ELE306 Fall 2001 Tiny CPU

1. General Descriptions:

The documentation herein describes a small CPU designed for the teaching purpose. This CPU is used in conjunction with the ELE306 Laboratory number 7. In the Lab example, this CPU is used to control a simple traffic light. Read the program listing in the Section 5 of this document to gain more insight to the CPU operations.

2. Instruction Set Architecture (ISA) Design

Here are the ISA components for this CPU:

1. 8-bit data bus
2. 8-bit address bus
3. Memory mapped I/O addresses
4. One Accumulator and no general purpose register
5. Five instructions have been implemented (room to expand to 8 or 16 instructions)
6. Only two addressing modes
7. Only "z" flag or condition code was implemented
8. Equipped with an Adder/Subtractor but only addition was implemented.
9. Built-in parallel I/O ports: one 4-bit parallel output port and one 4-bit parallel input port
10. This CPU and the built-in I/O ports used 20% of Xilinx's 4010XL FPGA CLB's

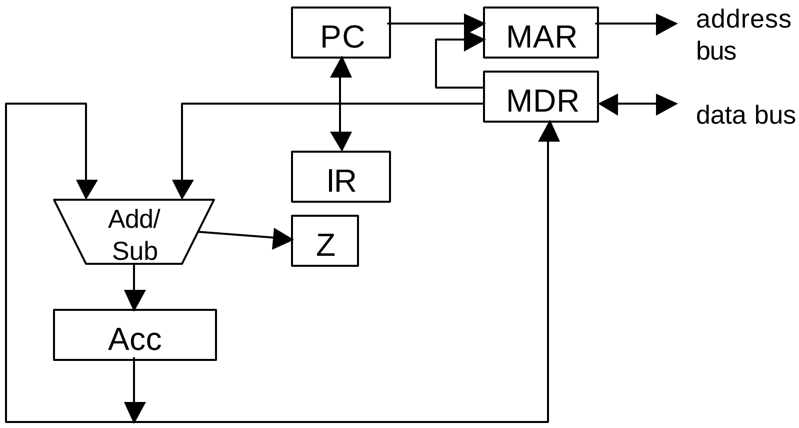
2.1. Instruction Set

The following is the instruction set (five instructions):

Assembly	Operations	Instruction format	Opcode	Z	cycles
ADD $M	Acc←Acc+M	OP(3) | unused | M	001	Y	5
STR $M	M←Acc	OP(3) | unused | M	010	N	4
CLA	Acc←0	OP(3) | unused	011	N	1
JNZ $M	PC←M if Z=0	OP(3) | unused | M	101	N	3
RST	PC←0	OP(3) | unused	111	N	1

2.2. Data Path

Details of the following data path block diagram can be verified from the schematic.



*The Acc is an 8-bit data register.
*A multiplexer is used for the MAR input.

2.3. Control Signals

There are 12 control signals. However, you will see only 11 in the table below since the loading of Acc is coincided with the loading of the Z flag (therefore CLA does not affect the Z flag).

Inst.	state	Micro-operations	Control signals										
			accl	cla	mmx	mal	mdol	mdil	intrs	pcinc	pcl	irl	mem_w
IF	S1	MAR←PC; MEM_RD;			1	1							
	S2	MDR←MEM; PC←PC+1;						1		1			
	S2	IR←MDR;										1	
ADD	S4	MAR←PC; MEM_RD;			1	1							
	S5	MDR←MEM; PC←PC+1;						1		1			
	S6	MAR←MDR; MEM_RD;				1							
	S7	MDR←MEM;						1					
	S8	Acc←Acc+MDR;	1										
STR	S4	MAR←PC; MEM_RD;			1	1							
	S5	MDR←MEM; PC←PC+1;						1		1			
	S6	MAR←MDR; MDR←Acc;				1	1						
	S7	MEM_WR;											1
CLA	S4	Acc←"00";		1									
JNZ	S4	MAR←PC; MEM_RD;			1	1							
	S5	MDR←MEM; PC←PC+1;						1		1			
	S6	If(Z='0') PC←MDR;									1		
RST	S4	PC←"00";							1				

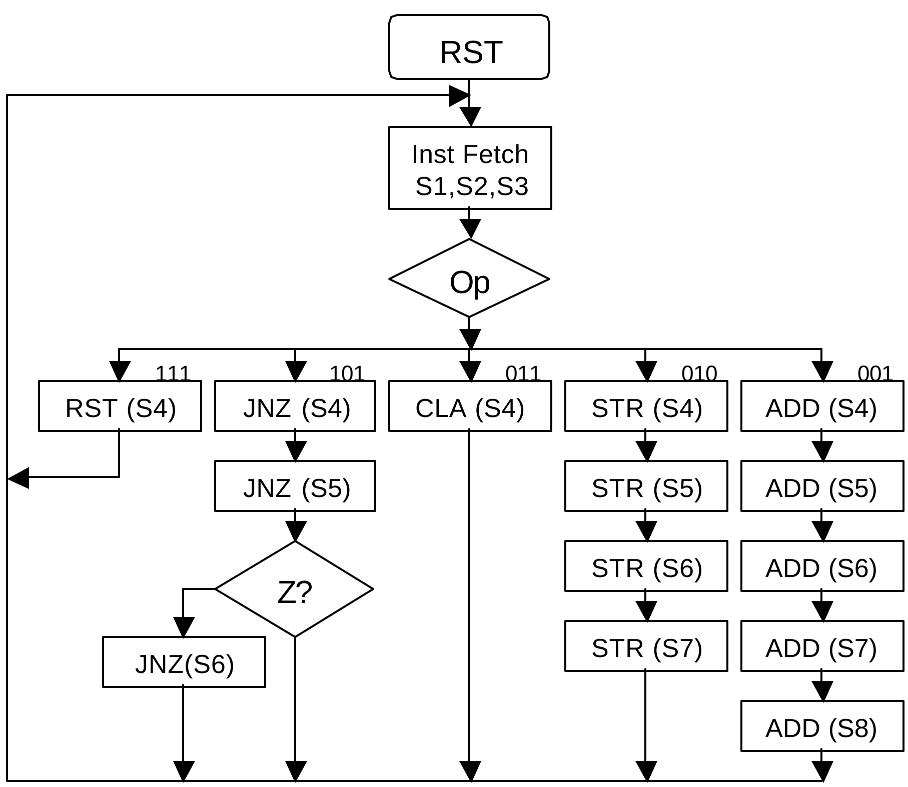
- Accl: load enable for Acc and is also the load enable for the Z flip-flop.
- Cla: reset signal to Acc
- Mmx: selecting signal to the MAR's multiplexer. 0 for MDR and 1 for PC
- Mal: load enable signal for MAR
- Mdol: load enable for the MDR output register (to memory/IO)
- Mdil: load enable for the MDR input register (from memory/IO)
- Intrst: the internal reset signal for the PC (to be merged with the external RST)
- Pcinc: enable the increment of PC
- Pcl: load enable for PC
- Irl: load enable for IR
- Mem_wr: read/write control to memory/IO; 0: read and 1: write

NOTE1: IF stands for instruction fetch. It occurs before the instruction (see below)
NOTE2: The first micro-operation of each instruction always starts at S4, because the IF takes three states to complete (see below).

3. Control Unit Design

The control unit is design as a finite state machine. It mission is simply to implement the control signals specified in the above table.

3.1. Flowchart



NOTE1: For simplicity, I listed only ADD (S4), ADD (S5), etc., you should refer to the micro-operations-to-control signals table above for the detailed micro-operations and control signal assignments.
NOTE2: For JNZ, the decision on Z is made at S6. So is the action if taken.

3.2. Finite State Machine

From the above flowchart, we know that we need a FSM with only eight states (the longest route goes through ADD instruction). My approach is to design a simple FSM that goes through eight states (S1 through S8) as indicated in the flowchart. For instance, at S4, if the Opcode is either "011" or "111" then the FSM should go right back to S1. At S6, check if the Opcode is "101" and go back to S1 if so. At S7, go back to S1 if Opcode is "010". Finally, always go back to S1 after S8. Design the FSM based on this and ignore all the control signals for now.

Once the FSM structure is established, we will establish the logic that generates the control signals. For instance, IRL is '1' only when in S3 and therefore

```
IRL <= '1' when (Sreg0=S3) else
       '0';
```

Here, Sreg0 is the internal state variable as found in the VHDL code. Similarly, MMX is '1' in the following conditions:

S1
Opcode="001" (ADD) and S4
Opcode="010" (STR) and S4
Opcode="101" (JNZ) and S4

A similar VHDL assignment can be constructed:

```
MMX <=    '1' when (Sreg0=S1) else
          '1' when (IR(7 downto 5)="001" and Sreg0=S4) else
          '1' when (IR(7 downto 5)="010" and Sreg0=S4) else
          '1' when (IR(7 downto 5)="101" and Sreg0=S4) else
          '0';
```

IR(7 downto 5) is of course the Opcode field of the IR!

3.3. State Diagram and HDL Editors

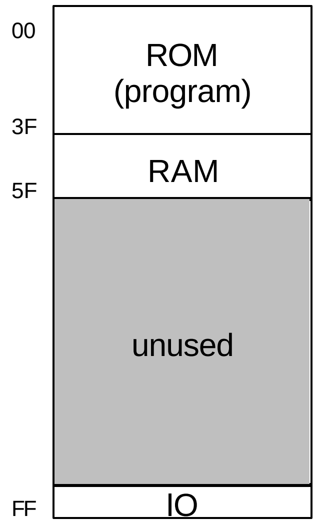
While the state diagram editor is quite convenient to generate the FSM structure, it is quite awkward to type in all the logic assignments. My preference is to use the state diagram editor to establish the FSM structure and then use the HDL editor to finish the logic assignment. Of course, the macro is then generated from the HDL editor. (You may check the Control.asf for the draft and compare it to the final Control.vhd)

4. Main Memory and I/O

For the demonstration purpose, the CPU is explicitly connected to the main memory and the parallel I/O making it a complete embedded system. There are three components: ROM, RAM and I/O here. The ROM contains the program that runs the traffic lights. The RAM is basically unused in this demonstration but maybe useful as a demonstration and/or lab assignment purpose.

4.1. Memory Map

Since I/O is memory mapped, all three memory components are found in the memory map. There are 64 bytes in ROM, 32 bytes in RAM and one byte address for the I/O (see below).



4.2. Interface

Since the ROM only need six address bits, special attention is given to connect it to the CPU. Note that on the schematic, the address bus going into ROM needs to be named MEM_ADR[5:0] to indicated the desired connections to the main MEM_ADR[7:0] bus. Similar arrangement is need for the RAM (in this case only five address bits). The I/O actually uses all eight address bits to decode.

A multiplexer is used here to select the appropriate memory input to the CPU. This is commonly implemented with tri-state buffers. However, a multiplexer will serve the same purpose and is much easier to negotiate when simulating the circuits. Note: A buffer is added to tap into the memory address bus for just one bit MEM_ADR6.

4.3. Parallel I/O

The Parallel I/O has a 4-bit input and a 4-bit output. Therefore, the higher 4 bits from the data bus were not used here. You may add them if you wish. The input and output ports share one memory address such that when write to address $FF you write to the output port and when read from address $FF you read from the input port.

4.4. Miscellaneous

The OR gates on the main schematic is used to merge the external reset (from an outside pin) and the internal reset (generated by the control unit when RST instruction executes). This will control the reset signal on PC. Also, the OSC4 is used here to provide clock signals. These clock signals are routed to the pins so clock frequency can be changed from outside jumpers. (NOTE: OSC4 allows only three outputs be used at the same time and the 8MHz output must be used)

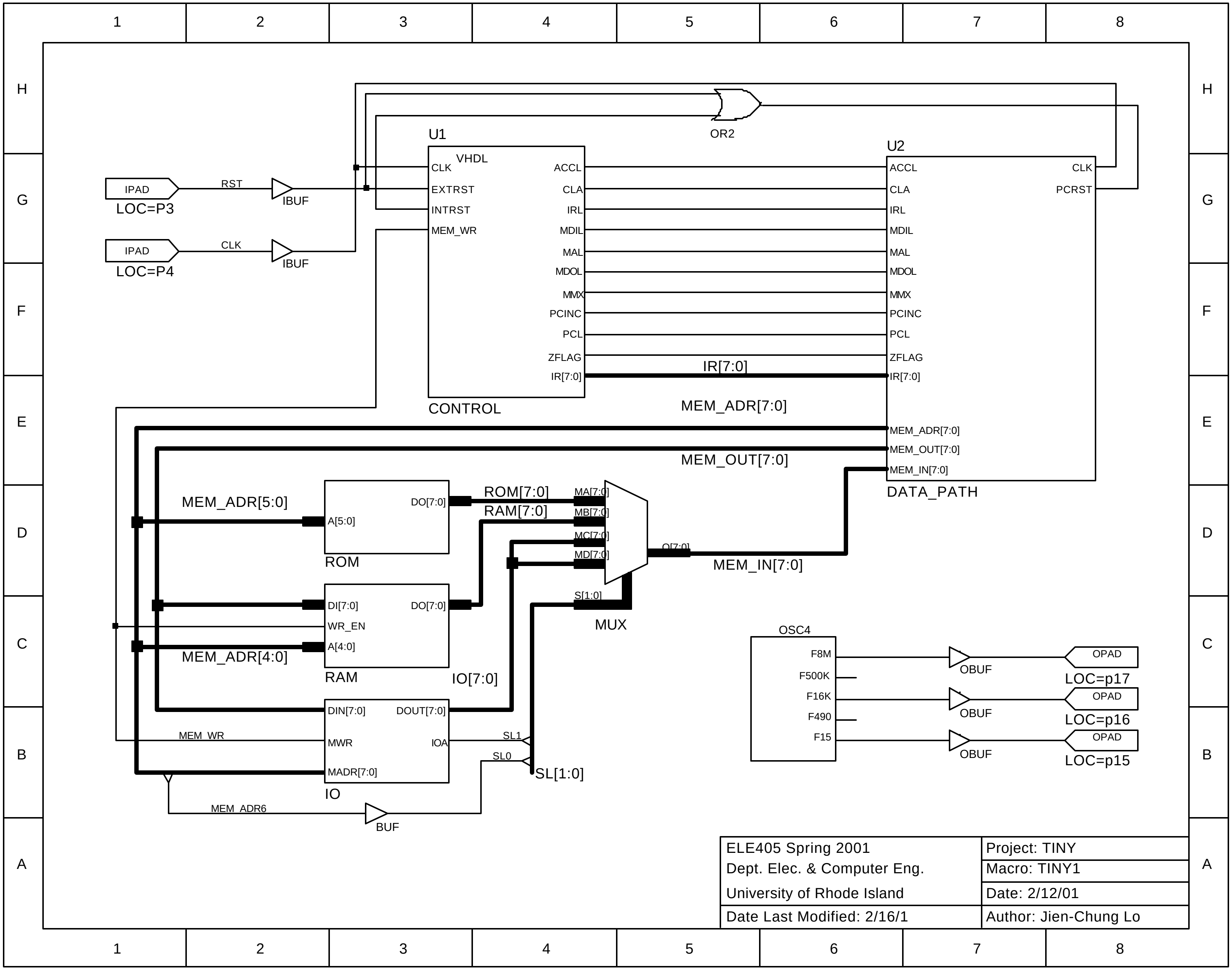
5. Demonstration

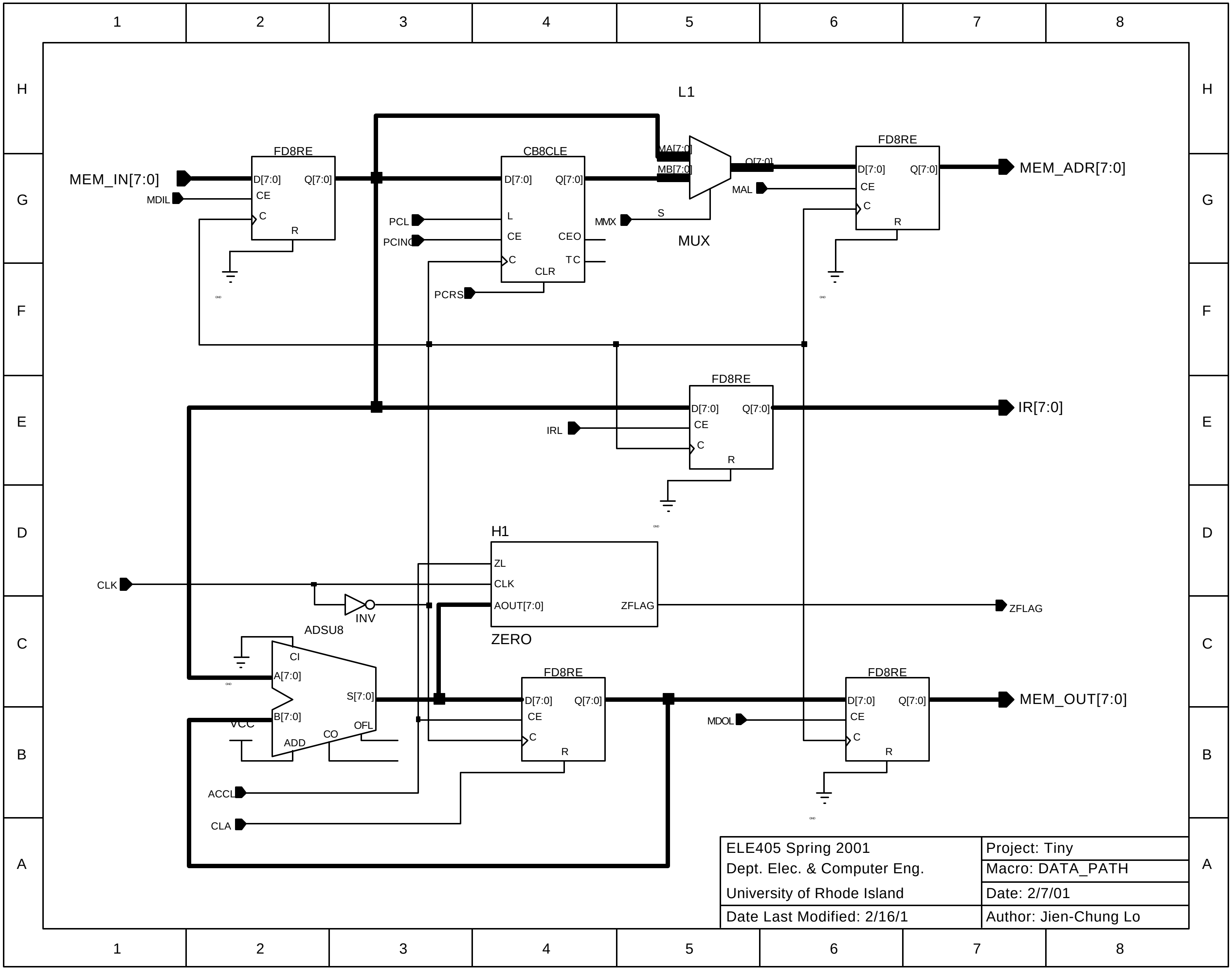
The pin assignments:

Reset:	Pin 3 ;the external reset signal, active high
Clock:	Pin 4 ;the system clock input
OSC:	Pin 15 → 15Hz Pin 16 → 16KHz Pin 17 → 8MHz
Output:	Pin 5 → Green LED Pin 6 → Yellow LED (actually amber) Pin 7 → Red LED Pin 8 → unused
Input	Pin 35 → Fast/Slow control Pin 36 → unused Pin 37 → unused Pin 38 → unused

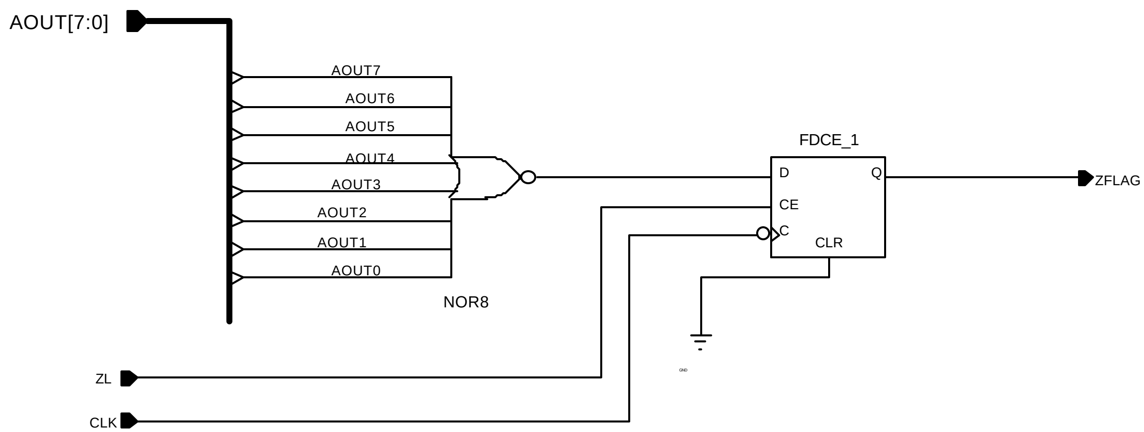
The program listing (ROM contents):

```
Addr. Machine Codes
00    60                  CLA          ;Clear Acc and then add the operand is
01    20 22               ADD $GDAT    ;equivalent to move the operand to Acc
03    40 FF               STR $IOP
05    60                  CLA
06    20 25               ADD $ONE
08    20 FF               ADD $IOP
0A    A0 0E               JNZ $FG      ;checking fast/slow input
0C    60                  CLA
0D    60                  CLA
0E    60            FG:   CLA
0F    20 23               ADD $YDAT
11    40 FF               STR $IOP
13    60                  CLA
14    20 24               ADD $RDAT
16    40 FF               STR $IOP
18    60                  CLA
19    20 25               ADD $ONE
1B    20 FF               ADD $IOP
1D    A0 21               JNZ $RG      ;checking fast/slow input
1F    60                  CLA
20    60                  CLA
21    E0            RG:   RST          ;jump back to addr. 00

22    FE            GDAT: DAT   1111 1110   ;Constant to turn on Green LED
23    FD            YDAT: DAT   1111 1101   ;Constant for Yellow LED
24    FB            RDAT: DAT   1111 1011   ;Constant for Red LED
25    01            ONE:  DAT   0000 0001   ;Constant 1
                    IOP:  EQU   $FF
```

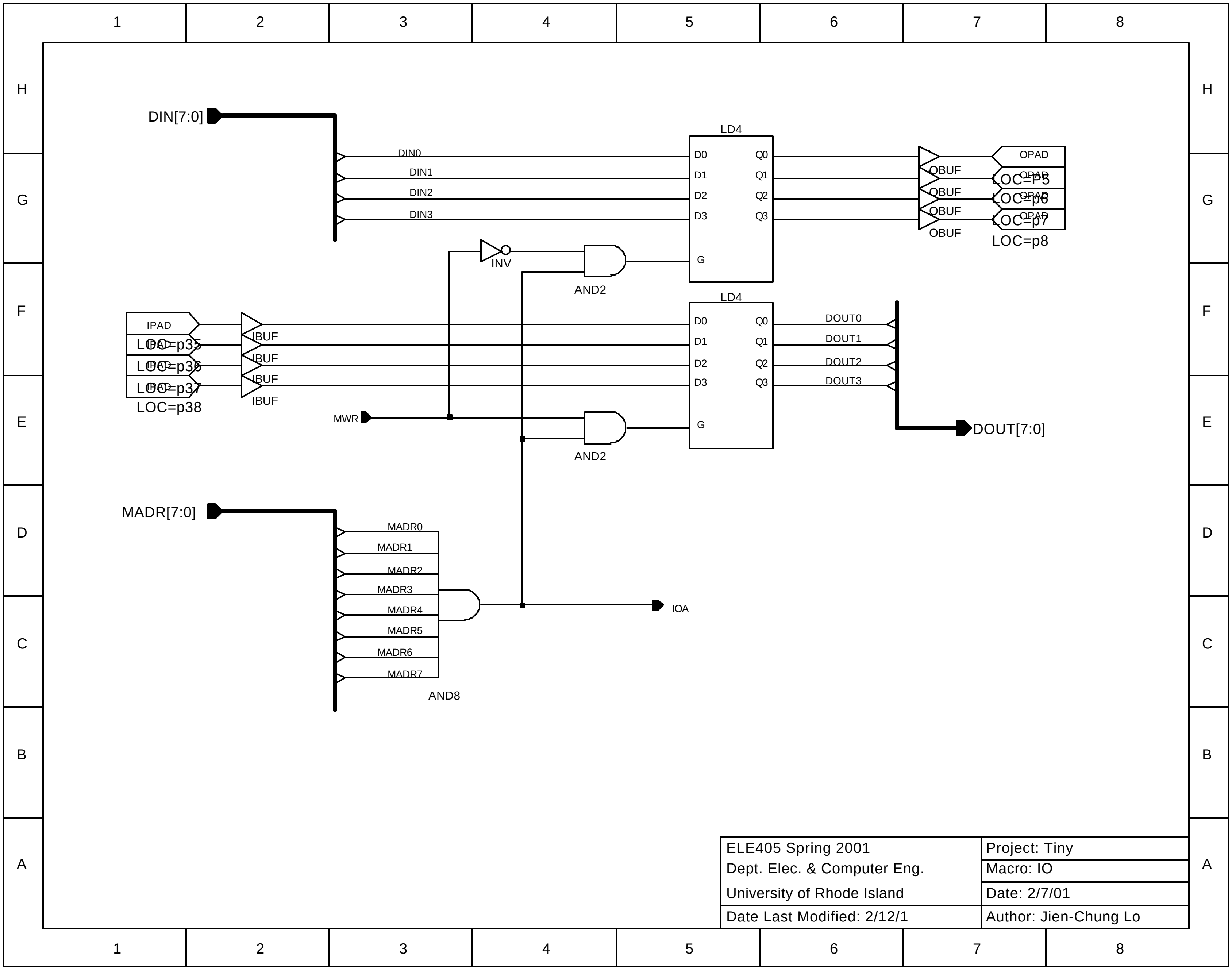

U1
VHDL
CLK
EXTRST
INTRST
MEM_WR
ACCL
CLA
IRL
MDIL
MAL
MDOL
MMX
PCINC
PCL
ZFLAG
IR[7:0]
CONTROL
OR2
U2
DATA_PATH
PCRST
MEM_ADR[7:0]
MEM_OUT[7:0]
MEM_IN[7:0]
IPAD
RST
IBUF
LOC=P3
CLK
LOC=P4
MEM_ADR[5:0]
A[5:0]
DO[7:0]
ROM
ROM[7:0]
RAM[7:0]
MA[7:0]
MB[7:0]
MC[7:0]
MD[7:0]
O[7:0]
S[1:0]
MUX
DI[7:0]
WR_EN
A[4:0]
MEM_ADR[4:0]
RAM
IO[7:0]
DIN[7:0]
DOUT[7:0]
MWR
IOA
MADR[7:0]
IO
MEM_WR
SL1
SL0
SL[1:0]
MEM_ADR6
BUF
OSC4
F8M
F500K
F16K
F490
F15
OBUF
OPAD
LOC=p17
LOC=p16
LOC=p15
ELE405 Spring 2001
Dept. Elec. & Computer Eng.
University of Rhode Island
Date Last Modified: 2/16/1
Project: TINY
Macro: TINY1
Date: 2/12/01
Author: Jien-Chung Lo



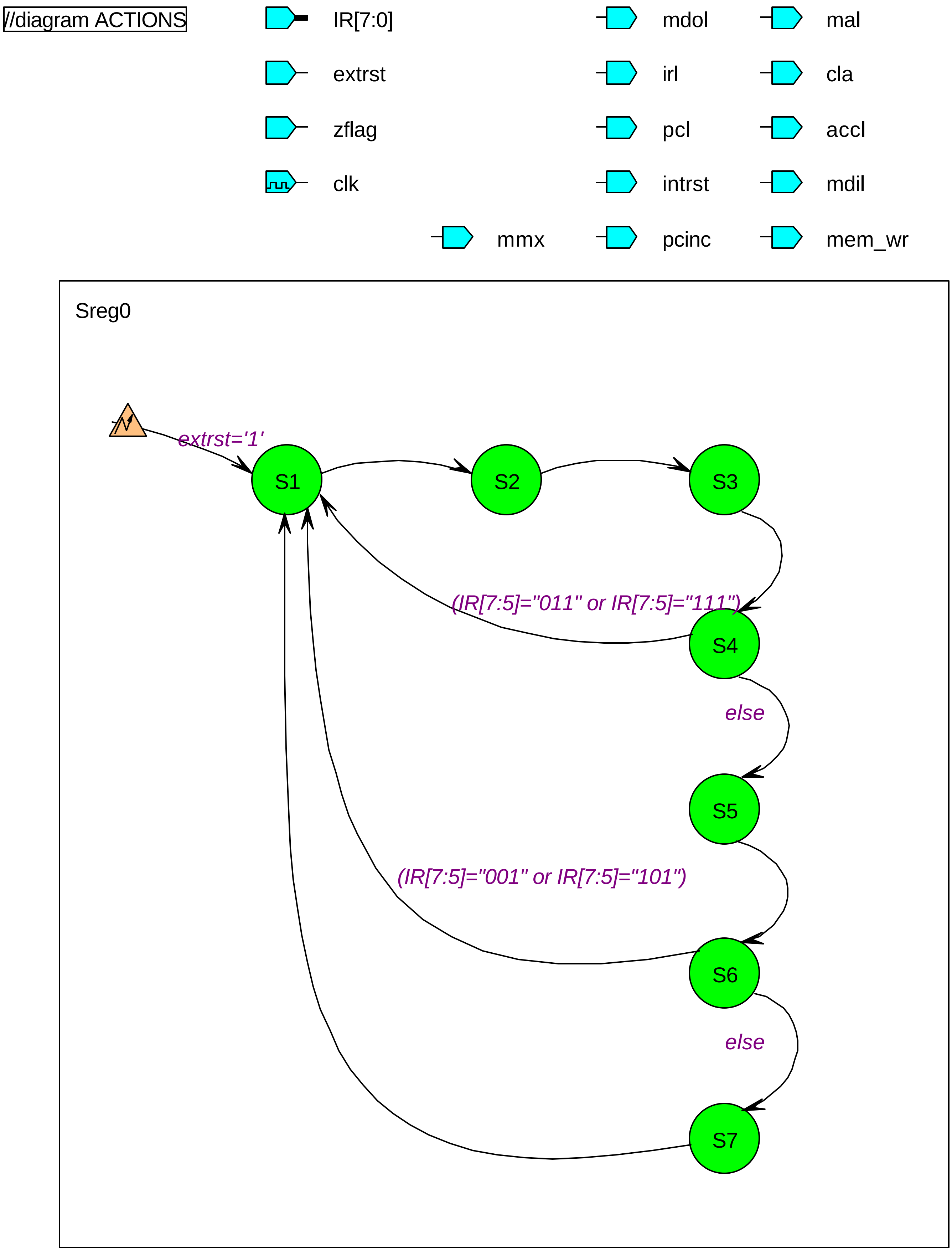
L1
MUX
FD8RE
CB8CLE
MEM_IN[7:0]
MDIL
PCL
PCINC
PCRS
MMX
MAL
MEM_ADR[7:0]
IRL
IR[7:0]
H1
ZERO
ZL
CLK
AOUT[7:0]
ZFLAG
INV
ADSU8
VCC
ACCL
CLA
MDOL
MEM_OUT[7:0]
ELE405 Spring 2001
Dept. Elec. & Computer Eng.
University of Rhode Island
Date Last Modified: 2/16/1
Project: Tiny
Macro: DATA_PATH
Date: 2/7/01
Author: Jien-Chung Lo



ELE405 Spring 2001	Project: Tiny
Dept. Elec. & Computer Eng.	Macro: ZERO
University of Rhode Island	Date: 2/7/01
Date Last Modified: 2/12/1	Author:


DIN[7:0]
DIN0
DIN1
DIN2
DIN3
LD4
D0 D1 D2 D3 G
Q0 Q1 Q2 Q3
OBUF
OPAD
LOC=P5
LOC=p6
LOC=p7
LOC=p8
INV
AND2
IPAD
LOC=p35
LOC=p36
LOC=p37
LOC=p38
IBUF
DOUT0
DOUT1
DOUT2
DOUT3
DOUT[7:0]
MWR
MADR[7:0]
MADR0
MADR1
MADR2
MADR3
MADR4
MADR5
MADR6
MADR7
AND8
IOA
ELE405 Spring 2001
Dept. Elec. & Computer Eng.
University of Rhode Island
Date Last Modified: 2/12/1
Project: Tiny
Macro: IO
Date: 2/7/01
Author: Jien-Chung Lo

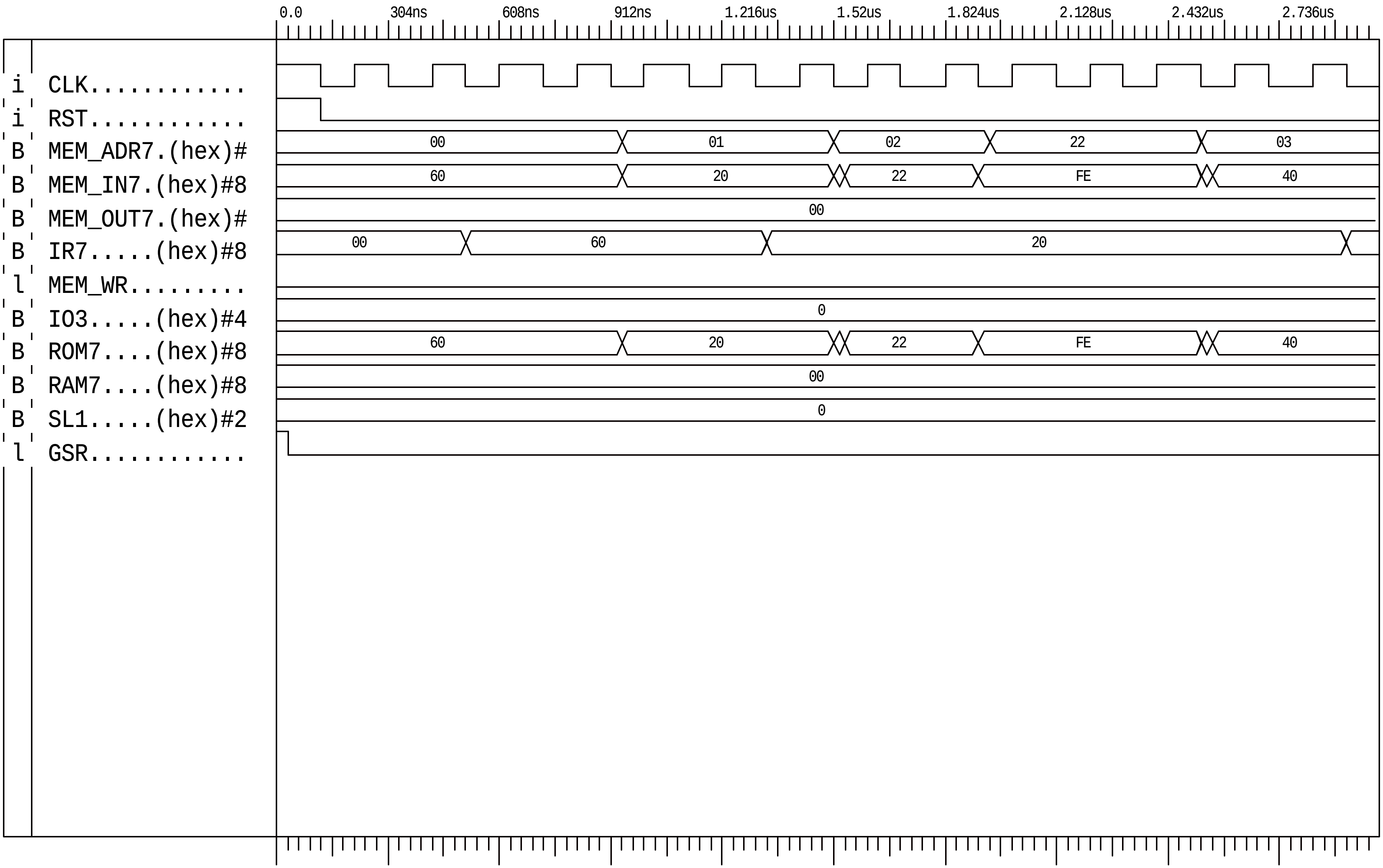
Control



//diagram ACTIONS
IR[7:0]
extrst
zflag
clk
mmx
mdol
irl
pcl
intrst
pcinc
mal
cla
accl
mdil
mem_wr
Sreg0
extrst='1'
S1
S2
S3
(IR[7:5]="011" or IR[7:5]="111")
S4
else
S5
(IR[7:5]="001" or IR[7:5]="101")
S6
else
S7

```
--
-- File: C:\WINDOWS\DESKTOP\COURSES\405\DESIGNS\TINY\Control.vhd
-- created: 02/09/01 08:34:58
-- from: 'C:\WINDOWS\DESKTOP\COURSES\405\DESIGNS\TINY\Control.asf'
-- by fsm2hdl - version: 2.0.1.53
--
library IEEE;
use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library SYNOPSYS;
use SYNOPSYS.attributes.all;

entity control is
  port (clk: in STD_LOGIC;
        extrst: in STD_LOGIC;
        IR: in STD_LOGIC_VECTOR (7 downto 0);
        zflag: in STD_LOGIC;
        accl: out STD_LOGIC;
        cla: out STD_LOGIC;
        intrst: out STD_LOGIC;
        irl: out STD_LOGIC;
        mal: out STD_LOGIC;
        mdil: out STD_LOGIC;
        mdol: out STD_LOGIC;
        mem_wr: out STD_LOGIC;
        mmx: out STD_LOGIC;
        pcinc: out STD_LOGIC;
        pcl: out STD_LOGIC);
end;

architecture control_arch of control is


-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S1, S2, S3, S4, S5, S6, S7, S8);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments


Sreg0_machine: process (clk, extrst)

begin

if extrst='1' then
        Sreg0 <= S1;
elsif clk'event and clk = '1' then
        case Sreg0 is
                when S1 =>
                        Sreg0 <= S2;
                when S2 =>
                        Sreg0 <= S3;
                when S3 =>
                        Sreg0 <= S4;
                when S4 =>
                        if (IR(7 downto 5)="011" or IR(7 downto 5)="111") then
                                Sreg0 <= S1;
                        else    Sreg0 <= S5;
                        end if;
                when S5 =>
                        Sreg0 <= S6;
                when S6 =>
                        if (IR(7 downto 5)="101") then
                                Sreg0 <= S1;
                        else Sreg0 <= S7;
                        end if;
                when S7 =>
                        if (IR(7 downto 5)="010") then
                                Sreg0 <= S1;
                        else Sreg0 <= S8;
                        end if;
                when S8 =>
                        Sreg0 <= S1;
                when others =>
                        null;
        end case;
end if;
end process;

--Control Signal Assignments

accl <= '1' when (IR(7 downto 5)="001" and Sreg0=S8) else
        '0';

cla <=  '1' when (IR(7 downto 0)="01100000" and Sreg0=S4) else
        '0';

mmx <=  '1' when (Sreg0=S1) else
        '1' when (IR(7 downto 5)="001" and Sreg0=S4) else
        '1' when (IR(7 downto 5)="010" and Sreg0=S4) else
        '1' when (IR(7 downto 5)="101" and Sreg0=S4) else
        '0';

```

```
mal <= '1' when (Sreg0=S1) else
        '1' when (IR(7 downto 5)="001" and Sreg0=S4) else
        '1' when (IR(7 downto 5)="001" and Sreg0=S6) else
        '1' when (IR(7 downto 5)="010" and Sreg0=S4) else
        '1' when (IR(7 downto 5)="010" and Sreg0=S6) else
        '1' when (IR(7 downto 5)="101" and Sreg0=S4) else
        '0';

mdol <= '1' when (IR(7 downto 5)="010" and Sreg0=S6) else
        '0';

mdil <= '1' when (Sreg0=S2) else
        '1' when (IR(7 downto 5)="001" and Sreg0=S5) else
        '1' when (IR(7 downto 5)="001" and Sreg0=S7) else
        '1' when (IR(7 downto 5)="010" and Sreg0=S5) else
        '1' when (IR(7 downto 5)="101" and Sreg0=S5) else
        '0';

intrst<='1' when (IR(7 downto 5)="111" and Sreg0=S4) else
        '0';

pcinc<='1' when (Sreg0=S2) else
        '1' when (IR(7 downto 5)="001" and Sreg0=S5) else
        '1' when (IR(7 downto 5)="010" and Sreg0=S5) else
        '1' when (IR(7 downto 5)="101" and Sreg0=S5) else
        '0';

pcl <= '1' when (IR(7 downto 5)="101" and Sreg0=S6 and zflag='0') else
        '0';

irl <= '1' when (Sreg0=S3) else
        '0';

mem_wr<='1' when (IR(7 downto 5)="010" and Sreg0=S7) else
        '0';

end control_arch;
```


0.0
304ns
608ns
912ns
1.216us
1.52us
1.824us
2.128us
2.432us
2.736us
i CLK............
i RST............
B MEM_ADR7.(hex)#
00
01
02
22
03
B MEM_IN7.(hex)#8
60
20
22
FE
40
B MEM_OUT7.(hex)#
00
B IR7.....(hex)#8
00
60
20
l MEM_WR.........
B IO3.....(hex)#4
0
B ROM7....(hex)#8
60
20
22
FE
40
B RAM7....(hex)#8
00
B SL1.....(hex)#2
0
l GSR............