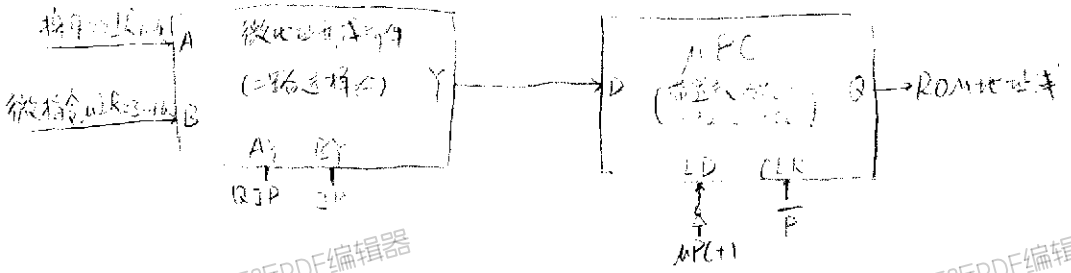
μIR_2	μIR_1	μIR_0	
0	0	0	$\mu PC+1$, 顺序执行
0	0	1	以IP, 高四位指操作码转移. 低四位为0
0	1	0	JP. 无条件转移. 微地址由 μIR_{23}~μIR_{16} 提供
其他			备用

② 微命令形成逻辑



③ 后继微地址产生逻辑

使用以下结构：



此处二路选择器经过特殊设计，即A端仅接受高四位，低四位始终为0。

由图可知：
a) μPC+1=1 时，μPC+1
b) QJP=1 时，指操作码高四位转移
c) JP=1 时，指微指令无条件转移

即：a) μPC+1=1 时，Q=Q+1
b) QJP=1 时，Q=IR7IR6IR5IR40000
c) JP=1 时，Q=μIR23...16

(3) 支持的微操作

经由以上设计，下面列举所有支持的且用到的微指令，具体数据流向见队友报告，指令具体内容见附录2：

① 通过总线
a) 送地址指令 PC→MAR；
b) 程序计数器加1 PC+1→PC；
c) 取指令 MDR→IR；
d) 取数到通用寄存器指令 MDR→R；
e) 寻址指令 MDR→MAR；

f) 准备存数指令 Ri → MDR；

g) 累加指令 Ri + Rj → Rj；

h) 无条件跳转指令 MDR + PC → PC；

i) 递增指令 Ri + 1 → Ri；

j) 保存状态递增指令 Ri + 1 → Ri +；

注：1. 涉及寄存器组的指令，具体寄存器编号的确定见通用寄存器组部件设计；

2. 条件跳转指令 XJP + PC → PC 见扩展部分的实现方法说明。

② RAM与MDR之间

a) 读内存指令 RAM → MDR，RD = 1

b) 写内存指令 MDR → RAM，WR = 1

③ ~~跳转指令~~ 特殊转移指令

a) 无条件转移指令 JP

b) 按操作码转移指令 QJP

④ 停机指令

a) 停机指令 HALT

6. 各部件设计

(1) 启停器

电路图见附录图 1-23。

输出脉冲前的3路与门分别连接连续脉冲与启动条件。清零信号 CLR 为低电平时，将 DFF 触发器输出置为低电平。

启动时，将 CLR 置 1，HALT 置 0，然后触发 START 信号，即可使 CP 输出微指令的时钟信号。

(2) 八位寄存器

电路图见附录图 1-2。

由八个 DFF 置数器并联而成，可以统一置数、复位。

此寄存器结构较为简单，直接将被封装地应用于各部件。

数据宽度为 8 位，模型机使用总体功能封装为一个组件，如附录图 1-3。

(3) 三态门

① 单向三态门

见附录图1-16.

由8个三态门并联而成，可以统一对输入进行使能控制，在各部件中有普遍应用。

② 双向三态门

见附录图1-18。

由8个单位组成，每个单位由一个双向、一个输入、一个输出端口组成，有两个使能信号对其双向端口的数据流向进行统一控制。

在RAM与MDR数据交互时有应用，故被封装入MDR中。

(4) 译码器

译码器将二进制控制信号转化为直接的控制信号。

① 2-4译码器

单个见附录图1-8，两个并列使用的见图1-9.

② 3-8译码器

见图1-6。

均为高电平有效。

(5) 内存数据寄存器 MDR

见附录图1-24.

由三个单向三态门、一个双向三态门和一个八位寄存器封装而成。

其中两个与RAM有关的使能信号控制MDR的数据~~来源~~输入：

① 写信号有效时，与系统信号配合可将MDR数据写入RAM；

② 读信号有效时，与系统信号配合可从RAM数据读出到MDR。

使能信号ENQ控制MDR内数据流向ALU-B端。

(6) 选择器

① 二路选择器

如图1-11。

将两个宽度为8的输入A、B，通过控制信号选择其中一个输出。

二路选择器被应用于各处，其中微地址寄存器是一个将H端高四位截取，并在低位补零的二路选择器，见图1-21。

② 四路选择器

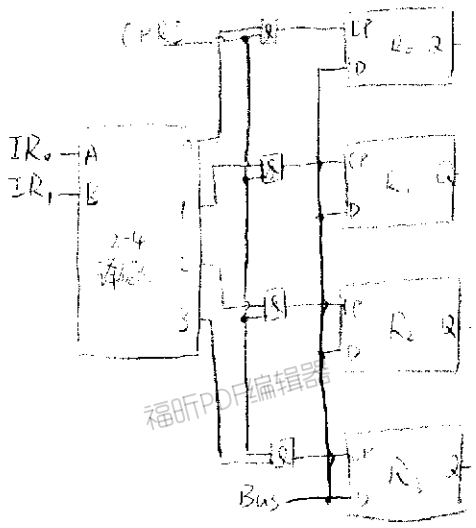
见图1-13，为三个二路选择器组合而成，作用为选择4路数据，去用于通用寄存器组中。

(7) 通用寄存器组RS

见附录图1-25。

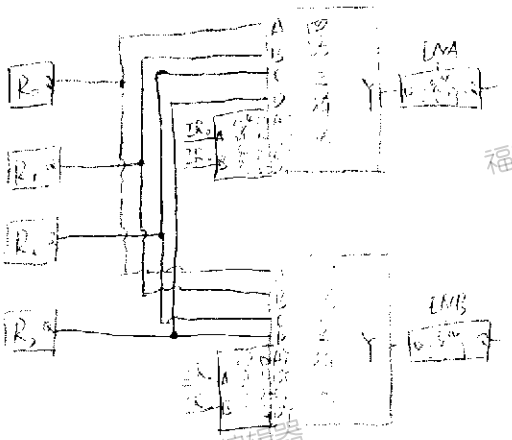
以4个八位寄存器为核心封装成的通用寄存器组，可以通过来自IR低四位的控制信号控制具体寄存器的读写。总体分为两部分：

① 数据打入部分



IR低两位译出4个互斥的信号，分别表数据要打入的寄存器编号，配合外界打入的CPRS脉冲，可以使总线内数据打入IR1 IR0指定的通用寄存器。

② 数据输出部分



与打入结构类似，IR1 IR0控制寄存器组A端的输出，IR3 IR2控制B端的输出，每一端都可以输出任一个寄存器的值，所以两端数据可以是相同的。

此外还有ENA、ENB两个使能信号，只有该信号有效时该输出端才会输出到ALU的输入端。

注意到IR1 IR0是复用的，因此执行形如ADD Ri,Rj操作时，和宽是会锁保存到IR低两位表示的寄存器（本系统为Rj）内，不能打入其他寄存器。

(8) 运算器

模型机使用的加法器见图1-14，将两片74181通过一片74182并联，并将控制位固定为仅能进行加法运算，得到超前进位加法器。同时产生C、V、N、Z的状态位，设加数为$A_{7..0}$、$B_{7..0}$，和为$S_{7..0}$，则产生的逻辑如下：(均为高电平有效)

① C进位：即高位74181的$\overline{CN_4}$，对$\overline{CN_4}$取反得到。

② V溢出：$\overline{A_7 \oplus B_7} \oplus S_7$；

③ N结果为负：S_7；

④ Z结果为零：$\overline{\sum_{i=0}^{7} S_i}$。

同时题目要求将状态打入PSW，故最终封装为图1-22，即与一个寄存器封装，可通过控制信号有选择地将本次运算结果打入PSW。

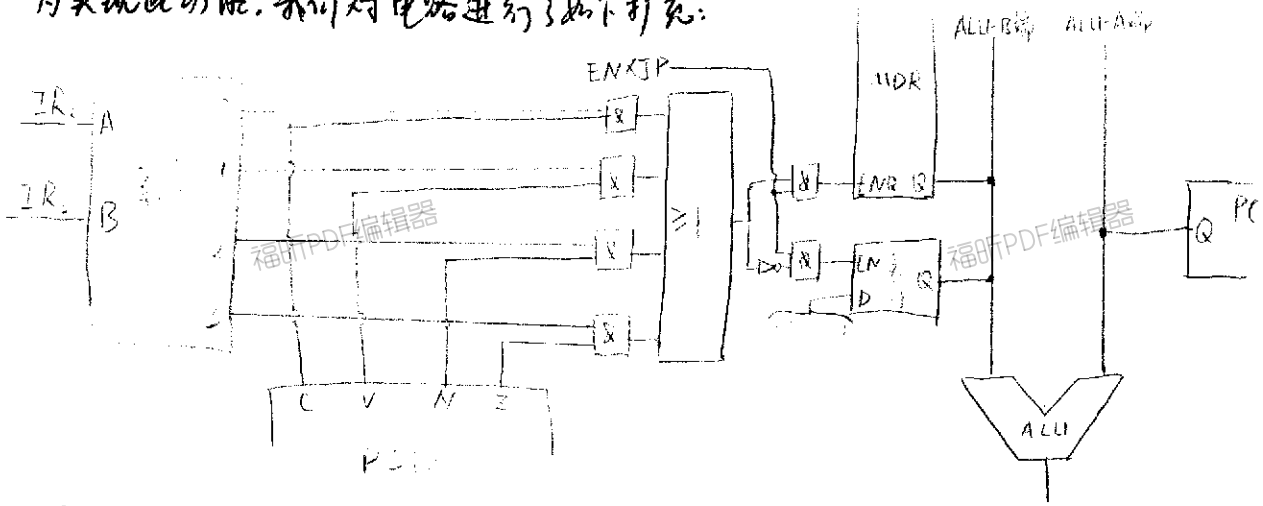
(9) 计数器

从PC的设计采用带置数功能的加1模256计数器，使用两片74161同步控制位串联而成，见图1-4。

二. 扩展部分的实现说明

1. 扩展要求(一)的实现说明

本次扩展要求增加条件指令JX A。即前文"一.2.(4)条件跳转指令JX A"。我们对此的理解是在PSW具有X状态时，执行PC+A→PC，否则PC+1→PC。为实现此功能，我们对电路进行了如下扩充：



图中ENXJP对应前文"一.5.(2)①a)ALU-B端数据来源"部分中一个特殊的使能信号。

可见，通过图中电路的逻辑，可以根据PSW的状态选择对应的值输入到ALU-B端，从而实现有条件地改变PC的值。

与之对应的微操作XJP+PC→PC的数据流向，此微操作亦收录于附录2中：

XJP(MDR/01H) → ALU-B
PC → ALU-A
ALU → Bus → PC

此操作要求此前已将A打入MDR，可通过微程序控制实现，此处不多加赘述。

完成后，可修改加法程序，使结果溢出时执行该操作，使PC跳转到异常处理程序段，进行将某单元地内容修改为FFH的操作。程序见附录表3-3溢出异常加法。

```
LD   R0, (16)→A1   ← 加数
LD   R1, (17)→A2
ADD  R1, R0
ST   R0, (18)→A3   ← 结果
JV   2
HALT
```

JV 2 ⇒ 若PSW中溢出位有效，转异常处理程序第一条指令。

异常处理程序：

```
LD   R0, (19)→内容为FFH
ST   R0, (20)→A4
HALT
```

2. 扩展要求(二)的实现说明

我们完成了累加法，乘积16位的要求。

为了完成该要求，我们添加了如下机器指令：

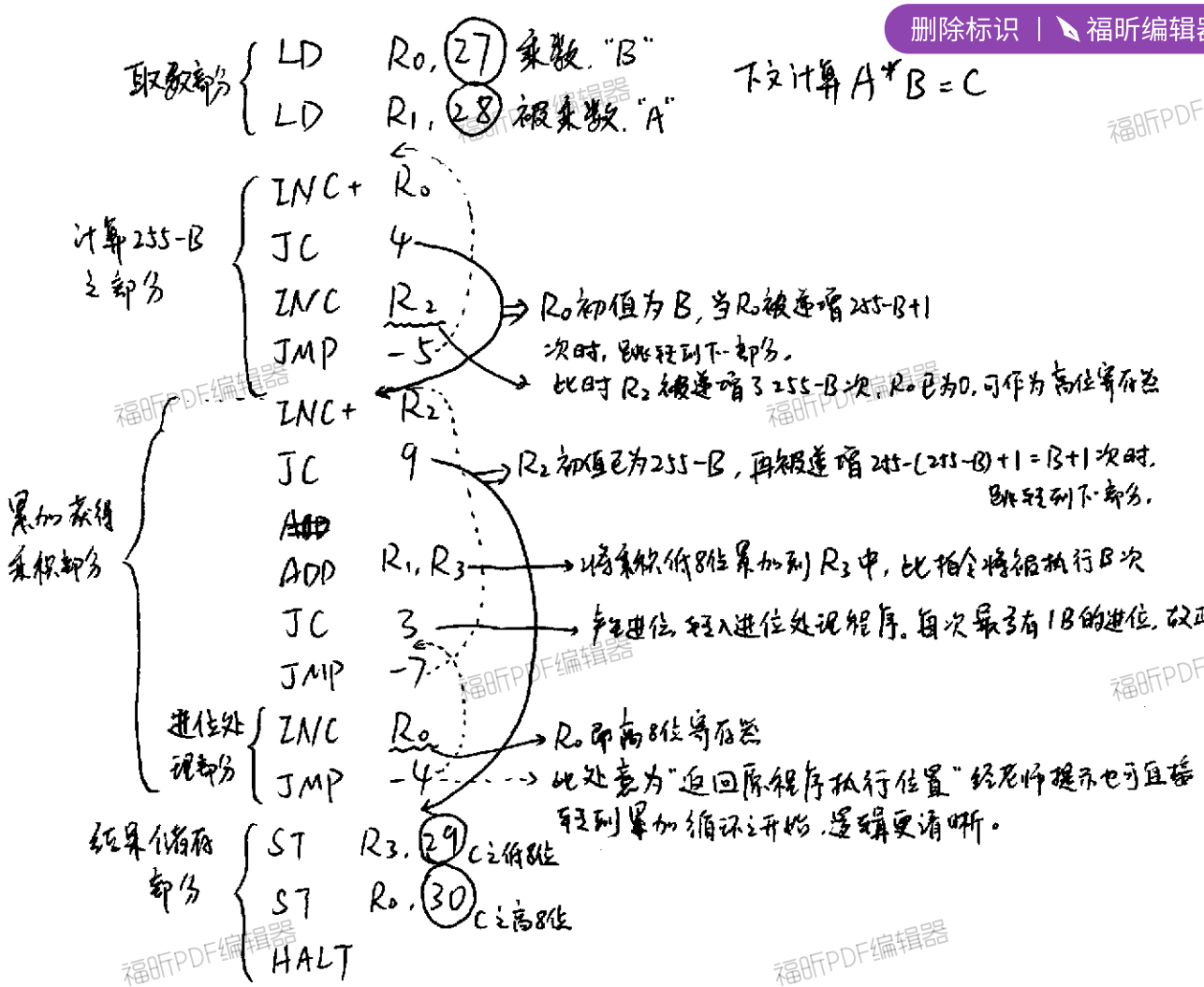
(1) 一、2.(6) 无条件跳转指令 JMP A；

(2) 一、2.(7) 递增指令 INC R；

(3) 一、2.(8) 带状态位递增指令 INC+ R；

此后即可编制程序完成该任务，具体程序见附录表3-4乘法。

(程序分析见下页)



乘法在演示时出现了结果错误，经过当场initial分析为程序错误，但事后经过分析发现程序实际上无误，是ALU的状态产生部分的逻辑电路出了问题，详见下文"课程设计总结部分。

三. 系统的调试、测试

本部分出现的具体问题将在"课程设计总结"部分说明。

1. 实机前的调试

在下载到FPGA之前，需要通过Quartus II软件的编译，期间会出一些问题，修改后直至编译通过。此过程保证连线具有逻辑(不一定正确)，且能被下载到FPGA中。

2. 调试微程序的执行

下载电路至FPGA后，将取指微程序写入指存，将连续脉冲频率降低至2Hz，观察FPGA上显示ROM内容的灯泡状态是否与取指微程序一致。

此过程可以保证ROM写入、μPC+1的后继微地址形成方式正常。

3. 调试部分寄存器

从此调试阶段开始，使用附录表3-1中存取数据的程序调试，即将该程序写入RAM。

由队友连线可以通过灯泡观察寄存器内容是否正常，比如MDR是否可以取出正确地址的数据，IR中是否有首条指令的内容，PC是否在恰当的时刻+1，MAR是否被正确设置为PC的值。

此过程保证打入脉冲、读取信号、部分寄存器、ALU直通与递增功能的正常工作。

4. 调试其他后续微地址形成方式

仍然依速观察MIR的变化，观察其是否正确对应各个指令周期的微程序内容，直至停机。

此过程保证QJP、JP的正常工作，即微地址形成部件的正常工作。

5. 调试通用寄存器组

观察R0是否如同微程序指示一样从内存中取出了正确的数据。

此过程保证控制寄存器组的打入、读出信号一定程度上的正确性。

6. 调试内存写入

观察程序结束后的RAM内容，是否将指定数据写入到指定的内存单元。

此过程保证存数指令各控制信号正常。

7. 调试加法程序

将RAM内容更换为表3-2简单加法的内容，观察结果是否正确。与执行过程中各寄存器状态，更换不同数据观察PSW状态等。

此过程进一步保证了与寄存器组有关的控制信号被正确使用、PSW状态正常、ALU加法功能的控制信号正确。

8. 调试条件跳转程序

将RAM内容更换为表3-3循环累加法，再观察寄存器状态与结果，更换不同的数据查看结果。

此过程保证了条件跳转指令的正确执行。

9. 调试乘法程序

将RAM内容更换为表3-4乘法，再观察寄存器状态与结果，注意更换数据。

此过程保证新增指令的正确执行，进一步保证寄存器组的正确工作（仅到此步才完全使用了4个通用寄存器）。

四、小组成员各自的任务及完成情况

焦境彤：任务：

各部件的设计及封装、总体数据通路的设计、指令系统格式的设计、微操作的确定、测试电路的绘制、测试过程中状态的观察、测试过程的记录、报告的撰写。

完成情况：正常完成。

张博宁：任务：

控制信号的设计与连接、指令系统内容的确定、微程序与程序的编写、测试方法的设计、观察测试状态、排查出现的问题、撰写实验报告。

完成情况：正常完成。

五、课程设计总结

1. 收获与体会

通过此次课程设计，我更加熟练了对于本学期之前、以及上学期学习的知识，尤其是关于指令系统、微程序控制、总线系统、时钟、时序的了解，感觉到从前只是纸上谈兵。：

(1) 对机器指令与微操作界定很模糊，不知道多么底层的操作才算"微操作"；

(2) 对时钟的作用并不了解，不知道有些操作只能通过脉冲完成；

(3) 对各芯片功能了解不足，在设定功能位时常有出错，也会弄混正负逻辑；

(4) 对机器代码，也就是一堆高低电平控制机器运行的具体方式完全不了解，现在深入最底层后，才知道哪些状态需要控制，用什么方式能跳到微程序入口等细节的操作。

(5) 对各元件标准的认识更加深刻，整机的实装运行需要各个元件的配合，封装的元件最好要进行功能测试。

(6) 亲自设计CPU，才知道以前的框框里是什么成分，可以由哪些信号控制、具体的职能、该怎么与其他元件配合等。

(7) 熟悉了Quartus II软件的使用，了解了FPGA的功能与作用，丰富了实践经历。

(8) 调试过程中磨炼了意志，让我更深刻地了解到了迎难而上的重要性。世上无难事，只怕有心人；不经历风雨，怎能见彩虹；

(9) 此次开发使用git进行版本控制，使用github进行托管，仓库地址 BoringZ/model_machine，让我熟悉了git的使用，增加了协作开发的经验。

从模型机正确编译到第一个取指周期，再到存取数值、加法乘法，以及调试过程异常艰难，

每一个进步都令人欢呼，极多加强了我的自信，也让我见证了实践的喜悦、与"实际是检验真理的唯一标准"的道理，也让我熟练了多人合作的方式。不能因为难找了错了原理，就可以把元件连接、微程序写错、地址算错这种错误，每一个细节都可能成为千里之堤的蚁穴。比起其他程序的bug，计组bug是最底层的，没有什么环境问题可以怀疑抱怨，但其原因通常也令人啼笑皆非，在这种环境中调试程序实在是一种独特的体验，也让我感到了独特的乐趣。

2. 遇到的问题

演示前的问题：

(1) 编译时有命名错误，比如空格、中文、关键字；

(2) 读写内存与下载程序后可能导致开机；

(3) PC的值不改变。原因为PC输出的使能端硬件连接为译码器的11，但微指令中为10；

(4) PC异常自增。74181做正逻辑加法时，Co未取反；

(5) JP、LOJP不能被正确执行，会跳转到说明的位置。后继微地址形成方式前的译码器与微指令对应出错；

(6) JP、LOJP不能被执行。原因为微地址由选择器打入μPC时未添加时钟信号；

(7) 无法开机。怀疑有时控存的值无法读出；

(8) 总线中的值总是被错误地打入到多个寄存器中。脉冲信号产生混乱；

(9) 执行双字长指令的周期后，有指令被跳过，地址计算错误，有多余PC+1→PC；

(10) 写入RAM的数据总是出现在FFH。发现是MAR后三态门没有被使能，高阻态被认为是FFH（也可能输出管脚默认为高电平）；

(11) 加法的结果错误，发现是按位或的结果；

(12) PSW的进位状态与实际情况相反；

(13) 加法的结果被打入了错误编号的寄存器，发现IR的控制位与微指令中不一致；

(14) 乘法指令死循环，发现地址计算错误；

(15) 模拟微程序时发现有多余的微指令。

演示时的问题：

(1) 大数乘法（FF*FF）结果出错，比正确结果少256，即少进了一次位；

(2) 将ALU的功能固定的为仅做加法，未将74181的功能充分利用，不利于其他功能的实现，如按位操作、逻辑运算，还增加了乘法程序的复杂性；

(3) 部分元件的封装程度不足。

3. 解决的方法

源未前的问题：

(1) 标记命名，直至编译通过；

(2) 每次执行电脑与FPGA操作的时候，将CLR置为低电平，禁止所有寄存器动动；

(3) 将PC输出使能信号修改为11；

(4) 将74181低位Co取反；

(5) 修改μPC的时钟信号为每一步均打入；

(6) 交换JP与DJP的寻址方式编码；

(7) 将 $\overline{\mu RD}$ 恒置为低电平；

(8) 之前将时钟信号T与输出控制位 μIR_{11}~μIR_9 在译码前取与，导致译码器有半个周期输入为000，而此情况与"打入寄存器组"信号重复，所以原来内容被重复打入寄存器组，后果将与门移到了每个寄存器组的CP前；

(9) 删除多余的PC+1→PC；

(10) 删除MAR后的三态门（在写信号时使能端也可，但多此一举）；

(11) 74181功能选择错误，应为"A加B"而非"A+B"；

(12) 将高位74181的 $\overline{CN4}$ 输出取反；

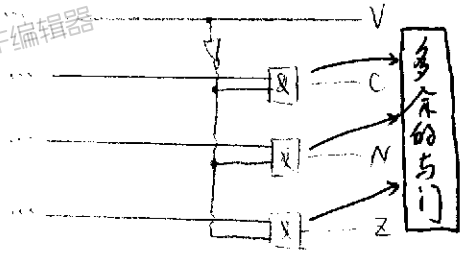
(13) 修改微指令，注意加法结果只能被打入IR低两位代表的寄存器，这是此模型机的硬件决定的。

(14) 修改相对跳转的立即数；

(15) 删除多余的微指令；

源未时的问题：

(1) 初步以为程序出错，即应该计算256-B。但事后发现原程序并无问题，数据在图本不会超过2位。以为是少了256，即少进了一次位，少跳进了一次进位处理程序，那么可能是有一次低位产生了进位没有跳入进位处理程序，即JC的判断有误，ALU未正确产生C信号。查看ALU后果真有此错误：



在最开始设计ALU的状态位时，认为"溢出会导致结果其他状态均无效"。但事实上，溢出指的是补码运算，进位对应无符号数运算，两者竟无关系！且另两种状态也与V无关，遂将与门删去，应当可以获得正确的结果。在有与门的情况下，导致有一次累加，低位寄存器为80H(1000 0000)，累加FFH后应为7FH(0111 1111)并产生进位C，此次运算若为补码则产生溢出V，但作为无符号数运算产生的进位C被多余的与门错误地消除了，导致跳转条件没有满足，高位寄存器少了一次递增，导致结果少了256。

没过这个错误，我意识到一定要测试边界值（猜测300#00，东测FF#FF，推断...），才能确保无虞。

(2) 使我认识到了74181的强大，如果早意识到，会给程序带来很大的简化。这与之前忘记取负一样，本质是对芯片功能的不熟悉与查看功能表时的漫不经心造成的；

(3) 对MDR、通用寄存器组等元件进行了封装。