计算机组成与系统结构实验报告

班级 F0203306

组号 7

完成日期 2005年5月10日

一 总线传输实验

1. 实验原理

本实验中, M, BUF位于实验仪器内, M为6116RAM, IAB10..IAB0为其地址线(IAB10应置0), RC为有效读信号, WC为有效写信号(均为0有效), BUF为74245, 在按下实验仪器控制台的STEP键后，LED数码管的小数点全亮，这是RF=0，允许74245，DIR控制74245导通方向: 0为A->B, 1为B->A. IDB7..IDB0为内部总线，接L15..L8来显示IDB的数据.

实验框图上的以下部件由abel-hdl描述模拟:

74244为8位三态门，OE=0时，把K7..K0的数据输入到IDB上.

74377为8位D触发器，CK为上跳有效时钟，EN为允许输入(恒接0), 它的输出接L15..L8.

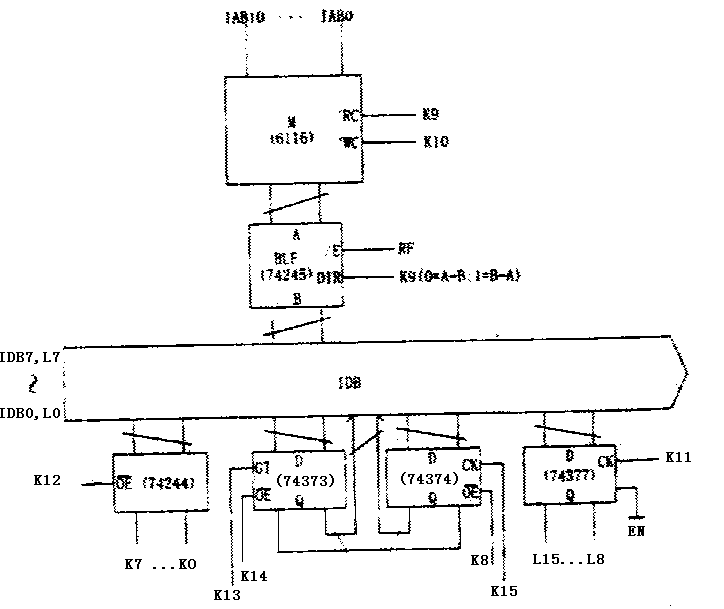
74373位8位带三态透明锁存器，GT为接数门控端，OE为输出控制，OE=0时锁存器输出至IDB.

74374为8位D触发器，CK为电平上跳有效接数时钟，OE为输出控制，OE=0时输出至IDB.

2. 实验要求

把两个数据分别写入74373和74374中，再使用RAM作为中间但原来交换这两个数据.

3. 实验框图



4. 实验程序

MODULE BUSES

IAB10 PIN 45;

IAB9 PIN 44;

IAB8 PIN 43;

IAB7 PIN 42;

IAB6 PIN 41;

IAB5 PIN 40;

IAB4 PIN 39;

IAB3 PIN 38;

IAB2 PIN 37;

IAB1 PIN 36;

IAB0 PIN 35;

RC PIN 34;

WC PIN 32;

B2 PIN 31;

IDB0 PIN 30;

IDB1 PIN 29;

IDB2 PIN 28;

IDB3 PIN 27;

IDB4 PIN 26;

IDB5 PIN 25;

IDB6 PIN 24;

IDB7 PIN 23;

K15 PIN 128;

K14 PIN 126;

K13 PIN 124;

K12 PIN 123;

K11 PIN 122;

K10 PIN 119;

K9 PIN 117;

K8 PIN 108;

K7 PIN 105;

K6 PIN 103;

K5 PIN 99;

K4 PIN 2;

K3 PIN 127;

K2 PIN 125;

K1 PIN 121;

K0 PIN 120;

K23 PIN 118;

K22 PIN 109;

K21 PIN 107;

K20 PIN 106;

K19 PIN 104;

K18 PIN 102;

K17 PIN 101;

K16 PIN 100;

L15 PIN 98;

L14 PIN 95;

L13 PIN 93;

L12 PIN 91;

L11 PIN 89;

L10 PIN 87;

L9 PIN 85;

L8 PIN 76;

L7 PIN 74;

L6 PIN 72;

L5 PIN 70;

L4 PIN 68;

L3 PIN 96;

L2 PIN 94;

L1 PIN 92;

L0 PIN 90;

L23 PIN 88;

L22 PIN 86;

L21 PIN 77;

L20 PIN 75;

L19 PIN 73;

L18 PIN 71;

L17 PIN 69;

L16 PIN 67;

U244A0 NODE;

U244A1 NODE;

U244A2 NODE;

U244A3 NODE;

U244A4 NODE;

U244A5 NODE;

U244A6 NODE;

U244A7 NODE;

U244OE NODE;

U377Q0 NODE ISTYPE 'REG';

U377Q1 NODE ISTYPE 'REG';

U377Q2 NODE ISTYPE 'REG';

U377Q3 NODE ISTYPE 'REG';

U377Q4 NODE ISTYPE 'REG';

U377Q5 NODE ISTYPE 'REG';

U377Q6 NODE ISTYPE 'REG';

U377Q7 NODE ISTYPE 'REG';

U377CK NODE;

U377EN NODE;

U374Q0 NODE ISTYPE 'REG';

U374Q1 NODE ISTYPE 'REG';

U374Q2 NODE ISTYPE 'REG';

U374Q3 NODE ISTYPE 'REG';

U374Q4 NODE ISTYPE 'REG';

U374Q5 NODE ISTYPE 'REG';

U374Q6 NODE ISTYPE 'REG';

U374Q7 NODE ISTYPE 'REG';

U374CK NODE;

U374OE NODE;

U373Q0 NODE;

U373Q1 NODE;

U373Q2 NODE;

U373Q3 NODE;

U373Q4 NODE;

U373Q5 NODE;

U373Q6 NODE;

U373Q7 NODE;

U373GT NODE;

U373OE NODE;

EQUATIONS

RC = K9;

WC = K10;

[IAB0..IAB10] = 0;

B2 = K9;

[IDB0..IDB7]= (U244OE == 0)&[U244A0..U244A7]

#(U373OE == 0)&[U373Q0..U373Q7]

#(U374OE == 0)&[U374Q0..U374Q7];

[IDB0..IDB7].OE = !(U244OE&U373OE&U374OE);

[L0..L7] = [IDB0..IDB7];

[U244A0..U244A7] = [K0..K7];

U244OE = K12;

[L8..L15] = [U377Q0..U377Q7];

U377CK = K11;

U377EN = 0;

[U377Q0..U377Q7] := [IDB0..IDB7];

[U377Q0..U377Q7].CLK = U377CK#U377EN;

[U374Q0..U374Q7] := [IDB0..IDB7];

[U374Q0..U374Q7].CLK = U374CK;

U374CK = K15;

U374OE = K8;

[U373Q0..U373Q7] = (U373GT == 1)&[IDB0..IDB7]#(U373GT == 0)&[U373Q0..U373Q7];

U373GT = K13;

U373OE = K14;

END

5. 实验体会

通过本实验，我们了解了系统总线上传输数据的要点，包括总线与各个器件的连接（线或），总线上数据传输同一时刻只能有一个器件输出，而其他期间可以处于接数状态或断开状态。接数者必须在输出者停止输出前完成接数动作锁住总线上的数据。对于电平触发的器件，意味着输出停止前”关门”, 而对于脉冲触发的器件，意味着输出停止前时钟必须触发它的接数动作。实验所需要的3次总线传输依次进行，即可达到目的。

本实验还使我们学会了ABEL-HDL语言的基本使用以及其语义和在实验中所起的作用(软接线), 学会了equations的写法。

二 运算器部件实验

1. 实验原理

本实验中, IDB7..IDB0为内部总线，接L15..L8来显示IDB的数据.

实验框图上的以下部件由abel-hdl描述模拟:

74244为8位三态门，OE=0时，把K7..K0的数据输入到IDB上.

BUF为74244, 它接通至ALU的输出F端.

74377为8位D触发器，CK为上跳有效时钟，EN为允许输入(恒接0), 它的输出接ALU的A端.

74373位8位带三态透明锁存器，GT为接数门控端，OE为输出控制，OE=0时锁存器输出至ALU的B端.

ALU为8位逻辑运算部件，它的输出为F0~F8(其中F8为进位输出). 它可有如下8种功能，具体由开关K13..K15选择：

F = A + B 允许有进位输入(K9)和进位输出(L9)

F = A + 1 允许有进位输入(K9)和进位输出(L9)

F = A & B

F = A | B

F = A

F = B

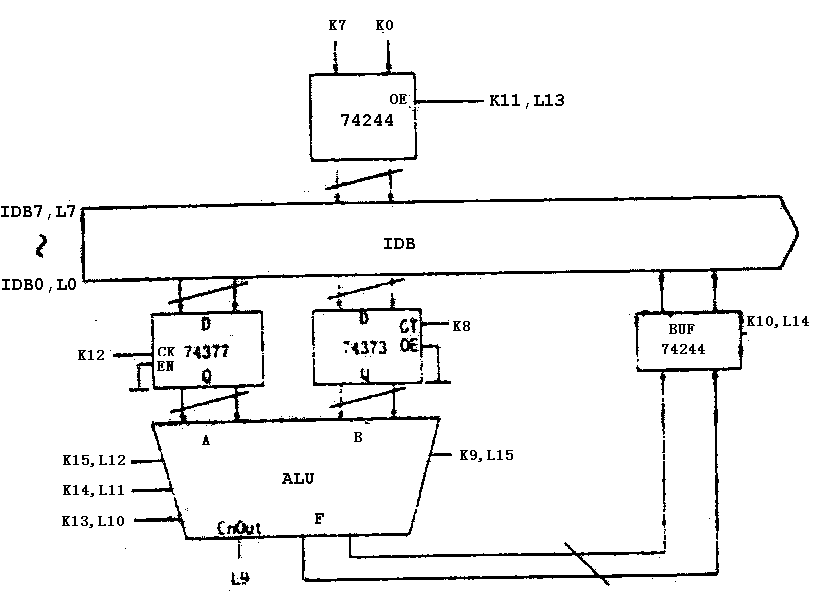
F = 0xFF

F = 0x00

2. 实验要求

设计一个简单的运算器模块，它包括寄存器、运算器ALU, 数据输入（开关）和数据输出（LED）通道。

3. 实验框图



4. 实验程序

MODULE alu

IAB10 PIN 45;

IAB9 PIN 44;

IAB8 PIN 43;

IAB7 PIN 42;

IAB6 PIN 41;

IAB5 PIN 40;

IAB4 PIN 39;

IAB3 PIN 38;

IAB2 PIN 37;

IAB1 PIN 36;

IAB0 PIN 35;

RC PIN 34;

WC PIN 32;

B2 PIN 31;

IDB0 PIN 30;

IDB1 PIN 29;

IDB2 PIN 28;

IDB3 PIN 27;

IDB4 PIN 26;

IDB5 PIN 25;

IDB6 PIN 24;

IDB7 PIN 23;

K15 PIN 128;

K14 PIN 126;

K13 PIN 124;

K12 PIN 123;

K11 PIN 122;

K10 PIN 119;

K9 PIN 117;

K8 PIN 108;

K7 PIN 105;

K6 PIN 103;

K5 PIN 99;

K4 PIN 2;

K3 PIN 127;

K2 PIN 125;

K1 PIN 121;

K0 PIN 120;

K23 PIN 118;

K22 PIN 109;

K21 PIN 107;

K20 PIN 106;

K19 PIN 104;

K18 PIN 102;

K17 PIN 101;

K16 PIN 100;

L15 PIN 98;

L14 PIN 95;

L13 PIN 93;

L12 PIN 91;

L11 PIN 89;

L10 PIN 87;

L9 PIN 85;

L8 PIN 76;

L7 PIN 74;

L6 PIN 72;

L5 PIN 70;

L4 PIN 68;

L3 PIN 96;

L2 PIN 94;

L1 PIN 92;

L0 PIN 90;

L23 PIN 88;

L22 PIN 86;

L21 PIN 77;

L20 PIN 75;

L19 PIN 73;

L18 PIN 71;

L17 PIN 69;

L16 PIN 67;

U244A0 NODE;

U244A1 NODE;

U244A2 NODE;

U244A3 NODE;

U244A4 NODE;

U244A5 NODE;

U244A6 NODE;

U244A7 NODE;

U244OE NODE;

U377Q0 NODE ISTYPE 'REG';

U377Q1 NODE ISTYPE 'REG';

U377Q2 NODE ISTYPE 'REG';

U377Q3 NODE ISTYPE 'REG';

U377Q4 NODE ISTYPE 'REG';

U377Q5 NODE ISTYPE 'REG';

U377Q6 NODE ISTYPE 'REG';

U377Q7 NODE ISTYPE 'REG';

U377CK NODE;

U377EN NODE;

U373Q0 NODE;

U373Q1 NODE;

U373Q2 NODE;

U373Q3 NODE;

U373Q4 NODE;

U373Q5 NODE;

U373Q6 NODE;

U373Q7 NODE;

U373GT NODE;

U373OE NODE;

B244A0 NODE;

B244A1 NODE;

B244A2 NODE;

B244A3 NODE;

B244A4 NODE;

B244A5 NODE;

B244A6 NODE;

B244A7 NODE;

B244OE NODE;

F0 NODE;

F1 NODE;

F2 NODE;

F3 NODE;

F4 NODE;

F5 NODE;

F6 NODE;

F7 NODE;

F8 NODE;

FT NODE;

EQUATIONS

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*U244\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[U244A0..U244A7] = [K0..K7];

U244OE = K11;

L13 = K11;

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*U377\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

U377CK = K12;

U377EN = 0;

[U377Q0..U377Q7]:=[IDB0..IDB7];

[U377Q0..U377Q7].CLK = U377CK # U377EN;

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*U373\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[U373Q0..U373Q7]=(U373GT == 1)&[IDB0..IDB7]#(U373GT == 0)&[U373Q0..U373Q7];

U373GT = K8;

U373OE = 0;

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*B244\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[B244A0..B244A7] = [F0..F7];

B244OE = K10;

L14 = K10;

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*BUS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[IDB0..IDB7] = (U244OE == 0)&[U244A0..U244A7]

#(B244OE == 0)&[B244A0..B244A7];

[IDB0..IDB7].OE = !(U244OE&B244OE);

[L0..L7] = [IDB0..IDB7];

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ALU\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[FT,F3..F0]= ([K15,K14,K13]==0)&([0,U377Q3..U377Q0]+[0,U373Q3..U373Q0]+[0,0,0,0,K9]);

[L9,F7..F4]= ([K15,K14,K13]==0)&([0,U377Q7..U377Q4]+[0,U373Q7..U373Q4]+[0,0,0,0,FT]);

[FT,F3..F0]= ([K15,K14,K13]==1)&([0,U377Q3..U377Q0]+[0,0,0,0,1]+[0,0,0,0,K9]);

[L9,F7..F4]= ([K15,K14,K13]==1)&([0,U377Q7..U377Q4]+[0,0,0,0,0]+[0,0,0,0,FT]);

[F7..F0] = ([K15,K14,K13]==2)&([U377Q7..U377Q0]&[U373Q7..U373Q0]);

[F7..F0] = ([K15,K14,K13]==3)&([U377Q7..U377Q0]#[U373Q7..U373Q0]);

[F7..F0] = ([K15,K14,K13]==4)&[U377Q7..U377Q0];

[F7..F0] = ([K15,K14,K13]==5)&[U373Q7..U373Q0];

[F7..F0] = ([K15,K14,K13]==6)&[1,1,1,1,1,1,1,1];

[F7..F0] = ([K15,K14,K13]==7)&[0,0,0,0,0,0,0,0];

END

5. 实验体会

通过本实验，我们了解了ALU的功能以及如何用able-hdl语言来实现它的功能。并且本实验用到了上次实验中的BUS, 所以对BUS的认识进一步巩固。

三 整机实验

1. 设计要求

(1) 外设：本实验计算机采用PIO方式支持键盘，不支持打印机

(2) 实验计算机的结构：运算器采用单累加器多寄存器结构。

(3) 实验计算机的功能：可以对键盘的两个2位10进制进行乘法运算

(4) 实验计算机指令系统规模：

实验计算机共有16条指令，包括：add, sub, mov, dec, lda, sta, rrc, jmp(系列), halt等，其中mov指令支持及存器直接、寄存器间接和立即数寻址。

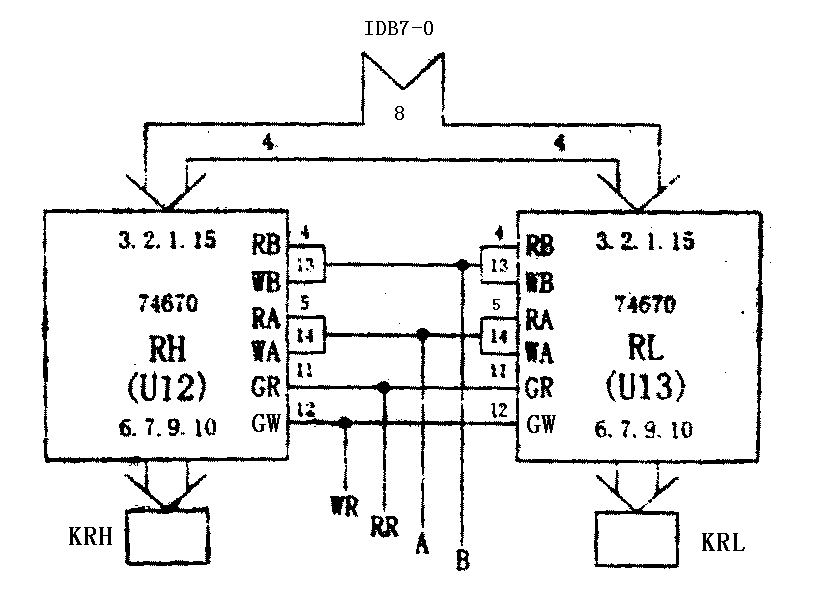
(5) 微操作控制信号以指令的[I7, I6, I5, I2, 1, 1]作为控存入口地址，每个指令微程序入口地址间间隔为4. 控存的顺序控制方式为基数增量方法。

2. 实验原理 (各model简介)

(1) 寄存器模块

寄存器模块由4个8位通用寄存器组成，使用两片74670实现。寄存器可以从总线接数，但只能输出到ALU的B端. 寄存器由A, B, RR, WR来控制读写，其中AB决定选中哪个寄存器，然后由RR和WR来决定进行的操作。RR = 0表示读寄存器，WR = 0表示写入，RR = WR = 1表示保持。

寄存器堆模块逻辑框图



(2) 总线模块

总线模块分数据总线和地址总线。

地址总线为11位，内部地址总线IAB和外部地址总线OAB通过一个单向门控制，并在本机中常开。所以对于地址总线，IAB和OAB没有太多区别。

IR1低3位及IR2通过一个三态传输门挂在IAB上，当指令需要输出地址到OAB时便打开此门。

PC也通过一个三态传输门挂在IAB上，用于指令的fetch过程。

数据总线为8位，内部数据总线IDB和外部数据总线ODB通过一个三态双向传输门连接。门常有效，但通过B2来控制开放方向。内存模块以及外设模块（键盘）接在ODB上，所以当读内存或外设时应允许ODB向IDB输出，其余时间只要IDB向ODB输出即可。

累加器和寄存器堆ro..r3以及IR1, IR2两个内部寄存器作为接受者挂在IDB上，可以从IDB上接数。

ALU通过一个74245挂在IDB上，可以输出其结果。

IO查询口则直接挂在IDB上。其输出由IAB和RC控制：只有读取地址0x404的时候才输出。

(3) 运算器模块

运算器模块包括累加器74198, 它的全0指示信号ZD, 辅助寄存器ACT和TMP，两个74181组成ALU，以及输出口BUFFER 74254, 选择CY的74153, 和CY寄存器7474.

累加器74198可以进行移位操作，与74153和7474(CY)配合可以实现循环右移。其余运算操作在ALU中完成，包括add, sub, dec, and.

运算器模块符号说明：

KAH、KAL 累加器A的输入选择开关 置左，输入来自IDB

（简称KA） 置右，输入来自ALU

KBH、KBL 缓冲器BUF的输入选择开关 置左，输入来自A

（简称KB） 置右，输入来自ALU

KCH、KCL 暂存器ACT的输入选择开关 置左，输入来自IDB

（简称KC） 置右，输入来自A

CG ACT的接数控制电平 低电平有效

CC ACT的接数控制脉冲 电平正跳有效

CT TMP的接数控制 高电平有效

OT TMP的输出控制 低电平有效

OB BUFFER的输出控制 低电平有效

CA A的工作脉冲 电平正跳有效

X0、X1 A的工作方式选择（据74198功能）；

SL，SR 分别为A的右移入、左移入

A7，A0 分别为A的最高位和最低为输出

SA，SB 进位输入选择 （见74153器件介绍）

P2，P1，P0 进位输入源 高电平有效

CP 进位触发器的接数脉冲 电平正跳有效

CY 进位触发器输出

ZC 为零触发器的接数脉冲 电平正跳有效

ZD 为零触发器的数据输入 ZD=”1”表示累加器A为全零

Z 为零触发器输出

算术逻辑ALU是由两片74181构成，它是运算器的核心。它可以对两个8位二进制数进行多种算术或逻辑运算。两个参加运算的数分别来自ACT和TMP（或Ri），运算结果可以直接送到累加器或经BUFFER送到累加器，以便进行移位操作或参加下次运算。

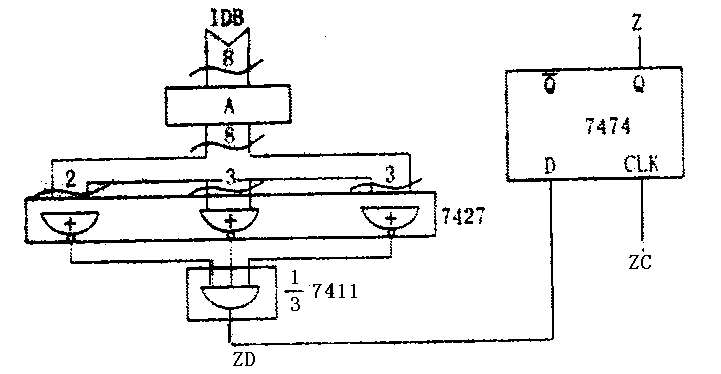
ACT采用74377，CG为低电平且接数控制脉冲CC电平正跳时，ACT接数。ACT的输出不受控制地直接加在ALU的A组输入端参加运算。

TMP采用三态输出锁存器74373。当它的接数控制端CT为高电平时，接收内部数据总线上的信息；当它的输出控制端OT为低电平时，其所存信息加到ALU的B组输入端参加运算。

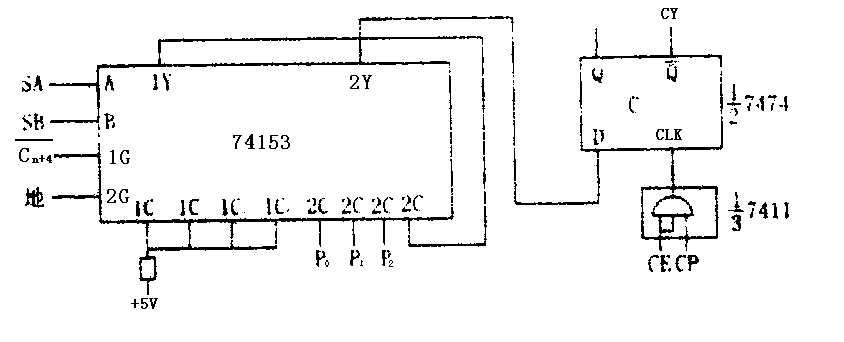
BUFFER采用三态传输器件74245，由OB信号控制，OB为”0”时，BUFFER开通，此时其输出等于其输入；当OB为”1”时，BUFFER不同，此时其输出呈高阻。

累加器A采用74198，它具有并行接数、左移、右移、保持功能，具体由X0、X1、SR、SL决定。CA是它的工作脉冲，正跳变有效。

累加器A判零线路



进位发生线路



(4) 内存模块

内存模块由6116RAM组成，内存大小为2k, 但由于和IO口共享内存地址且空出a10作为访问IO设备的indicator, 所以实际可用的内存为1k. 内存模块挂接在ODB上进行数据传输，地址由OAB提供，读取/写入由RC和WC控制.

内存模块符号说明：

RC 运行时CPU读内存 低电平有效，宜采用负脉冲

WC 运行时CPU写内存 低电平有效，宜采用负脉冲

当要对6116进行读操作时，应使选片端CS为低电平，读控制端R为低电平，写控制W为高电平，此时ODB上的信息为从6116中地址由OAB9-OAB0决定的单元读出的内容；当要对6116进行写操作时，应使选片端CS为低电平，写控制W为低电平，读控制端R为高电平，此时ODB上的信息被写入6116中由OAB9-OAB0决定的地址单元中。

为了能使外设与内存统一编址，统一操作指令，设置OAB10，以区别OAB9-OAB0使外设地址还是内存地址，当OAB10=”1”时，OAB9-OAB0为外设地址；当OAB10=”0”时，OAB9-OAB0为内存地址。

(5) 控制模块

控制模块由辅助寄存器IR1和IR2, 3片74163用来存放PC, 可输出其中的地址至IAB.

控制模块符号说明：

GI IR1的接数控制电平 低电平有效

CI IR1的接数控制脉冲 正跳变有效

CL IR2的接数控制脉冲 正跳变有效

OI IR1、IR2输出到IAB的控制电平 低电平有效

P+1 PC的加1计数控制电平 高电平有效

CLR PC的清零控制电平 低电平有效

LP PC的接数控制电平 高电平有效

CK PC的工作脉冲 负跳变有效

PCO PC输出到IAB的控制电平 低电平有效

I7-I0 IR1的状态引出，供引用

指令寄存器共16位，分IR1和IR2，用来接收和寄存内部数据总线IDB上的指令信息。IR1采用74377，IR2采用74374，它们都是8位的寄存器，但工作方式不同。当IR1的接数控制端GI为低电平且它的接数控制脉冲端CI出现电平正跳变时接数，它的输出不受控制，作为I7-I0被直接引出，以供使用。

IR2是三态输出的8位寄存器。当它的接数控制脉冲端CL出现电平正跳变时接数，当它的输出控制OI为低电平输出，OI为高电平时IR2输出呈高阻。

(6) 微控制模块

为控制模块由MPC, CM, MIR组成，MPC的地址MD5..MD0接IR1的I7, I6, I5, I2, 1, 1, 其余地址线接0. 控存输出按照软接线表接到各个控制信号处控制整个计算机的运转.

微程序控制模块符号说明：

MCLR MPC清零控制 低电平有效

MCLK MPC工作时钟 电平正跳有效

MLD MPC接数控制 低电平有效

MP+1 MPC加1计数控制 高电平有效

MD10-0 MPC输入，决定微程序首址，可由指令操作码编码产生

MIG MIR的接数控制 低电平有效

M23-0 MIR的输出，供引用

停机时（RF=1）可由控制台对控存进行读或写，微地址由OAB经74157提供。MF为内存/控存选择开关M/CM的状态信号，开关置右时，MF=0，控制台可读/写控存。

运行时（RF=0）控存的工作与M/CM开关状态MF无关，此时控存处于只读不写状态。控存中读出的微指令一方面作为信号CM23-CM0送控制台显示，另一方面可寄存到微指令寄存器MIR中。

微程序计数器MPC由三片4位二进制同步计数器74161组成，它具有清零、计数、接数和保持功能。

(7) 外设模块

外设模块包括键盘和IO查询口键盘数据输出到ODB, IO查询口则输出到IDB, 也由RC信号控制其读取.

KA 键盘设备选择 高电平有效

KB 键盘状态 KB为”0”表示有键入，复位后或数据已被取走时KB=1

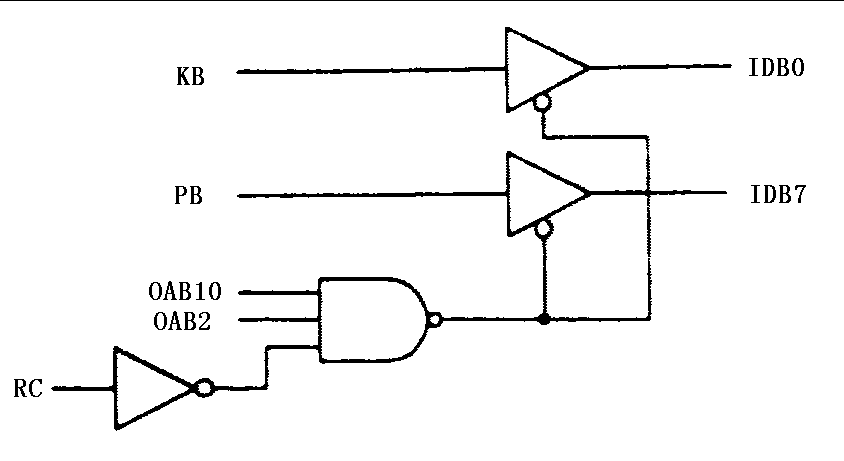
RO 运行时RO为”1”

RC CPU读寄存器 低电平有效

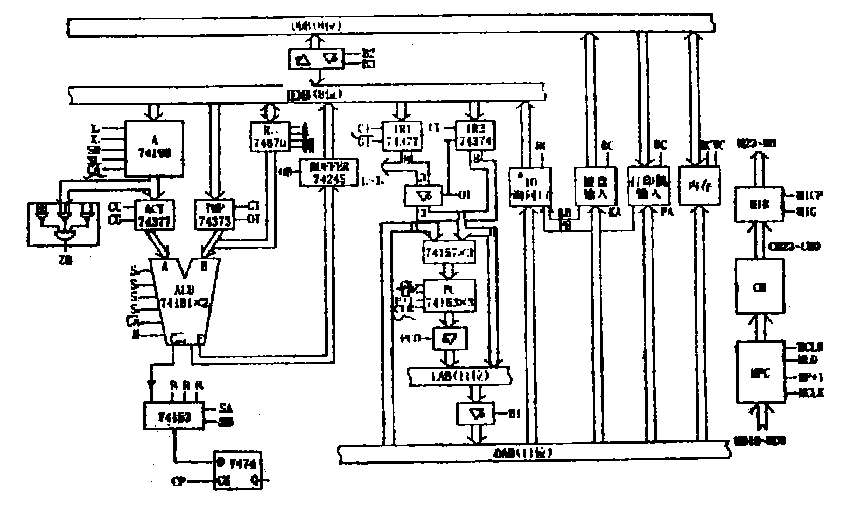
启动时KB=1。运行后，每当有数字键按下时使KB=0，表示有键输入，键码（00-0FH）被写入键盘数据缓冲器中。该数据缓冲器总是存放最新键码。

实验计算机运行时（RO=1）需读键盘时，应执行指令使OAB10=1（选外设）、KA=1（选键盘）、RC有效（负脉冲）。这样，可将键盘数据缓冲器内容读至ODB，并使键盘状态KB为”1”。这时如无新键输入，则键盘数据缓冲器仍存放原来的数据。

IO询问口地址为404H。CPU在要键盘输入前均先执行读询问口指令，使下图中的两个三态输出门开放，把KB和PB的状态分别传送到内部数据总线最高位IDB0和最低位IDB7，并写入某个累加器或寄存器，供CPU判别当前可否使用键盘或打印机。



3. 实验框图



4. 软接线表

MODULE CPU

"define computer input/output pins

"block1

IAB10 PIN 45;

IAB9 PIN 44;

IAB8 PIN 43;

IAB7 PIN 42;

IAB6 PIN 41;

IAB5 PIN 40;

IAB4 PIN 39;

IAB3 PIN 38;

IAB2 PIN 37;

IAB1 PIN 36;

IAB0 PIN 35;

RC PIN 34;

WC PIN 32;

B2 PIN 31;

IDB0 PIN 30;

IDB1 PIN 29;

IDB2 PIN 28;

IDB3 PIN 27;

IDB4 PIN 26;

IDB5 PIN 25;

IDB6 PIN 24;

IDB7 PIN 23;

I0 PIN 22;

I1 PIN 21;

I2 PIN 13;

I3 PIN 12;

I4 PIN 11;

I5 PIN 10;

I6 PIN 9;

I7 PIN 8;

PCO PIN 7;

PINC PIN 6;

LP PIN 5;

GI PIN 4;

OI PIN 3;

"BLOCK2

CL PIN 2;

RR PIN 128;

WR PIN 127;

SB PIN 126;

SA PIN 125;

A PIN 99;

B PIN 100;

P1 PIN 123;

MIG PIN 56;

CG PIN 124;

CY PIN 110;

CT PIN 122;

OT PIN 121;

ZD PIN 20;

CP PIN 120;

S3 PIN 119;

S2 PIN 118;

S1 PIN 117;

S0 PIN 109;

M PIN 108;

CN PIN 107;

SR PIN 106;

SL PIN 105;

X0 PIN 104;

X1 PIN 103;

A7 PIN 46;

A0 PIN 51;

OB PIN 102;

RO PIN 78;

DR PIN 101;

CLK PIN 84;

"BLOCK3

M23 PIN 98;

M22 PIN 96;

M21 PIN 95;

M20 PIN 94;

M19 PIN 93;

M18 PIN 92;

M17 PIN 91;

M16 PIN 90;

M15 PIN 89;

M14 PIN 88;

M13 PIN 87;

M12 PIN 86;

M11 PIN 85;

M10 PIN 77;

M9 PIN 76;

M8 PIN 75;

MD8 PIN 66;

MD7 PIN 63;

MD6 PIN 62;

MD5 PIN 61;

MD4 PIN 60;

MD3 PIN 59;

M7 PIN 74;

M6 PIN 73;

M5 PIN 72;

M4 PIN 71;

M3 PIN 70;

M2 PIN 69;

M1 PIN 68;

M0 PIN 67;

MD2 PIN 58;

MLD PIN 57;

"BLOCK4

PA PIN 55;

KA PIN 53;

PB PIN 54;

KB PIN 52;

EQUATIONS

“\*\*\*\*ALU\*\*\*\*\*\*

“ACT

CG = M5;

“TMP

CT = 1;

OT = M7;

“ALU

CN = M23;

M = M22;

S3 = M21;

S2 = M20;

S1 = M19;

S0 = M18;

“\*\*\*\*A\*\*\*\*\*\*

X1 = M17;

X0 = M16;

SR = CY;

SL = CY;

“\*\*\*\*\*BUFFER\*\*\*\*\*

OB = M8;

“\*\*\*74153 CARRY CONTROL\*\*\*\*

“P2 = A7;

P1 = A0;

SA = X0;

SB = X1;

CP = M12 # !CLK;

“\*\*\*\*REG\*\*\*\*\*\*\*\*\*\*

RR = M1;

WR = M0 # !CLK;

A = I0;

B = I1;

“\*\*\*\*CONTROLLER\*\*\*\*\*\*

“IR1

GI = M4;

“IR2

CL = M10 # !CLK;

OI = M11;

“\*\*\*\*\*\*PC\*\*\*\*\*\*\*\*\*\*\*\*

PINC = M6;

LP = (([I4,I3] == 0) & ZD #

([I4,I3] == 1) & CY #

([I4,I3] == 2) & A0 #

([I4,I3] == 3)) & M9;

PCO = !M6;

“\*\*\*\*\*\*MEMORY\*\*\*\*\*\*\*\*

RC = M14 # !CLK;

WC = M15 # !CLK;

“\*\*\*\*\*\*START UP\*\*\*\*\*\*\*

DR = M13;

“\*\*\*\*\*\*\*\*MPC\*\*\*\*\*\*\*\*\*\*

MIG = 0;

MLD = M2;

[MD8..MD2] = [0,0,0,I7..I5,I2];

“\*\*\*\*\*\*IO CONTROL\*\*\*\*\*\*

B2 = IAB2 & IAB10 # RC;

IDB0 = IAB2 & IAB10 & !RC & KB;

KA = IAB0;

END

5. 指令系统设计（执行流程）

(1) 指令简介(详见执行流程图)

a. 指令格式

本机支持的指令分为单字节和双字节指令。设单字节指令和双字节指令的第一个字节为I7..I0, 则指令有下列字段:

操作码字段 opcode <- [I7, I6, I5, I2];

寄存器字段 ri <- [I1, I0];

地址高3位 addrh <- [I2, I1, I0];

地址字段 addr <- addrh ## IR2;

立即数字段 data <- IR2;

从上面可以看到addr的最高位和opcode的最低位重叠。而addr的最高位是是否读取IO设备的indicator, 所以在指令可以被用来同时操作memory和IO设备的时候，必须占用两条指令的opcode来完成工作, 例如lda. 当只能操作memory时, 可以令addr[10]恒为0, 这样就必须为该指令分配I2 = 0的opcode.

b. 指令分类

1. 运算指令: add, sub, and, dec, rrc

都具有指令格式opcode[3..1] 00 opcode[0] ri, 功能为 a <- a op ri

2. 数据传输指令: mov, lda, sta

执行数据传输操作, 分为寄存器直接寻址、寄存器间接寻址、立即数寻址、直接寻址。

寄存器直接、间接寻址的指令格式类似运算指令: (mov a, ri 和 mov a, @ri)

opcode[3..1] 00 opcode[0] ri

功能为 a <- ri 和 a <- (addrh ## ri)

立即数寻址的指令格式: (mov a, data 和 mov ri, data)

opcode[3..1] 00 opcode[0] ri, data

功能为 a <- data (此时ri字段任意) 和 ri <- data

直接寻址的指令格式：(lda addr 和 sta addr)

opcode[3..1] 00 addr

功能为 a <- (addr) 和 (addr) <- a

(2) 执行流程图: 见附录1

6. 微指令格式和指令微程序

(1) 微指令格式

微指令长24位，采用全水平不编码纯控制场的格式，和控制信号一一对应（组合信号还需和其他信号进行逻辑组合）.

(2) 指令微程序

见附录2

7．调试程序

(1) 调试存、取及停机指令

地址 指令 指令编码

202 55

205 LDA 202 82 ;(002) = 55 -> A

206 02

207 STA 20A A2 ;(A) = 55 -> 00A地址中

208 0A

209 HALT FF

20A 00

结果202内存地址中的内容被复制到20A中,(20A) = 55H

(2) 调试算术逻辑类指令

调试右移

地址 指令 指令编码

203 A9

20B LDA 202 82

20C 02

20D MOV R0,A 40

20E LDA 203 82

20F 03

210 ADD A,R0 00

211 RRC A,R0 C0

212 STA215 A2

213 15

214 HALT FF

215 00

结果202中的内容右移一位，存入215中，(215) = AA

调试减法

地址 指令 指令编码

216 LDA 202 82

217 02

218 MOV R0,A 40

219 LDA 003 82

21A 03

21B SUB A,R0 04

21C STA 020 A2

21D 20

21E HALT FF

220 00

结果（203）- （202）-> (220),(220) = 54

(3) 调试跳转类和输入输出类指令

地址 指令 指令编码

230 LDA 404 84

231 04

232 JA0 230 F2

233 30

234 LDA 401 84

235 01

236 STA 239 A2

237 39

238 HALT FF

239 00

结果读取键盘输入的字符存入到239中

(4) 调试MOV A,Ri指令

地址 指令 指令编码

23A 01

23B LDA 23A 82

23C 3A

23D MOV R1,A 41

23E MOV A,@R1 21

23F STA 242 A2

240 42

241 HALT FF

242 00

结果(242) = (101)

(5) 调试AND指令

地址 指令 指令编码

246 LDA 202 82

247 02

248 MOV R0,A 40

249 LDA 203 82

24A 03

24B AND A,R0 24

24C STA 24F A2

24D 4F

24E HALT FF

24F 00

结果（24F） = (202) ^ (203) = 01H

(6) 调试MOV A,#DATA指令

地址 指令 指令编码

252 00

253 MOV A,55 44

254 55

255 STA 252 A2

256 52

257 HALT FF

结果（252）= 55

(7) 调试ADD指令

地址 指令 指令编码

25A 00

25B MOV R2,44 62

25C 44

25D MOV A,22 44

25E 22

25F ADD A,R2 02

260 STA 25A A2

261 5A

262 HALT FF

结果（25A）= 44H + 22H = 66H

(8) 调试DEC指令

地址 指令 指令编码

265 00

266 MOV R3,10 63

267 10

268 DEC A,R3 67

269 STA 265 A2

26A 65

26B HALT FF

结果(265) = 10H – 01H = 0FH

(9) 调试JC指令

地址 指令 指令编码

26F 00

270 MOV A,05 44

271 05

272 MOV R0,07 60

273 07

274 SUB A,R0 04

275 JC 300 EB

276 00

277 MOV A,00 44

278 00

279 STA 26F A2

27A 6F

27B HALT FF

300 MOV A,01 44

301 01

302 STA 26F A2

303 6F

304 HALT FF

结果(26F) = 00，因为没有进位，所以JC没有跳转

8．应用程序

(1) 应用程序简介

应用程序实现了2个2位10进制的乘法，操作数从键盘输入，结果以bcd码保存于两个字节内。

本程序的算法为逐位交叉相乘相加，例如:

34 \* 56

= (3\*10 + 4) \* (5\*10 + 6) ……………………………………………………………… (a)

= 3\*5 \* 100 + (3\*6 + 4\*5) \* 10 + 4\*6 …………………………………………… (b)

= 15 \* 100 + 38 \* 10 + 24 ……………………………………………………………… (c)

化为上述中间结果后，从低位到高位进行进行整理:

= 15 \* 100 + (38 + 2) \* 10 + 4

= 15 \* 100 + 40 \* 10 + 4

= (15 + 4) \* 100 + 0 \* 10 + 4

= 19 \* 100 + 0 \* 10 + 4

= 1 \* 1000 + 9 \* 100 + 0 \* 10 + 4

= 1904

以上，最终结果的4个10进制位均已出现，随后只要把两位整合到1个字节中即可。

(2) 应用程序的实现

乘法程序主要分成如下步骤:

1. 读入数字(002..029)

此阶段依次从键盘读入4个数字，存放在变量d00..d11(0d1..0d4)中，其中读入数字用到一个过程rdkey(152..15c), 其功能是从键盘读入一个数字后放置在变量key(152)中，然后返回。由于读入后4个数字分别存放在4个内存单元中，所以此阶段结束，算法进行到上述(a)步。

2. 按位相乘(02a..075)

此阶段即上述算法的(b)和(c)步. 这里用到一个过程dmd(0d9..0f3), 此过程利用加法实现了两个10进制位的乘法。调用前首先将两个10进制位填入x0(0d9), x1(0da), 该过程会将x1自加x0次，将结果放在内存变量y(0db)中返回。

通过此过程，依次算出d00 \* d10, (d00 \* d11 + d01 \* d10), d01 \* d11, 分别存放在y0..y2(0d5..0d7)中，便得到了(c)的结果.

3. 十进制化(076..0ab)

此阶段主要用到一个过程nlr(0f4..10d), 有输入参数rem(0f4), 其中存放着需要整理的数。该过程通过对rem每次减10然后测试是否下溢，如果没有下溢，则将qut(0f5)加1, 如果下溢，则返回。于是返回后的rem即该数除以10的余数，而qut就是对应的商.

通过此过程，从低位到高位处理y0, y1, y2, 每次的余数存回正在处理的yi, 而商则加到y(i+1)上, 然后继续处理y(i+1). 最后处理完y2的商存放在y3(0d8)中. 至此，4位10进制数已经存在于内存中了，最后的步骤便是两两合并。

4. 两两合并(0ac..0d0)

此阶段主要用到过程cmb(128..151), 有输入参数d0和d1, 将两个数合并后存入yy然后返回。主要的算法就是将d1移到高4位，然后低4位清0, 最后将上述结果加上d0存入yy即可。

其中由于指令集的关系，没有左移指令，只有循环右移指令，而且右移后的结果并不覆盖原操作数，故使用上有点麻烦。因此写了个辅助过程shr(10e..127), 该过程的输入参数为shrx(10e), 该过程将shrx循环(但不包括cy)右移后写回shrx后返回。最后cmb只需要在一个循环中调用shr 4次即完成了将d1移动到高位的任务.

借助cmb, 只需将两两合并后的结果分别存入ly(000)和hy(001)中，便完成了整个乘法过程。

(3) 一些注记

a. 关于过程调用和返回

本机指令集没有call和ret, 所以函数调用和返回机制是通过jmp来完成的。对于调用者，在准备好参数后直接jmp到相应的过程即可。在准备的参数中，包括返回地址。这个地址存放在被调用过程的xxxip内存变量中，而这个变量的位置十分特殊，是该过程最后那条返回用的jmp指令的第二个指令字节。所以整个过程调用是通过修改该过程的返回指令本身完成的。

例如dmd过程的返回:

0f2 f8 ; done0: jmp (000)

0f3 00 ; dmdip: .db 0

注意0f3这个变量其实应该是jmp的第二个指令字，和0f2一起组成一条完整的jmp指令，因此在先行填写好的情况下，机器会毫不知情的将它作为一条指令执行，起到返回的作用。

这样做的一个限制是不可以递归调用，因为第二次调用的返回值会覆盖第一次的返回值，导致无法返回最初的调用点。但本程序中不用递归。另一个限制是jmp的高3位已经被定好，不做修改该。如上面的dmd过程，它的返回点就必须在000..0ff之间。超出范围的调用可以通过调用前直接修改该0f2字节完成，再返回后立即还原0f2即不会影响其他调用。但本程序唯一的大于100的返回点是cmb过程调用shr的返回点。而shr也仅在此处调用，所以shr的返回指令被写成了jmp (001), 也避免了修改第一指令字。

b. 关于指令集

本机的指令集为黑板上的示例指令集，未作任何修改。但黑板上的指令集由于没有mov a, ri指令，导致了很多不便。致使程序长度超出预料。在完成了此程序后，我们讨论了微调指令集对程序的影响，发现只要增加一条mov a, ri并把rrc a, ri修改为rrc a即可大大缩短程序长度。我们也做出了修改后的程序（此程序还用了另一个转变：dec a, ri转变为dec a, 这样可以缩短dec指令的执行周期，也增加了灵活度，操作数不必是在ri中，而可以直接是lda的结果。而原来的dec a, ri的功能，可以通过mov a, ri; dec a;两条指令来完成。可见mov a, ri的重要性）

修改后的程序见附录3.

(4) 应用程序代码

000 00 ; ly: .db 0

001 00 ; hy: .db 0

002 44 ; mov a, rtp00

003 08 ;

004 a1 ; sta rdkeyip

005 5c ;

006 f9 ; jmp rdkey

007 53 ;

008 81 ; rtp00: lda key

009 52 ;

00a a0 ; sta d01

00b d2 ;

00c 44 ; mov a, rtp01

00d 12 ;

00e a1 ; sta rdkeyip

00f 5c ;

010 f9 ; jmp rdkey

011 53 ;

012 81 ; rtp01: lda key

013 52 ;

014 a0 ; sta d00

015 d1 ;

016 44 ; mov a, rtp02

017 1c ;

018 a1 ; sta rdkeyip

019 5c ;

01a f9 ; jmp rdkey

01b 53 ;

01c 81 ; rtp02: lda key

01d 52 ;

01e a0 ; sta d11

01f d4 ;

020 44 ; mov a, rtp03

021 26 ;

022 a1 ; sta rdkeyip

023 5c ;

024 f9 ; jmp rdkey

025 53 ;

026 81 ; rtp03: lda key

027 52 ;

028 a0 ; sta d10

029 d3 ;

02a 80 ; lda d00

02b d1 ;

02c a0 ; sta x0

02d d9 ;

02e 80 ; lda d10

02f d3 ;

030 a0 ; sta x1

031 da ;

032 44 ; mov a, rtp10

033 38 ;

034 a0 ; sta dmdip

035 f3 ;

036 f8 ; jmp dmd

037 dc ;

038 80 ; rtp10: lda y

039 db ;

03a a0 ; sta y0

03b d5 ;

03c 80 ; lda d00

03d d1 ;

03e a0 ; sta x0

03f d9 ;

040 80 ; lda d11

041 d4 ;

042 a0 ; sta x1

043 da ;

044 44 ; mov a, rtp11

045 4a ;

046 a0 ; sta dmdip

047 f3 ;

048 f8 ; jmp dmd

049 dc ;

04a 80 ; rtp11: lda y

04b db ;

04c a0 ; sta y1

04d d6 ;

04e 80 ; lda d01

04f d2 ;

050 a0 ; sta x0

051 d9 ;

052 80 ; lda d10

053 d3 ;

054 a0 ; sta x1

055 da ;

056 44 ; mov a, rtp12

057 5c ;

058 a0 ; sta dmdip

059 f3 ;

05a f8 ; jmp dmd

05b dc ;

05c 80 ; rtp12: lda y

05d db ;

05e 40 ; mov r0, a

05f 80 ; lda y1

060 d6 ;

061 00 ; add a, r0

062 a0 ; sta y1

063 d6 ;

064 80 ; lda d01

065 d2 ;

066 a0 ; sta x0

067 d9 ;

068 80 ; lda d11

069 d4 ;

06a a0 ; sta x1

06b da ;

06c 44 ; mov a, rtp13

06d 72 ;

06e a0 ; sta dmdip

06f f3 ;

070 f8 ; jmp dmd

071 dc ;

072 80 ; rtp13: lda y

073 db ;

074 a0 ; sta y2

075 d7 ;

076 80 ; lda y0

077 d5 ;

078 a0 ; sta rem

079 f4 ;

07a 44 ; mov a, rtp20

07b 80 ;

07c a1 ; sta nlrip

07d 04 ;

07e f8 ; jmp nlr

07f f6 ;

080 80 ; rtp20: lda rem

081 f4 ;

082 a0 ; sta y0

083 d5 ;

084 80 ; lda qut

085 f5 ;

086 40 ; mov r0, a

087 80 ; lda y1

088 d6 ;

089 00 ; add a, r0

08a a0 ; sta rem

08b f4 ;

08c 44 ; mov a, rtp21

08d 92 ;

08e a1 ; sta nlrip

08f 04 ;

090 f8 ; jmp nlr

091 f6 ;

092 80 ; rtp21: lda rem

093 f4 ;

094 a0 ; sta y1

095 d6 ;

096 80 ; lda qut

097 f5 ;

098 40 ; mov r0, a

099 80 ; lda y2

09a d7 ;

09b 00 ; add a, r0

09c a0 ; sta rem

09d f4 ;

09e 44 ; mov a, rtp22

09f a4 ;

0a0 a1 ; sta nlrip

0a1 04 ;

0a2 f8 ; jmp nlr

0a3 f6 ;

0a4 80 ; rtp22: lda rem

0a5 f4 ;

0a6 a0 ; sta y2

0a7 d7 ;

0a8 80 ; lda qut

0a9 f5 ;

0aa a0 ; sta y3

0ab d8 ;

0ac 80 ; lda y0

0ad d5 ;

0ae a1 ; sta d0

0af 28 ;

0b0 80 ; lda y1

0b1 d6 ;

0b2 a1 ; sta d1

0b3 29 ;

0b4 44 ; mov a, rtp30

0b5 ba ;

0b6 a1 ; sta cmbip

0b7 51 ;

0b8 f9 ; jmp cmb

0b9 2c ;

0ba 81 ; rtp30: lda yy

0bb 2a ;

0bc a0 ; sta ly

0bd 00 ;

0be 80 ; lda y2

0bf d7 ;

0c0 a1 ; sta d0

0c1 28 ;

0c2 80 ; lda y3

0c3 d8 ;

0c4 a1 ; sta d1

0c5 29 ;

0c6 44 ; mov a, rtp31

0c7 cc ;

0c8 a1 ; sta cmbip

0c9 51 ;

0ca f9 ; jmp cmb

0cb 2c ;

0cc 81 ; rtp31: lda yy

0cd 2a ;

0ce a0 ; sta hy

0cf 01 ;

0d0 ff ; halt

0d1 00 ; d00: .db 0

0d2 00 ; d01: .db 0

0d3 00 ; d10: .db 0

0d4 00 ; d11: .db 0

0d5 00 ; y0: .db 0

0d6 00 ; y1: .db 0

0d7 00 ; y2: .db 0

0d8 00 ; y3: .db 0

0d9 00 ; x0: .db 0

0da 00 ; x1: .db 0

0db 00 ; y: .db 0

0dc 44 ; dmd: mov a, 0

0dd 00 ;

0de a0 ; sta y

0df db ;

0e0 80 ; next0: lda x0

0e1 d9 ;

0e2 e0 ; jz done0

0e3 f2 ;

0e4 40 ; mov r0, a

0e5 64 ; dec a, r0

0e6 a0 ; sta x0

0e7 d9 ;

0e8 80 ; lda x1

0e9 da ;

0ea 40 ; mov r0, a

0eb 80 ; lda y

0ec db ;

0ed 00 ; add a, r0

0ee a0 ; sta y

0ef db ;

0f0 f8 ; jmp next0

0f1 e0 ;

0f2 f8 ; done0: jmp (000)

0f3 00 ; dmdip: .db 0

0f4 00 ; rem: .db 0

0f5 00 ; qut: .db 0

0f6 44 ; nlr: mov a, 0

0f7 00 ;

0f8 a0 ; sta qut

0f9 f5 ;

0fa 63 ; mov r3, 10

0fb 0a ;

0fc 62 ; mov r2, 1

0fd 01 ;

0fe 80 ; next1: lda rem

0ff f4 ;

100 07 ; sub a, r3

101 e9 ; jc again1

102 05 ;

103 f8 ; jmp (000)

104 00 ; nlrip: .db 0

105 a0 ; again1: sta rem

106 f4 ;

107 80 ; lda qut

108 f5 ;

109 02 ; add a, r2

10a a0 ; sta qut

10b f5 ;

10c f8 ; jmp next1

10d fe ;

10e 00 ; shrx: .db 0

10f 44 ; shr: mov a, 0x7f

110 7f ;

111 43 ; mov r3, a

112 81 ; lda shrx

113 0e ;

114 40 ; mov r0, a

115 c0 ; rrc a, r0

116 e9 ; jc seth2

117 1b ;

118 27 ; and a, r3

119 f9 ; jmp done2

11a 24 ;

11b a1 ; seth2: sta shrx

11c 0e ;

11d 44 ; mov a, 0x80

11e 80 ;

11f 42 ; mov r2, a

120 81 ; lda shrx

121 0e ;

122 27 ; and a, r3

123 02 ; add a, r2

124 a1 ; done2: sta shrx

125 0e ;

126 f9 ; jmp (001)

127 00 ; shrip: .db 0

128 00 ; d0: .db 0

129 00 ; d1: .db 0

12a 00 ; yy: .db 0

12b 00 ; cnt3: .db 0

12c 81 ; cmb: lda d1

12d 29 ;

12e a1 ; sta shrx

12f 0e ;

130 44 ; mov a, 4

131 04 ;

132 a1 ; sta cnt3

133 2b ;

134 44 ; mov a, crtp3

135 3a ;

136 a1 ; sta shrip

137 27 ;

138 f9 ; again3: jmp shr

139 0f ;

13a 81 ; crtp3: lda cnt3

13b 2b ;

13c 40 ; mov r0, a

13d 64 ; dec a, r0

13e a1 ; sta cnt3

13f 2b ;

140 e1 ; jz next3

141 44 ;

142 f9 ; jmp again3

143 38 ;

144 44 ; next3: mov a, 0xf0

145 f0 ;

146 40 ; mov r0, a

147 81 ; lda d0

148 28 ;

149 41 ; mov r1, a

14a 81 ; lda shrx

14b 0e ;

14c 24 ; and a, r0

14d 01 ; add a, r1

14e a1 ; sta yy

14f 2a ;

150 f8 ; jmp (000)

151 00 ; cmbip: .db 0

152 00 ; key: .db 0

153 84 ; rdkey: lda 0x404

154 04 ;

155 f1 ; ja0 rdkey

156 53 ;

157 84 ; lda 0x401

158 01 ;

159 a1 ; sta key

15a 52 ;

15b f8 ; jmp (000)

15c 00 ; rdkeyip: .db 0

9．建议与体会

（1）体会：

通过这次的整机实验，我们了解了FD-CES系统硬件中各个功能模块的原理及其使用方法，根据整机逻辑框图，结合前两次部件实验中学习到的ABEL-HDL语言编写经验，参照书上示例，完成软接线表和微指令集的设计和实现，最后完成调试程序和应用程序的编写。

刚开始做的时候比较迷茫，有点不知所措、无从下手的感觉，然后通过对整机逻辑框图的仔细研究体会，终于写出了软接线表，在后面的调试中，由于软接线表中B2的设计错误导致进度停滞了很久，经过反复检查，调试，修改软接线表中与指令fetch有关的控制信号，找出了错误的症结所在。微指令的错误不多，而且也很快被解决了。随后的调试程序很简单，但应用程序比较长，在X86汇编上调通后，在将程序翻译成机器码的时候出现了不少的翻译错误，导致最后在实验计算机上调试的时候花费了比较长的时间。

总的来说，整机试验的难度比部件实验提高了不少，通过对书本的认真学习，在实验时遇到错误反复调试，碰到困难毫不气馁，终于还是顺利完成了实验。

PS：细心、耐心是关键，很多时候都是由于粗心大意造成的低级错误。

（2）建议

建议修改指令集，目前的指令集中只允许将数据从A累加器移到Ri寄存器，反之则不能经过一步操作完成，需MOV A,0和ADD A,Ri两步或以上操作才可完成。所以应该加一条MOV A,Ri的指令，而MOV A,@Ri指令没有多大用处，可以删除。此外可以把DEC A,Ri改为DEC A，这样在使用时更符合一般逻辑，同时对于使用2096芯片的小组可以在RLC指令的位置上添加INC A的指令。