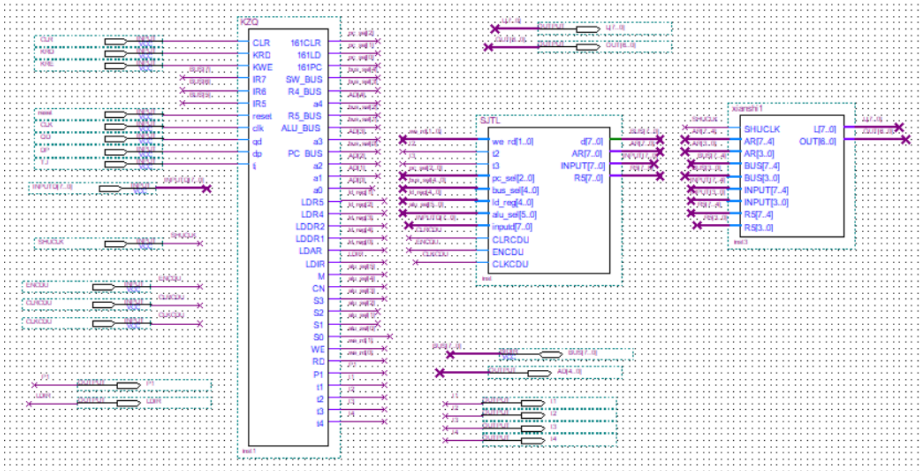
模型机报告

一、实验任务

1.实验目标任务

系统掌握计算机的组成和工作原理，能够自助熟练准确地阐述计算机执行机器指令的工作过程，熟练应用并设计微指令、微程序的设计及调试。

2.模型机顶层电路截图



文字说明

控制器模块（控制部件）

控制器模块时序分配同上一周实验控制器模块的实现。

输入端 CLR 为清零信号，用于将微命令、微地址、指令寄存器输出等等异步清零，低电平有效。输入端 KRD、KRE 为强读与强写信号，低电平异步有效，用来让程序强行进入强读循环、强写循环。输入端 IR7、IR6、IR5 连接着子模块 IR 寄存器的输入端（高三位），对应到总线上高三位（RAM 读出指令码的高三位）。控制器模块中整合了时序电路，控制器的输入端 CLK、QD、DP、TJ 连接到控制器子模块时序电路的时钟端与控制端用于产生时序 t1、t2、t3、t4。

输出端为 28 位微指令，a4-0 为 t2 时刻地址绝对映射读出的下一微地址，P1 为分支跳转信号（地址重新映射信号），t1、t2、t3、t4 是时序电路产生的时序，其余的均为控制器模块的输入端（控制信号）。

数据通路模块（执行部件）

数据通路中时序分配情况：

t1——由于 t1 有毛刺现象，且 t1 时序在控制器模块内部起作用（读出当前微地址对应的微命令）且为下降沿操作（可近似等同于 t2 上升沿），故不在数据通路为 t1 分配时序

t2——RAM

t3——DR1、DR2、AR、PC、R4、R5

t4——t4 时序在控制器内部地址转移逻辑起作用（地址重新映射），故不在数据通路中为其分配时序。

输入端 CLRCDU、ENCDU、CLKCDU 分别为外部计数器的清零端、使能端、时钟端，用于控制外部计数器产生外部输入。输入端 t2、t3 为控制器（时序电路）产生的时序，其余输入端信号为控制器输出的控制信号，用于控制数据通路电路行为。

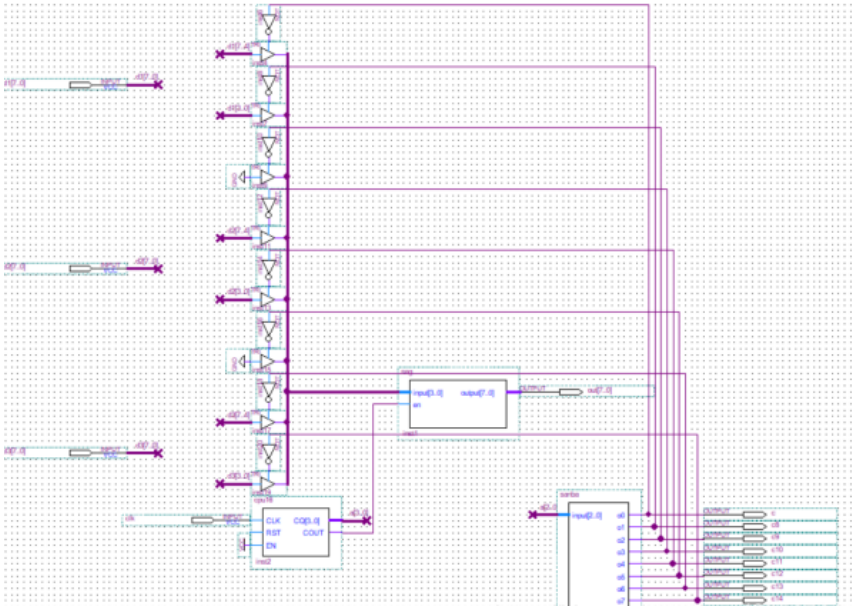
输出端 d[7..0]、AR[7..0]、INPUT[7..0]、R5[7..0]分别为数据通路中总线、AR、外部输入、R5 寄存器的即时值。

显示电路模块（用于下载时在实验箱上显示数据）

输入端 SHUCLK 为显示电路时钟端（与模型机时钟端不同），即为子模块模 8 计数器时钟端，其余输入为数据通路输出的总线、AR、外部输入、R5 寄存器的即时值，显示电路中用扫描数码管显示原理（数字系统数码管使用）用计数器通过译码器产生位选信号，并由位选信号选择输入端数据（总线、AR、外部输入、R5 寄存器）通过译码器译码成数码管段选信号并在对应数码管上显示输入端数据。其中输入端每 4 位显示到一位数码管（数码管每一位显示 16 进制 0-f，对应四位二进制数据输入），共需 8 位数码管。

输出端为每个时刻显示电路数码管对应的 8 位位选信号与 7 为段选信号。

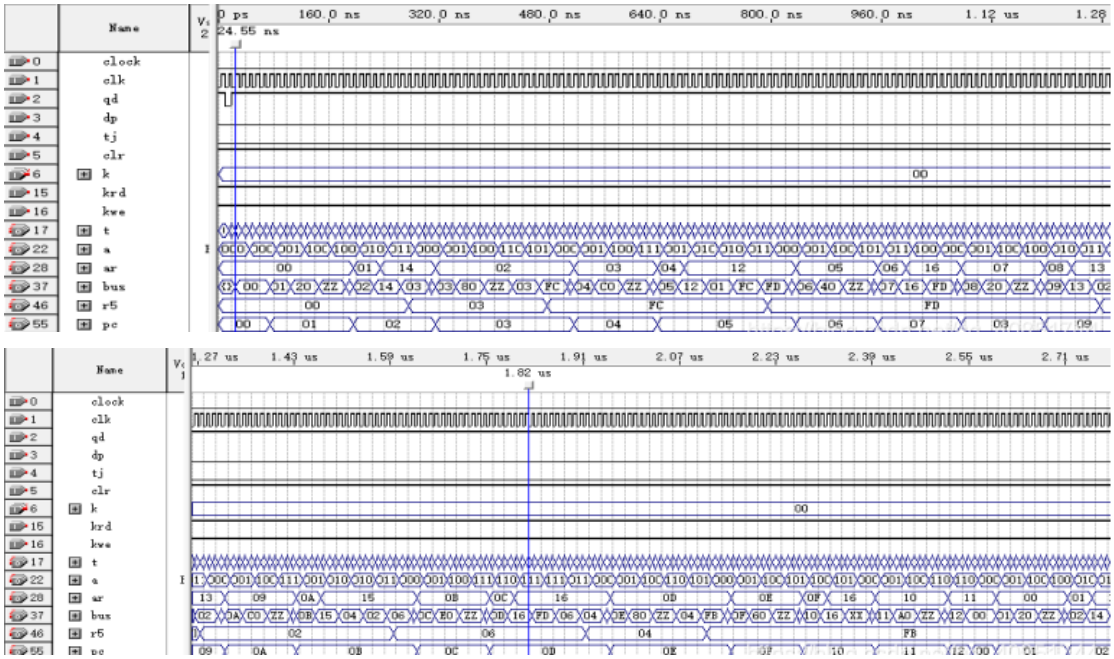
3.模型机附加电路（数码管显示）截图



文字说明

首先使用数码管动态扫描的思想，使用模 8 计数器模块在时钟沿的作用下往复循环产生 000-111 计数值并将其输入 3-8 译码器译码成 8 位段选信号。并通过 4 重 8 选 1 模块，使计数器计数输出作为控制选择端，使得每一时钟周期将对应当前位数码管的输入（总线、AR、外部输入、R5 寄存器的即时值）选择输出到段选译码器输入端。最后，段选译码器模块将 4 位二进制数据输入译码成对应十六进制输出的数码管七段码，从而显示对应位选信号的一位数码管，只要时钟频率够快，计数器计数频率加快，位选信号改变速度加快，就足以在人的肉眼产生视觉暂留效应从而肉眼观察 8 位数码管同时显示数据。

4.时序电路仿真波形图截图



功能说明

具体分析如下：

115ns~285ns 执行的是 LDA 指令，将 03H 给 r5 寄存器

先进行 01001 指令，将 pc 中的数值给 ar 寄存器，pc 再加 1，此时 ar 的值为 01H

再进行 10101 指令，将 ram 中 ar 位置的地址的数值取出来，再给 ar 寄存器，此时 ram 取出来的数值是 14H，ar 寄存器的数值即为 14H

再进行 10110 指令，将 ram 中 ar 位置的地址的数值取出来，再给 r5 寄存器，此时 ram 取出来的数值是 03H ，r5 寄存器的数值即为 03H

285ns~475ns 执行的是 COM 指令，将 03H 取反得到 FCH 再赋值给 r5 寄存器

先进行 01100 指令，将 r5 寄存器的数值放到 dr1 暂存器中，此时 dr1 暂存器的数值为 03H

再进行 11011 指令，将 dr1 中的数值进行取反操作后再赋值给 r5 寄存器，此时 r5 寄存器的数值为 FCH

475ns~795ns 执行的是 ADD 指令，相加后结果为 FDH 在赋值给 r5 寄存器

先进行 01110 指令，将 pc 中的数值给 ar 寄存器，pc 再加 1，此时 ar 的值为 04H

再进行 00011 指令，将 ram 中 ar 位置的地址的数值取出来，再给 ar 寄存器，此时 ran 取出来的数值是 12H，ar 寄存器的数值即为 12H

再进行 00100 指令，将 ram 中 ar 位置的地址的数值取出来，再给 dr2 暂存器，此时 ram 取出来的数值是 01H，dr2 寄存器的数值即为 01H

再进行 00101 指令，将 r5 寄存器的数值给 dr1 暂存器，此时 dr1 暂存器的数值为 FCH

再进行 00110 指令，将 dr1 和 dr2 的数值进行相加，再将结果给 r5 寄存器，此时 r5 寄存器的数值为 FDH

795ns~1045ns 执行的是 STA 指令，将 r5 寄存器中的值放到 ram 中

先进行 01010 指令，将 pc 中的数值给 ar 寄存器，pc 再加 1，此时 ar 的数值为 06H

再进行 10111 指令，将 ram 中 ar 位置的地址的数值取出来，再给 ar 寄存器，此时 ram 取出来的数值是 16H，ar 寄存器的数值即为 16H

再进行 11000 指令，将 r5 寄存器的数值放到 ram 中 ar 位置的地址
1045ns~1275ns 执行的是 LDA 指令，将 02H 给 r5 寄存器
先进行 01001 指令，将 pc 中的数值给 ar 寄存器，pc 再加 1，此时 ar 的值为 08H
再进行 10101 指令，将 ram 中 ar 位置的地址的数值取出来，再给 ar 寄存器，此时 ram 取出来的数值是 13H，ar 寄存器的数值即为 02H
再进行 10110 指令，将 ram 中 ar 位置的地址的数值取出来，再给 r5 寄存器，此时 ram 取出来的数值是 02H，r5 寄存器的数值即为 02H
1275ns~1595ns 执行的是 ADD 指令，相加后结果为 06H 再赋值给 r5 寄存器
先进行 01110 指令，将 pc 中的数值给 ar 寄存器，pc 再加 1，此时 ar 的值为 0AH
再进行 00011 指令，将 ram 中 ar 位置的地址的数值取出来，再给 ar 寄存器，此时 ran 取出来的数值是 15H，ar 寄存器的数值即为 15H
再进行 00100 指令，将 ram 中 ar 位置的地址的数值取出来，再给 dr2 暂存器，此时 ram 取出来的数值是 04H，dr2 寄存器的数值即为 04H
再进行 00101 指令，将 r5 寄存器的数值给 dr1 暂存器，此时 dr1 暂存器的数值为 02H
再进行 00110 指令，将 dr1 和 dr2 的数值进行相加，再将结果给 r5 寄存器，此时 r5 寄存器的数值为 06H
1595ns~1915ns 执行的是 AND 指令，相与后结果为 04H 再赋值给 r5 寄存器
先进行 01110 指令，将 pc 中的数值给 ar 寄存器，pc 再加 1，此时 ar 的值为 0CH
再进行 11101 指令，将 ram 中 ar 位置的地址的数值取出来，再给 ar 寄存器，此时 ram 取出来的数值是 16H，ar 寄存器的数值即为 16H
再进行 11110 指令，将 ram 中 ar 位置的地址的数值取出来，再给 dr2 暂存器，此时 ram 取出来的数值是 FDH，dr2 暂存器的数值即为 FDH
再进行 11111 指令，将 r5 寄存器的数值给 dr1 暂存器，此时 dr1 暂存器的数值即为 06H
再进行 00111 指令，将 dr1 和 dr2 的数值进行相与，再将结果给 r5 寄存器，此时 r5 寄存器的数值为 04H
1915ns~2115ns 执行的是 COM 指令，将 04H 取反得到 FBH 再赋值给 r5 寄存器
先进行 01100 指令，将 r5 寄存器的数值放到 dr1 暂存器中，此时 dr1 暂存器的数值为 04H
再进行 11011 指令，将 dr1 中的数值进行取反操作后再赋值给 r5 寄存器，此时 r5 寄存器的数值为 FBH
2115ns~2365ns 执行的是 OUT 指令，将存储器 16H 位置的值展示到 bus 上
先进行 01011 指令，将 pc 中的数值给 ar 寄存器，pc 再加 1，此时 ar 的数值为 0FH
再进行 11001 指令，将 ram 中 ar 位置的地址的数值取出来，再给 ar 寄存器，此时 ram 取出来的数值是 16H
再进行 11010 指令，将 ram 中 ar 位置的地址的数值取出来，再发送到总线，此时总线上的数值为 FDH
2365ns~2565ns 执行的是 JMP 指令，跳转回 00H 位置的指令重复执行
先进行 01101 指令，将 pc 中的数值给 ar 寄存器，pc 再加 1，此时 ar 的值为 11H
再进行 11100 指令，将 ram 中 ar 位置的地址的数值取出来，再给 pc 计数器，此时 ram 取出来的数值是 00H，pc 计数器的数值即为 00H.

5.Mif 文件结果（书上的程序）

Addr	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0a	+0b	+0c	+0d	+0e	+0f	+10	+11
00	20	14	80	C0	12	40	16	20	13	C0	15	E0	16	80	60	16	A0	00
12	01	02	03	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00

二、实验报告

1.假如控制器中模拟指令码得 IR7IR6IR5 对应到数据总线 D[7..0]的 D4D6D1,译码出 7 条机器指令的指令码?

机器指令	指令码	二进制指令码	十六进制指令码
LDA	001	00000001	01H
STA	010	01000000	40H
ADD	110	01010000	50H
AND	111	01010001	51H
COM	100	00010000	10H
OUT	011	01000001	41H
JMP	101	00010001	11H

2.用模型机已有的 7 条机器指令编写测试程序，完成复合运算：
not(((not(C))加 A) and (B 加 D)) 其中 A=01,B=02,C=03,D=04
MIF 文件结果截图

Addr	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0a	+0b	+0c	+0d	+0e	+0f	+10	+11
00	20	14	80	C0	12	40	16	20	13	C0	15	E0	16	80	60	16	A0	00
12	01	02	03	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00

文字说明

RAM地址	内容	说明
00H	20H	LDA双字节指令
01H	14H	LDA 14将地址14H中内容送到r5（r5=03H）
02H	80H	COM单字节指令，将r5取反送到r5（r5=FCH）
03H	C0H	ADD双字节指令
04H	12H	ADD 12将地址12H中的内容与r5相加送到r5（r5=FDH）
05H	40H	STA双字节指令
06H	16H	STA 16将r5的值放到地址16H的单元
07H	20H	LDA双字节指令
08H	13H	LDA 13将地址13H中内容送到r5（r5=02H）
09H	C0H	ADD双字节指令
0AH	15H	ADD 15将地址15H中的内容与r5相加送到r5（r5=06H）
0BH	E0H	AND双字节指令
0CH	16H	AND 16将地址16H中的内容与r5相与送到r5（r5=04H）
0DH	80H	COM单字节指令，将r5取反送到r5（r5=FBH）
0EH	60H	OUT双字节指令
0FH	16H	OUT 16将地址16H中的内容送到BUS
10H	A0H	JMP双字节指令
11H	00H	JMP 00无条件转移到00H地址
12H	01H	数据a
13H	02H	数据b
14H	03H	数据c
15H	04H	数据d
16H	00H	数据暂存

not((not(c))加 a)and(b 加 d))
其中 a=01H，b=02H，c=03H，d=04H
先将数据 c 放入 r5 寄存器中，再进行 not(c)，将结果放在 r5 寄存器中，
再进行 not(c)加 a，将结果放在 r5 寄存器中，将结果放到存储器中，
再将数据 b 放入 r5 寄存器中，再进行(b 加 d)，结果放到 r5 寄存器中，
再将存储器中存储的 not(c)加 a 取出，与 r5 进行相与，将结果放到 r5 寄存器中，
再对 r5 寄存器进行取反操作，结果放到 r5 寄存器中，
此时 r5 寄存器中的值就是我们要进行的复合运算的最终值。
管脚分配

	Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Group	Current Strength	PCB layer
12	clock	Input	PIN_17	1	B1_N0	3.3-V LVTTL (default)			24mA (default)	
13	qd	Input	PIN_43	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
14	kwe	Input	PIN_44	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
15	krd	Input	PIN_45	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
16	clr	Input	PIN_47	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
17	dp	Input	PIN_48	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
18	tj	Input	PIN_53	4	B4_N1	3.3-V LVTTL (default)			24mA (default)	
19	k[0]	Input	PIN_60	4	B4_N0	3.3-V LVTTL (default)		k[7..0]	24mA (default)	
20	k[1]	Input	PIN_63	4	B4_N0	3.3-V LVTTL (default)		k[7..0]	24mA (default)	
21	k[2]	Input	PIN_64	4	B4_N0	3.3-V LVTTL (default)		k[7..0]	24mA (default)	
22	k[3]	Input	PIN_65	4	B4_N0	3.3-V LVTTL (default)		k[7..0]	24mA (default)	
23	k[4]	Input	PIN_67	4	B4_N0	3.3-V LVTTL (default)		k[7..0]	24mA (default)	
24	k[5]	Input	PIN_69	4	B4_N0	3.3-V LVTTL (default)		k[7..0]	24mA (default)	
25	k[6]	Input	PIN_70	4	B4_N0	3.3-V LVTTL (default)		k[7..0]	24mA (default)	
26	k[7]	Input	PIN_71	4	B4_N0	3.3-V LVTTL (default)		k[7..0]	24mA (default)	
27	x[0]	Unknown	PIN_72	4	B4_N0	3.3-V LVTTL (default)			24mA (default)	
28	x[1]	Unknown	PIN_73	3	B3_N1	3.3-V LVTTL (default)			24mA (default)	
29	x[2]	Unknown	PIN_74	3	B3_N1	3.3-V LVTTL (default)			24mA (default)	
30	x[3]	Unknown	PIN_75	3	B3_N1	3.3-V LVTTL (default)			24mA (default)	
31	x[4]	Unknown	PIN_79	3	B3_N1	3.3-V LVTTL (default)			24mA (default)	
32	x[5]	Unknown	PIN_80	3	B3_N1	3.3-V LVTTL (default)			24mA (default)	
33	x[6]	Unknown	PIN_81	3	B3_N1	3.3-V LVTTL (default)			24mA (default)	
34	x[7]	Unknown	PIN_86	3	B3_N1	3.3-V LVTTL (default)			24mA (default)	
35	ir5	Output	PIN_87	3	B3_N1	3.3-V LVTTL (default)			24mA (default)	
36	clk	Input	PIN_91	3	B3_N0	3.3-V LVTTL (default)			24mA (default)	
37	ir6	Output	PIN_92	3	B3_N0	3.3-V LVTTL (default)			24mA (default)	
38	ir7	Output	PIN_93	3	B3_N0	3.3-V LVTTL (default)			24mA (default)	
39	a[0]	Output	PIN_94	3	B3_N0	3.3-V LVTTL (default)		a[4..0]	24mA (default)	
40	a[1]	Output	PIN_96	3	B3_N0	3.3-V LVTTL (default)		a[4..0]	24mA (default)	
41	a[2]	Output	PIN_97	3	B3_N0	3.3-V LVTTL (default)		a[4..0]	24mA (default)	
42	a[3]	Output	PIN_99	3	B3_N0	3.3-V LVTTL (default)		a[4..0]	24mA (default)	
43	a[4]	Output	PIN_100	3	B3_N0	3.3-V LVTTL (default)		a[4..0]	24mA (default)	
44	t[4]	Output	PIN_101	3	B3_N0	3.3-V LVTTL (default)		t[4..1]	24mA (default)	
45	t[3]	Output	PIN_103	3	B3_N0	3.3-V LVTTL (default)		t[4..1]	24mA (default)	
46	t[2]	Output	PIN_104	3	B3_N0	3.3-V LVTTL (default)		t[4..1]	24mA (default)	
47	t[1]	Output	PIN_112	2	B2_N0	3.3-V LVTTL (default)		t[4..1]	24mA (default)	
48	c21	Output	PIN_119	2	B2_N0	3.3-V LVTTL (default)			24mA (default)	
49	c20	Output	PIN_120	2	B2_N0	3.3-V LVTTL (default)			24mA (default)	
50	c19	Output	PIN_121	2	B2_N0	3.3-V LVTTL (default)			24mA (default)	
51	c18	Output	PIN_122	2	B2_N0	3.3-V LVTTL (default)			24mA (default)	

文字说明

芯片还是选取 ep5t144c8 的芯片，因为我们的芯片就是这个，仅支持这个。

将 qd 模型机启动控制信号，clr 清零控制信号，kwe 强读控制信号和 krd 强写控制信号放在了按钮开关，因为他们的初始值是 1；

将 k 八位输入数据，tj 停机信号，dp 单拍进行控制信号放在了拨码开关，因为他们的初始值是 0，而且这样可以方便我进行数值设置。

通过数码管显示 ar 寄存器，bus 总线和 r5 寄存器的数值。

通过二极管 101-104 来显示 t1-t4 四个节拍脉冲信号。二极管 93,92,87 分别代表 ir7，ir6，ir5，为 ir 寄存器输出的数值。

给模型机的时钟信号是 91 时钟信号，频率为 100hz，便于观测；

给显示数码管的时钟信号是 17 时钟信号，以此来消除频闪。

针脚设计好之后点击全编译，生成 sof 文件。

通过 programer 工具将 sof 文件加在到板子上。

下载操作截图

指令　　图示（由左到右依次是 ar-bus-r5）

初始状态

LDA 转移指令，r5=03H

COM 取反指令，r5=FCH

ADD 相加指令，r5=FDH

STA 存储指令，bus=FDH 

LDA 转移指令，r5=02H 

ADD 相加指令。r5=06H 

AND 相与指令，r5=04H 

COM 取反指令，r5=FBH 

OUT 输出指令，bus=00H 

JMP 跳转指令，ar=00H 

文字说明：

由此可以看出，在 JMP 指令执行后，r5 寄存器中的值就是我们所要的复合运算的值 FB。

MIF 文件书写注意：7 条指令均需用到；程序段的地址从 00H 地址开始书写，数据段可从程序段写完的下一地址开始书写，或稍靠后的地址段开始书写。

3.阅读教材 P257-263,回答：

Microcomputer.vhd 代码中进程 ct1、ct2、ct3、ct4 功能划分依据是什么？

ct1 的功能是微序列控制器下址跳转；

ct2 的功能是实现各种指令，主要集中在实现从存储器或寄存器释放数据到总线上；

ct3 的功能是完成各种指令，从总线上装载数据到相应的存储器或寄存器中；

ct4 的功能是生成下址。

Microcomputer.vhd 代码中如何定义并初始化 RAM?

通过定义一个名叫 ram8 的中间变量进行定义，并通过赋初值来进行初始化：

signalram8:RAM:=(x”20”,x”0d”,x”c0”,x”0e”,x”40”,x”10”,x”60”,x”10”,x”e0”,x”0f”,x”80”,x”a0”,x”00”,x”55”,x”8a”,x”f0”,x”ff”,others=>x”00”)

其中括号中的数据是放入的初始值，即原本 mif 文件中的数据。

Microcomputer.vhd 代码中 bus_reg_t2<=ram8(conv_integer(ar)) 与 ram8(conv_integer(ar))<=r5 的含义是什么？

第一句是将 ram 中 ar 地址位置的数据值传送到总线上；

第二句是将 r5 寄存器中的数据值传送到 ram 中 ar 地址的位置。

4.VHDL 语言中如果考虑多个时钟信号的情况？

由于 vhdl 中是通过 process 进程进行设计的，所以只需要在进程中对不同的时钟信号进行条件选择，就可以在不同的时钟信号执行不同的操作。
但是注意要考虑到时序上的整合问题。
比方说，本次试验的时候，可以有四个不同的时钟信号，但是一定要满足：
在最先到达的一个时钟的波峰进行微序列控制器的下址跳转；
在第二个到达的波峰进行取数据的功能，即将数据从寄存器或存储器中取出来放到总线上，例如从 ram 中取出运算数据发送到总线上；
在第三个到达的波峰进行存数据的功能，即将数据从总线上存到寄存器或存储器中，例如将总线上的运算结果存回到 r5 寄存器中；
在最后一个到达的波峰进行下一个下址的生成。
虽然我是通过 bdf 和 bsf 的元件拼接整合实现的，但在我的整合过程中也注意了时序上的整合问题，比方说我给当前下址寄存器的时钟信号是 t2 时序脉冲，给寄存器和存储器的时钟信号一般是 t3 时序脉冲，给 pp（我自己设计的下一下址寄存器）元件的时钟信号是 t4 时序脉冲，通过这样才达到了整个电路的正常运行。

5.模型机与控制器实验总结

“全面深化改革是一项复杂的系统工程，需要加强顶层设计和整体谋划，加强各项改革关联性、系统性、可行性研究”，习主席在党外人士座谈会指出的这句话，结合本次模型机实验的经历，犹如醍醐灌顶般给了我莫大的启发与共鸣。
作为计算机系统原理实验第一部分的最后一个大综合实验，本次实验难度系数高，实验复杂，涉及了开学五个周以来所学习的所有的知识点，而且不仅仅是简单的数据规模以及模块元件上的整合，更重要的是时钟脉冲信号上的整合，通过对 t1~t4 这四个时钟节拍的合理调用，来调度整个电路的正常运行。
总的来说，在昨晚之前的六个实验之后，模拟机各个组件的实现已经不再是十分困难的存在了，困难的是如何将所有的原件进行组织上和时序上的整合，一旦有什么不对，结果就很容易出差错。我基本上是通过书上的大部分内容进行设计的，在最后加上了少部分自己的一些元件用来确保整个电路的正常进行。
在整合过程中，各子模块的关联性、系统性也是核心所在，子模块的输入、输出相连关系需要确保正确、各子模块的功能以及相互之间的关联性需要明确、模型机系统的数据在各子模块的流动需要在时序的配合下有序的进行从而有条不紊地执行程序、不造成冲突确保系统性。这是模型机实验的核心所在，同样也是难点所在，如何正确的分配时序从而保证系统有条不紊执行让我困惑了许久，最好的办法是以系统思维、站在模型机顶层的视角对待各个子模块确定它们需要实现的功能以及实现的时机，从而正确分配时序以及确保各子模块的关联性，这不得不说又是系统思维在本次实验中的又一发光点。