ECE/COMP SCI 352

DIGITAL SYSTEM FUNDAMENTALS

DESIGN PROJECT 2

Due: Beginning of Class On Wednesday, April 11

In this project, you are to design the datapath for the BLT (Binary LogicTrainer), a simple, clearly unmarketable, digital appliance. This datapath will consist of the eight operation ALU designed in Project 1 along with additional datapath logic and register storage. You will design the additional datapath logic and registers and then interconnect them to form the datapath. The goals of this project are to:

1. **Improve your understanding of material covered in class, and**
2. **To provide a datapath module to be combined with a control module to implement the core processing logic for the BLT.**

This project is to be submitted and will be graded. It is to be performed by teams of two (not three or more) students to permit a more significant project and foster collaboration on difficult parts and in Mentor Graphics usage. The team is not to copy designs, simulations files, or other results from other student teams. However, sharing knowledge and tips on how to use Mentor Graphics with other teams is allowed and encouraged.

As a part of the project report, a table is to be included that reports in detail what parts of the project were performed by each team member; the table to be completed and submitted is included at the end of this write-up. Ordinarily the same grade will be given to each team member, but if there is a significant imbalance in overall contribution, individual grades will be given.

This project write-up is organized as follows. Initially, we give a general overview of the BLT followed by a specification of the datapath itself. Next we give the general steps to follow in the design and specify what is to be included in the project report. You are strongly encourage to study this entire write-up before beginning the project.

BINARY LOGIC TRAINER OVERVIEW

The exterior view of the BLT is shown in Figure 1. In addition to the **Power/Reset** button, the BLT has two digit buttons, **0** and **1**, for entering binary digits, and 10 operation buttons: **CLEAR**, **ENTER**, **ADD**, **SUB**, **INC**, **DEC**, **NOT**, **AND**, **XOR**, and **OR**. There is a 2-digit hexadecimal LCD display for displaying input entries and results. In addition, there is a minus sign (**–**) which appears for negative results and an overflow indicator (**OVF**) which appears when an overflow occurs.

The entered operands and results are bytes (eight bits). The result has a ninth bit that for value 0 appears as a blank (for plus) and for value 1 as a minus sign to the left of the two hexadecimal digits that display the byte. Thus, the result is in 9-bit sign-magnitude form. The entered operands are unsigned positive values. Operands are entered one bit at a time beginning with the most significant bit. If fewer than eight bits are entered, the byte is filled with zeros to the left of the entered bits. If more that eight bits are entered, the most recently entered eight bits before either ENTER or an operation button is pushed are used.

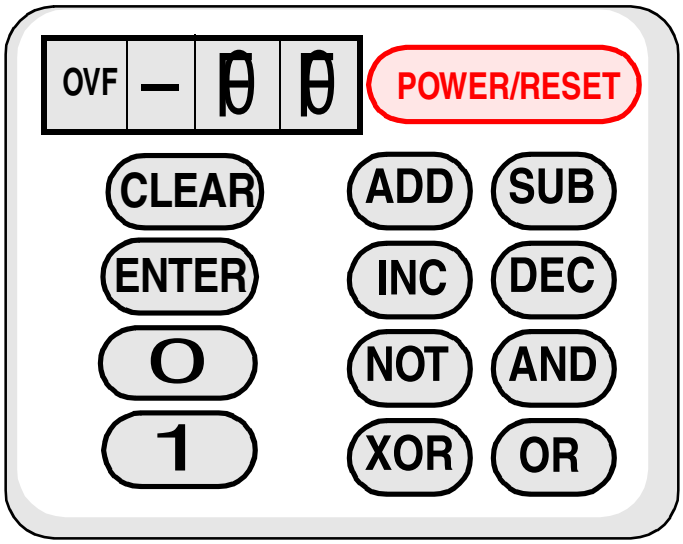

FIGURE 1
BLT - Exterior View

The BLT uses a somewhat unusual entry approach for operands and operations compared to a normal calculator. In a normal calculator, binary operations are ***infix***, i. e., an operation appears between the two operands to which it is applied. For example, in a string of calculations, to add B to the result obtained so far, the entry order is +, B. In the BLT, binary operations are ***postfix***, i. e., an operation appears after the operand to which it is applied. For example to add B to the result obtained so far, the entry order is:

B, ADD

Because of the postfix notation, after either a POWER/RESET or CLEAR operation, it is necessary to have a means for terminating the entry of the first operand entered. This termination is done by using the ENTER operation, for example:

A, ENTER

An example of entries for a binary computation with the displayed value after each entry in parentheses:

CLEAR (00), 1 (01), 0(02), 1(05), 0(0A), ENTER (0A), 1(01), 0(02), 0(04), 0(08), 0(10), SUB (– 06), 1(01), 0(02), 1(05), AND (04), DEC (011)

In detail, the buttons have the following functions:

POWER/RESET toggles the power on and off and, when the power is turned on, performs an asynchronous reset of the storage elements in the BLT.

ENTER enters the first operand to be processed.

0 enters a 0 value into an operand.

1 enters a 1 value into an operand.

CLEAR clears both the result and the input operand and returns BLT to its initial state. It is distinguished from **POWER/RESET** in two ways. It does not affect the power and the clear operation is synchronous rather than asynchronous.

ADD adds the entered operand to the result to produce a signed result and an OVF indication if overflow occurs. The result can be positive or negative.

SUB subtracts the entered operand from the result to produce a signed result and an OVF indication if overflow occurs. The result can be positive or negative.

INC adds 1 to the result to produce a signed result and an OVF indication if overflow occurs. The result can be positive or negative.

DEC subtracts 1 from the result to produce a signed result and an OVF indication if overflow occurs. The result can be positive or negative.

NOT "1's complements" the result, sets the sign to "blank," and resets the OVF indication.

AND "ands" the entered operand with the result, sets the sign to "blank," and resets the OVF indication.

XOR "exclusive-ors" the entered operand with result, sets the sign to "blank," and resets the OVF indication.

OR "ors" the entered operand with the result, sets the sign to "blank," and resets the OVF indication.

The top level of the design of the BLT appears in Figure 2.

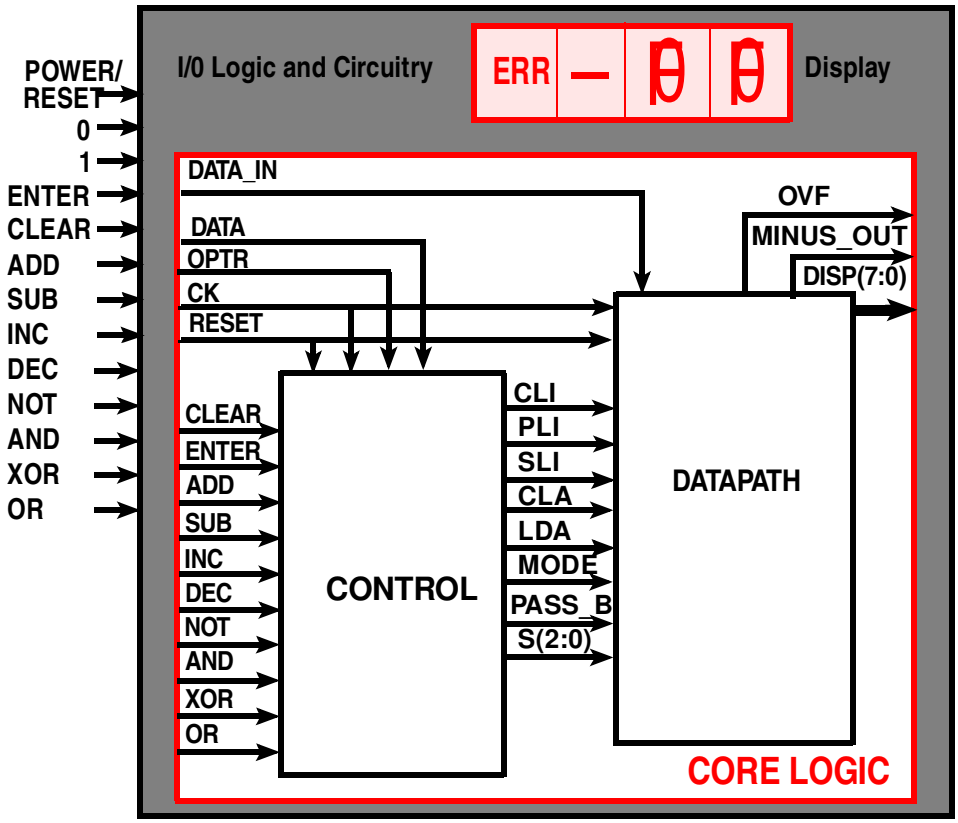

FIGURE 2
BLT - CORE LOGIC

In Project 2, you will complete the design of the **CORE LOGIC**. The **I/O Logic and Circuitry** is specified functionally as follows and **is not to be designed!** This description of the interface to the outside world will give you a sense of what is required to interface to the outside world and permit you to understand the environment for the **CORE LOGIC**.

The external pushbuttons and the display connect to the I/O Logic and Circuitry which does the following:

1) Provides a clock signal, **CK**. This clock operates much faster than the entries occur.

2) Provides a master reset signal, **RESET**, that is activated when the power is turned on.

3) Debounces and otherwise conditions signals from the pushbuttons,

4) Produces a signal DATA or a signal OPTR when specific pushbuttons are released. These signals are synchronized with CK and last for **exactly one clock cycle** from a positive edge of CK. DATA is active (=1) for 0 or 1 pushbuttons, and OPTR is active (= 1) for the eight operation pushbuttons ADD through OR.

5) Produces DATA_IN values 0 and 1 that are synchronous with the CK and valid for exactly one clock cycle from a positive edge of CK. These signals are valid only when DATA is active (=1).

6) Produces core inputs CLEAR, ENTER, and ADD through OR that are synchronous with CK and are valid for exactly one clock cycle from a positive edge of CK. ADD through OR values are valid only when OPTR is active (= 1).

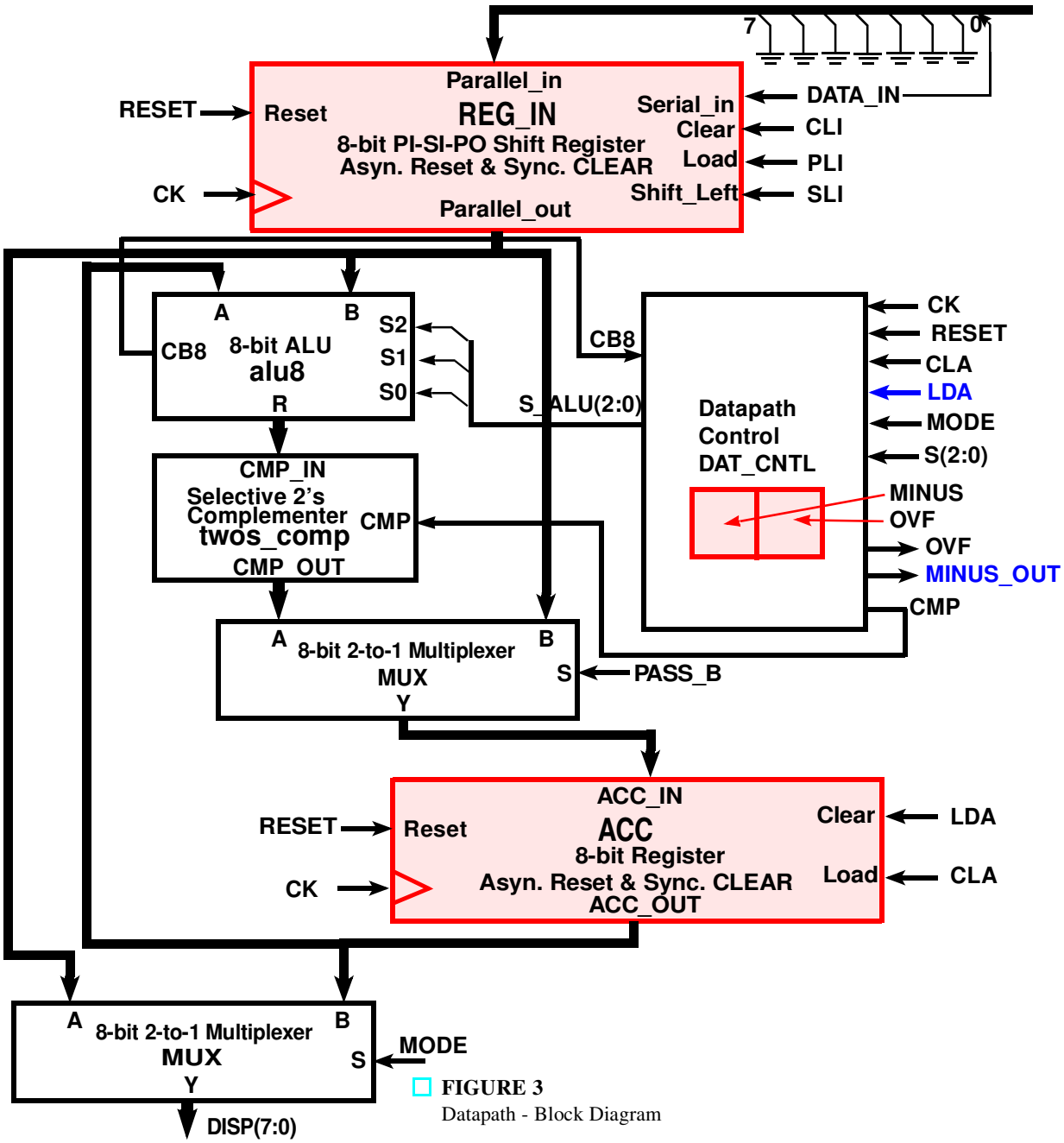

FIGURE 3
Datapath - Block Diagram

7) 5) Takes the 8-bit output **DISP(7:0)** from the **CORE LOGIC** and provides the display decoders and drivers for driving the two hex-digit displays, and

8) takes MINUS_OUT and OVF from the CORE LOGIC and provides the display drivers.

The **CORE LOGIC** is divided into two pieces, the **DATAPATH** and the **CONTROL**. You will design the **DATAPATH** in Project 2, and the **CONTROL** and **BLT CORE LOGIC** in Project 3.

DATAPATH SPECIFICATION

In this part of the project, you are to design the byte-wide DATAPATH for the BLT. Note that the DATAPATH contains seven components: two registers with different functions: **REG_IN** and **ACC**, an 8-bit ALU: **alu8**, a 2's complementer: **twoscomp**, two copies of an 8-bit 2-to-1 multiplexer: **MUX**, and the datapath control, **DAT_CNTL**. Within the datapath control are two flip-flops: **MINUS** and **OVF**, which store the sign and overflow signals, respectively. Clearly, your design should be hierarchical with these seven components interconnected to form the **DATAPATH**. In designing these components, you may use any of the components available in the Mentor **genlib** library; by looking for components already available in the library or subcomponents that can be used to easily build a component, you can save yourselves considerable effort. Next, the function of each of the datapath components is specified.

MUX – 8-bit 2-to-1 Multiplexer:

This multiplexer is an 8-bit version of the 2-to-1 line multiplexer shown in Figure 3-21, p. 121 of Mano and Kime. For PASS_B = 1, input B is placed on Y. Otherwise, input A is placed on Y.

alu8 – 8-bit ALU with 8 Functions:

The ALU designed in Project 1.

twos_comp – Selective 8-bit 2's Complementer:

For CMP = 1, the 2's complement of CMP_IN is placed on CMP_OUT. Otherwise, CMP_OUT = CMP_IN.

ACC - 8-bit Register with Asynchronous Reset and Synchronous Clear:

The function of this register consisting of eight D flip-flops is described in the following table.

OPERATION	RESET	CLA	LDA	CK	ACC_OUT (after operation)
Asynchronous Reset	1	—	—	—	**00000000**
Synchronous Reset	0	1	—	Positive Edge	**00000000**
Load	0	0	1	Positive Edge	**ACC_IN(7:0)**
Hold	0	0	0	Positive Edge	**ACC_OUT(7:0)**

Note that all eight possible combinations of RESET, CLA, and LDA are included in the table. RESET is the only active operation that can be performed without use of the Positive Edge on CK.

REG_IN - 8-bit Shift Register with Asynchronous Reset, Synchronous Clear, Load, and Shift Left Operations:

The function of this register consisting of eight D flip-flops is described in the following table.

Note that all sixteen possible combinations of RESET, CLI, and PLI, and SLI are included in the table. RESET is the only active operation that can be performed without use of the Positive Edge on CK.

OPERATION	RESET	CLI	PLI	SLI	CK	ACC_OUT (after operation)
Asynchronous Reset	1	—	—	—	—	**00000000**
Synchronous Reset	0	1	—	—	Positive Edge	**00000000**
Load	0	0	1	—	Positive Edge	**ACC_IN(7:0)**
Shift Left	0	0	0	1	Positive Edge	**ACC_OUT(6:0), Serial_in**
Hold	0	0	0	—	Positive Edge	**ACC_OUT(7:0)**

DAT_CNRL – Datapath Control:

The datapath control contains combinational logic and two flip-flops. The flip-flops store the sign of the current result (MINUS) and the overflow status (OVF) of the most recent operation (OVF). In addition to the logic to drive these two flip-flops, the datapath control also determines the following: 1) whether to complement the output of **alu8** and 2) the value of the result sign (blank = 0, – = 1) to be passed to the display.

COMBINATIONAL LOGIC The actual ALU operation to be performed, whether or not the result from the ALU is to be 2's complemented, the result sign, and the overflow indication are all defined by the rules of sign-magnitude operations with the sign of the "B" operand fixed to a positive value. For example, if the button pushed is ADD (S = 000), and the "A" operand in ACC is negative, then the actual ALU operation is SUB (S_ALU = 001) causing a subtraction of magnitude "B" from magnitude "A" to be performed. In this case, if a borrow occurs (CB8 = 1), the result must be complemented (CMP = 1), the result sign will change, i. e., be positive (MINUS = 0), and overflow is impossible. A similar analysis can be used to complete the table entries for the arithmetic operations. For all logic operations, by definition, the actual ALU operation performed is unaffected by MINUS, and CMP = 0, SIGN = 0, and OVERFLOW = 0.

S(2:0)	MINUS	CB8	S_ALU(2:0)	CMP	MINUS(t+1) = SIGN	OVF(t+1) = OVERFLOW
0 0 0	0	0	0 0 0	**0**	**0**	**0**
0 0 0	0	1	0 0 0	**0**	**0**	**1**
0 0 0	1	0	0 0 1	**0**	**1**	**0**
0 0 0	1	1	0 0 1	**1**	**0**	**0**
0 0 1	0	0	0 0 1			
0 0 1	0	1	0 0 1			
0 0 1	1	0	0 0 0			
0 0 1	1	1	0 0 0			
0 1 0	0	0	0 1 0			
0 1 0	0	1	0 1 0			
0 1 0	1	0	0 1 1			
0 1 0	1	1	0 1 1			
0 1 1	0	0	0 1 1			
0 1 1	0	1	0 1 1			
0 1 1	1	0	0 1 0			
0 1 1	1	1	0 1 0			
1 Y1 Y2*	0	0	1 Y1 Y2			

*Y1 Y2 take on values 0 0, 0 1, 1 0, and 1 1, respectively for NOT, AND, XOR, and OR.

MINUS AND OVF FLIP-FLOPS These flip-flops have the same functional properties as those used in ACC. For example, the MINUS flip-flop is specified in the following table.

OPERATION	RESET	CLA	LDA	CK	MINUS (after operation)
Asynchronous Reset	1	—	—	—	**00000000**
Synchronous Reset	0	1	—	Positive Edge	**00000000**
Load	0	0	1	Positive Edge	**SIGN**
Hold	0	0	0	Positive Edge	**MINUS (before operation)**

In addition to the MINUS flip-flop, there is logic to control the display of MINUS. The signal MINUS_OUT is 0 for MODE = 0 and is MINUS for MODE = 1.

DATAPATH DESIGN

Use the specifications above to obtain the data path components and the datapath. If a simulation of any component designed indicates an OVF, rework the design until it functions correctly. **Any schematics or traces that are not easily readable with be given 0 points!**

MUX – 8-bit 2-to-1 Multiplexer:

Design: Locate in **genlib** or design from parts in **genlib** the 8-bit 2_to_1 Multiplexer. Generate a symbol only if needed.

Validation: Do only if you did a design.

Submission: Submit the schematic only if you did a design.

alu8 – 8-bit ALU with 8 Functions:

Design: The ALU designed in Project 1. Generate a symbol for the ALU design.

Validation: None.

Submission: None.

twos_comp – Selective 8-bit 2's Complementer:

Design: Do this design by designing a bit cell module CMP_CELL such the bit cell module can be used with a ripple carry between cells for form the 8-bit selective 2's Complementer. The input to the cell should be CMP_IN, CMP, CMP and CARRY_IN and the outputs from the cell should be CARRY_OUT and COMP_OUT. The CARRY_IN to the least significant cell in the ripple carry structure should be 0 and the CARRY_OUT from the most significant cell should be ignored.

Validation: Use Quicksim simulation to validate your design. Apply the following combinations to your cell and provide annotated trace outputs.

CMP_IN	CMP	CMP_OUT (expected)
F F	0	F F
F F	1	0 1
0 0	0	0 0
0 0	1	0 0

CMP_IN	CMP	CMP_OUT (expected)
0 1	0	0 1
0 1	1	F F

Submission: Your schematic for CMP_CELL, Your schematic for twos_comp, the output traces from your simulation annotated to show correctness.

ACC - 8-bit Register with Asynchronous Reset and Synchronous Clear:

Design: Design a bit cell, ACC_CELL, of the register by using sequential circuit design techniques. Use a D flip-flop with asynchronous reset from **genlib** as the storage element. Generate a symbol for ACC_CELL and combine 8 cells in parallel to form the register ACC.

Validation: Apply the following **sequence** of operations to ACC using Quicksim simulation. Annotate the simulation output to show that it is correct.

OPERATION	RESET	CLA	LDA	CK	ACC_OUT (after operation)
Load 111111111	0	0	1	0->1	**11111111**
Asynchronous Reset	1	0	0	0	**00000000**
Hold	0	0	0	0->1	**00000000**
Load 11111111	0	0	1	0->1	**11111111**
Hold	0	0	0	0->1	**11111111**
Synchronous Reset	0	1	1	0->1	**00000000**
Load 11111111	0	0	1	0->1	**11111111**
Load 01010101	0	0	1	0->1	01010101
Load 10101010	0	0	1	0->1	10101010

Submission: Your ACC_CELL schematic, your ACC schematic. Do NOT submit simulation traces.

REG_IN - 8-bit Shift Register with Asynchronous Reset, Synchronous Clear, Load, and Shift Left Operations:

Design: Copy the contents of your ACC_CELL into a sheet REG_IN_CELL and add additional logic, and add and rename inputs and outputs to add the shift left function. Generate an ACC_CELL symbol. Combine 8 ACC_CELLS to form REG_IN.

Validation: Apply the following **sequence** of operations to REG_IN using Quicksim simulation. Annotate the simulation traces to show that it is correct.

OPERATION	RESET	CLI	PLI	SLI	CK	ACC_OUT (after operation)
Load 111111111	0	0	1	1	0->1	**11111111**
Asynchronous Reset	1	0	0	0	0	**00000000**
Hold	0	0	0	0	0->1	**00000000**
Load 11111111	0	0	1	1	0->1	**11111111**

OPERATION	RESET	CLI	PLI	SLI	CK	ACC_OUT (after operation)
Hold	0	0	0	0	0->1	**11111111**
Synchronous Reset	0	1	1	1	0->1	**00000000**
Load 11111111	0	0	1	1	0->1	**11111111**
Load 01010101	0	0	1	1	0->1	01010101
Load 10101010	0	0	1	1	0->1	10101010
Shift Left (Serial_in = 1)	0	0	0	1		
Shift Left(Serial_in = 0	0	0	0	1		

Submission: Your REG_IN_CELL schematic, your REG_IN schematic, and your annotated simulation traces.

DAT_CNTL – Datapath Control:

Design: Fill in the values for the six output variables in the last four columns of the table given for the Combinational Logic which is repeated at the end of this writeup for your convenience. Find equations for the logic manually or by using cafe (The equations are not very complex.) Use an ACC_CELL for the MINUS and OVF flip-flops. Draw the schematic for DAT_CNTL

Validation: Copy the force file `DAT_CNTL.force` by

```
cp ~ece352/public/DAT_CNTL.force
```

or from the course website and simulate your design with it. Annotate the simulation to show correctness of operation by indicating what should happen for each input pattern.

Submission: Completed truth table, equations, schematic, and annotated simulation traces for DAT_CNTL.

Data Path Integration

Design: Open a new sheet in **da** called **DATAPATH** and interconnect the components developed thus far to implement **DATAPATH**. Generate a symbol for **DATAPATH**.

Validation: Copy the force file `DATA_PATH.force` by

```
cp ~ece352/public/DATA_PATH.force
```

or from the course website and simulate your design with it. Annotate the simulation to show correctness of operation by indicating what should happen for each input pattern.

Submission: Your DATAPATH schematic and annotated output traces.

REPORT COVER PAGE (TO BE SUBMITTED)

ECE/COMP SCI 352

PROJECT 2 – SPRING 2001

TEAM MEMBERS:

1. NAME _____
2. NAME _____

TABLE OF CONTENTS

All items submitted must be in the order given below to assist us in grading. Special points will be deducted for items that are out of order. Also special points will be deducted for unreadable schematics or simulation traces.

STUDENT NUMBERS:

1. _____ _____
2. _____ _____

TEAM EFORT REPORT (TO BE SUBMITTED)

Submit: This page as the last page of your project report. **Each row of the table containing the task contributions must sum to 100%.**

Team Member Names:	1.	2.
MUX: Design (if done)	%	%
MUX: Schematic Entry (if done)	%	%
twos_comp: Design	%	%
twos_comp: Schematic Entry	%	%
twos_comp: Validation	%	%
ACC: Design	%	%
ACC: Schematic Entry	%	%
ACC: Validation	%	%
REG_IN: Design	%	%
REG_IN: Schematic Entry	%	%
REG_IN: Validation	%	%
DAT_CNTL: Design	%	%
DAT_CNTL: Schematic Entry	%	%
DAT_CNTL: Validation	%	%
Data Path Integration: Design	%	%
Data Path Integration: Schematic Entry	%	%
Data Path Integration: Validation	%	%
Other:	%	%
Other:	%	%
Other:	%	%
TOTAL (of columns 1 and 2 must be 100%)	%	%

Comments:

DAT_CNTL TRUTH TABLE (TO BE SUBMITTED)

S(2:0)	MINUS	CB8	S_ALU(2:0)	CMP	MINUS(t+1) = SIGN	OVF(t+1) = OVERFLOW
0 0 0	0	0	0 0 0	**0**	**0**	**0**
0 0 0	0	1	0 0 0	**0**	**0**	**1**
0 0 0	1	0	0 0 1	**0**	**1**	**0**
0 0 0	1	1	0 0 1	**1**	**0**	**0**
0 0 1	0	0	0 0 1			
0 0 1	0	1	0 0 1			
0 0 1	1	0	0 0 0			
0 0 1	1	1	0 0 0			
0 1 0	0	0	0 1 0			
0 1 0	0	1	0 1 0			
0 1 0	1	0	0 1 1			
0 1 0	1	1	0 1 1			
0 1 1	0	0	0 1 1			
0 1 1	0	1	0 1 1			
0 1 1	1	0	0 1 0			
0 1 1	1	1	0 1 0			
1 Y1 Y2*	0	0	1 Y1 Y2			

*Y1 Y2 take on values 0 0, 0 1, 1 0, and 1 1, respectively for NOT, AND, XOR, and OR.