6. The Computer Project

6.1. Overview

Figure 7 presents the block diagram of the target computer. Each black box in the figure is a TTL chip. The main board on the left implements the CPU and it also contains the RAM. The extension breadboard on the right implements the Input Function (e.g., input interrupt processing). The implementation of the latter will be discussed in Section 7. This section discusses the implementation of the Main Board .

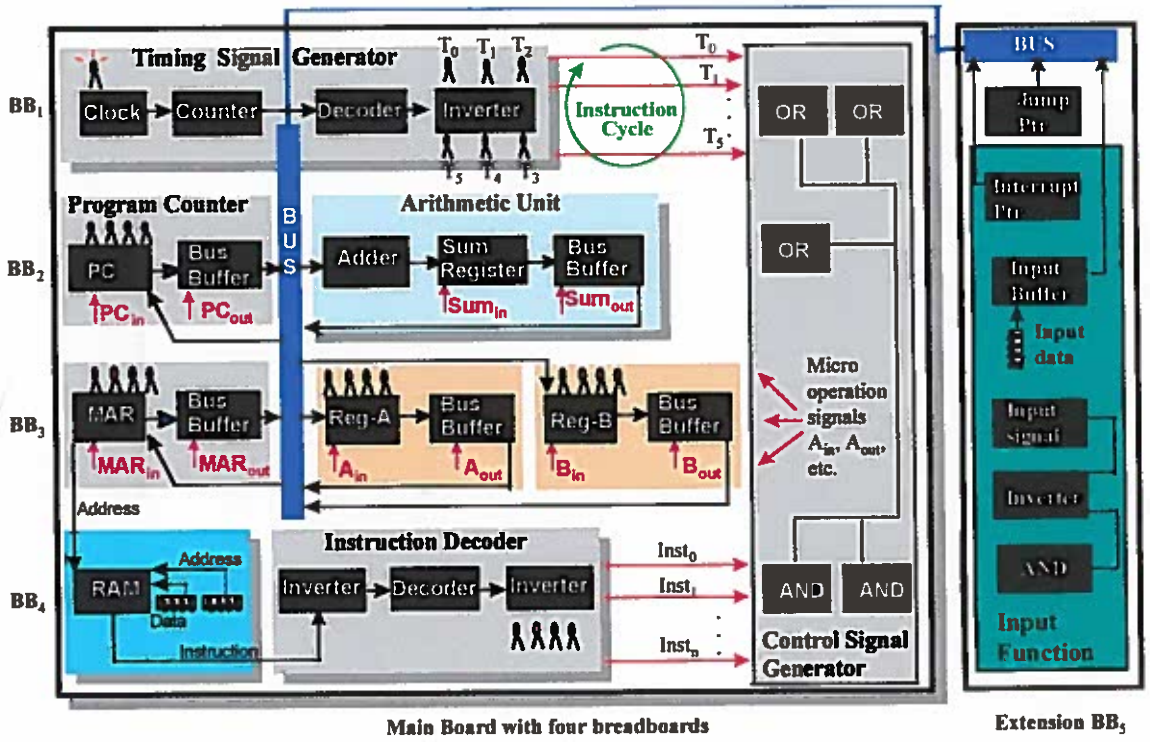


Figure 7. Block diagram of the target computer.

The Main Board consists of four breadboards BB_1 through BB_4 with independent power strips. You must produce one continuous HIGH power strip by interconnecting these individual power strips. An efficient way of doing this is by connecting the horizontal HIGH power strips in your set to the HIGH power strip in the vertical power board at the right end of your breadboard set. You do the same with the GND terminal strips.

In Figure 7, the open-ended arrows indicate control (i.e. micro-operation) signals. These are active HIGH (i.e., +5V) signals and they control the flow of data. For example Sum_{in} signal allows the

Adder output to flow into the Sum Register. Likewise, the Sum_{out} signal opens the Bus-Buffer gates towards the BUS. These control signals are generated by the Control Signal Generator.

The instructions that you will implement during this project series are tabulated below. The first three (and an optional instruction IncA) are discussed in this section. The others will be discussed in Section 7.

Instruction	Machine Code	Description
IncB	0000	Increments the contents of Register-B by one.
MovAB	0001	Moves the contents of Register-A into Register-B.
MovBA	0010	Moves the contents of Register-B into Register-A.
Jump	0011	Jumps to a specified address
Read	0100	Reads input buffer contents
Return	0101	Returns from an interrupt routine

As an instruction executes, you will be able to monitor the execution via the LED's that show the clock signal, the timing signals, the contents of the registers, the instruction signals, and other things you wish to monitor (e.g.,what is on data lines, the BUS, etc.).

An overview of the projects for building your computer is presented below. This presentation also includes how the control signals shown in Figure 7 function.

Project 1: Timing Signal Generator.
The Timing Signal Generator generates distinct time signals from T_0 through T_5 in repeating cycles. Each cycle executes one instruction and is called an instruction cycle. These time signals are generated at the rate of the clock frequency (e.g., approximately two signals per second). Timing signals are used to trigger certain operations during an instruction cycle.

Project 2 : The Program Counter and the Arithmetic Unit.
For simplicity, the Arithmetic Unit performs only a single operation: incrementation. The adder receives the data to be incremented from the BUS, increments it by one, and delivers the sum to its output pins. The SUM_{in} signal stores the adder output in the Sum Register, and the Sum_{out} signal opens the buffer gates allowing the sum to flow into the BUS.

The term "BUS" is a computer architecture term referring to a number of parallel conductors used by the CPU components for exchanging data. The term BUS implies that it is a shared medium. The BUS in Figure 7 consists of four parallel conductors used by several chips for exchanging data (i.e., a chip puts data on the BUS, another one takes it from the BUS). You will construct the BUS by interconnecting the internal conductor strips of the breadboards via wires.

The Program Counter (PC) chip contains the 4-bit RAM address of the next instruction to execute. It is incremented at the beginning of each instruction cycle. Incrementing the PC is implemented in two time steps:

1) At time T_0, the control signals PC_{out} and Sum_{in} are activated. The PC_{out} signal opens the buffer gates that connect to the BUS allowing the PC contents to flow into the adder, and the Sum_{in} signal stores the incremented address in the Sum Register. (The implementation is that T_0 signal connects to PC_{out} and Sum_{in} pins).
2) At time T_1, the control signals Sum_{out} and PC_{in} are activated. The Sum_{out} signal opens the buffer gates that connect to the BUS allowing the Sum-Register contents to flow into the BUS, and the PC_{in} signal allows the incremented address into the PC Register. (The implementation is that the T_1 signal connects to Sum_{out} and PC_{in} pins). The value of the PC stays the same during the rest of the instruction cycle.

Project 3: MAR, Data Register A, and Data Register B.
The Memory Address Register (MAR) chip contains the 4-bit RAM address of the currently executing instruction. It is updated at T_0, at the beginning of each instruction cycle. At T_0, it gets through the BUS the contents of the PC while PC is being incremented (e.g., T_0 signal connects to the MAR_{in} control pins of the MAR chip.

The Data Register A stores 4-bits of data. The input data pins of this register are connected to the BUS. It accepts the BUS data when the A_{in} control signal is HIGH. A buffer does not have input control and therefore the contents of Register A directly flows into its bus buffer. The A_{out} HIGH control signal opens the buffer gates allowing data flow into the BUS.

The same discussion in the above paragraph applies to Register B.

Project 4: RAM and the Instruction Decoder.
The RAM contains sixteen 4-bit data locations and it can store sixteen instructions. During programming, you manually specify these instructions and their RAM locations (i.e., the instruction address) via the DIP switches. During program execution, MAR continuously sends an instruction address to RAM which continuously outputs the addressed instruction to the Instruction Decoder.

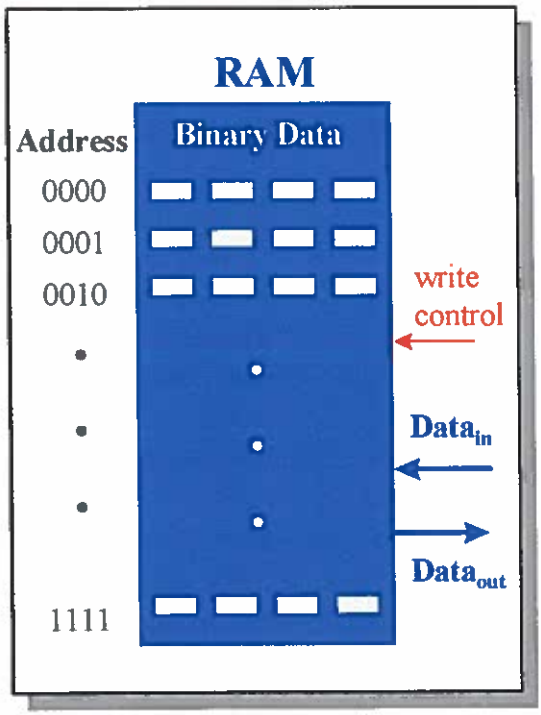
The Instruction Decoder circuit (inverter-decoder-inverter) decodes a 4-bit instruction to activate one of the six instruction signals that goes into the Control Signal Generator.

Project 5: Control Signal Generator.
The Control Signal Generator maps the incoming instruction signals and timing signals into control signals which trigger sub-operations within an instruction cycle. An instruction is executed during the T_2 and T_3 time steps of the instruction cycle. For example, the instruction IncB is executed via activating the control signals B_{out} and Sum_{in} at T_2, and activating the signals B_{in} and Sum_{out} at T_3. You should note that the contents of the MAR which is set at T_0 stays the same during the rest of the instruction cycle. Therefore, T_1 through T_5, the RAM output pins contain the same instruction and the Instruction Decoder continuously feeds into the Control Signal Generator the signal for this particular instruction.

Project 6: Interrupt Processing.
This optional project is discussed in Section 7.

Project 7: Programming and Running Your Computer

Before execution, you enter your program into **RAM** manually using the data and address DIP switches (see Figure 7). As the adjacent illustration shows, each 4-bit memory location for storing an instruction has a 4-bit address. These addresses range from 0000 to 1111. To write into RAM, you specify the memory location via the address switch, specify the instruction via the data switch, and manually send to RAM a write signal. The four pins of the data switch connects to RAM's data input pins and the four pins of the address switch connect to RAM's address pins



Turning on the power for the rest of the computer executes your machine-language software program

NOTES:
If you adhere to the chip layout shown in Figure 7, you will not run out of breadboard space. Also, plan ahead as you insert your chips horizontally: start from the very left end of a breadboard, and, in general, do not leave more than two slots between the two adjacent chips.

Please keep in mind that, when an input pin of a TTL device is not connected to a source with a HIGH or LOW value, the device by default assumes a HIGH input pin value. (Examples are: control pins PC_{in}, PC_{out}, Sum_{in}, Sum_{out}, etc., mentioned above). In other words, that an input pin is not connected to a source does not mean it is not a part of your circuit and if you overlook this you might observe unexpected circuit behavior.

The design in this manual is not binding in constructing this target-computer. You may change the design to your liking provided that you understand what you are doing. Here are some change examples:

- You can choose your own 4-bit codes for the given instructions. If you do this, you will have to redesign the wiring in the Instruction Decoder and the Control Signal Generator accordingly using the design in this manual as an example.
- Obviously, you can choose a chip layout different than that in Figure 7.
- By adding more breadboards to your set, following are some neat things you can do:
  - ◊ You can enhance your Arithmetic and Logic Unit by including other chips for division, multiplication, subtraction, comparison, etc. Accordingly you can have a larger instruction set that implements these functions.
  - ◊ You can implement instructions with data operands (e.g., a one-address computer, two-address computer, etc.)
- In short, if you are willing to put the time, with some research on additional computer-architecture concepts, you can build the functionality of a commercial computer.

6.2. PROJECT 1: The Timing Signal Generator (TSG)

The Timing Signal Generator implements the time steps of the instruction cycle (i.e., signals T_0 through T_5 in Figure 8). Figure 8 shows the layout of the TTL chips used in the Timing Signal Generator on breadboard-1. The flow diagram in the figure labels pin-to-pin connections which are detailed in the following sub-sections.

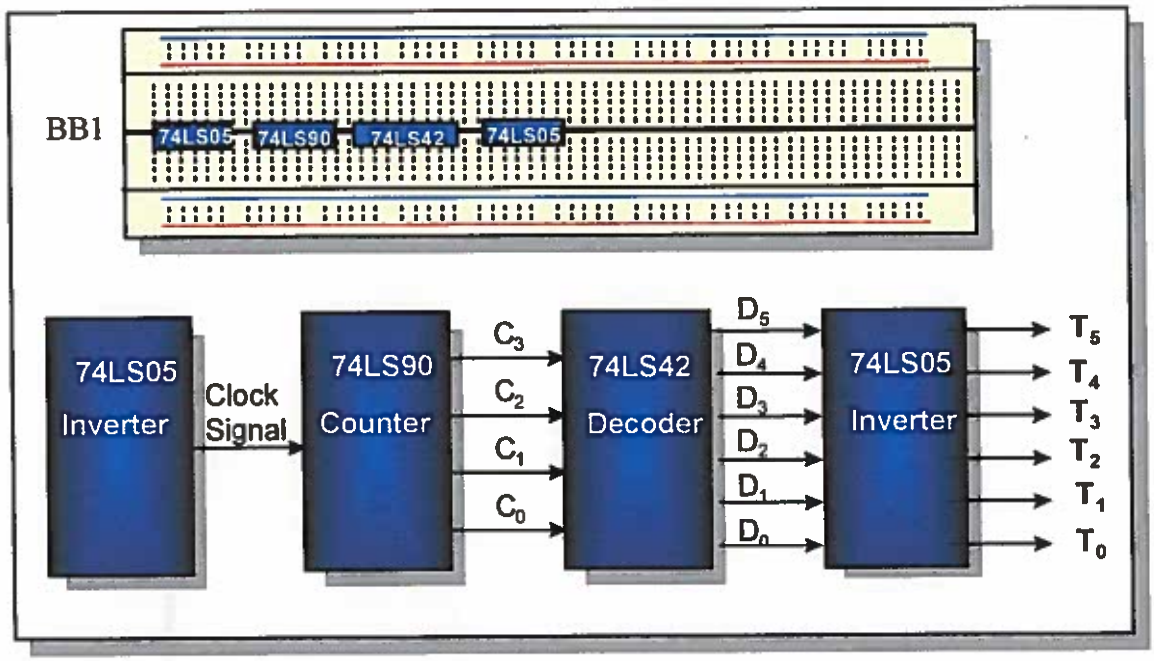


Figure 8. Timing Signal Generator (TSG).

The clock circuit is built using the TTL chip 74LS05 and generates a continuous stream of alternating signals of HIGH (i.e., +5 volts) and LOW (i.e., GND). This stream flows into the counter.

The counter chip TTL 74LS90 continuously outputs the count of the clock signals in binary which change with every clock pulse (e.g., 0000, 0001, 0010, 0011, ...). TTL 74LS90 counts from 0 to 9 and when the count reaches 9 it wraps around to 0. The counter repeats this with every ten clock pulses. In other words the duration of an instruction cycle is ten clock pulses. We shall use only the first six of these timing signals

The decoder TTL 74LS42 activates a single output pin corresponding to the binary count value it receives from the counter. For example, for count value of 0000 it activates the output pin D_0, for count value of 0001 it activates the output pin D_1 and so on. Only one of these pins is active at any one time. TTL 74LS42 produces active LOW outputs.

The inverter TTL 74LS05 contains six NOT gates and each gate converts a decoder's active LOW output signal into a HIGH signal (i.e., a timing signal).

6.2.1. Clock Circuit of the TSG

Figure 9 illustrates the clock circuit built around TTL 74LS05. For the purposes of this project, this is a vehicle to generate alternating HIGH and LOW signals and its internal workings will be ignored. Please construct the circuit via the following procedure:

1. **Build the power connections** - Insert TTL 74LS05 into breadboard-1 and connect pins 14 and 7 to HIGH and GND, respectively.
2. **Complete the pin-to-pin connections** - Each pin-to-pin connection in the figure is a slot-to-slot wire. Insert these wires for pin pairs 13-8, 12-11, and 10-9.
3. **Insert the capacitor** - One leg of the capacitor goes into a slot for pin 11, the other leg goes into a slot for pin 10.
4. **Insert the resistor** - One leg of the resistor goes into a slot for pin 8 and the other leg goes into a slot in the HIGH power strip. The preferred resistor is 1.6 kohm or 2.0 kohm, although the use of a 330 ohm resistor will result in a brighter LED signal.
5. **Monitor the clock's operation** - Insert an LED to connect a slot of pin 8 to GND. Power up the chip by connecting the power supply. The LED will start blinking at the rate of the clock pulse. To make sure that your clock is in working condition, do not remove this LED during the rest of the project.

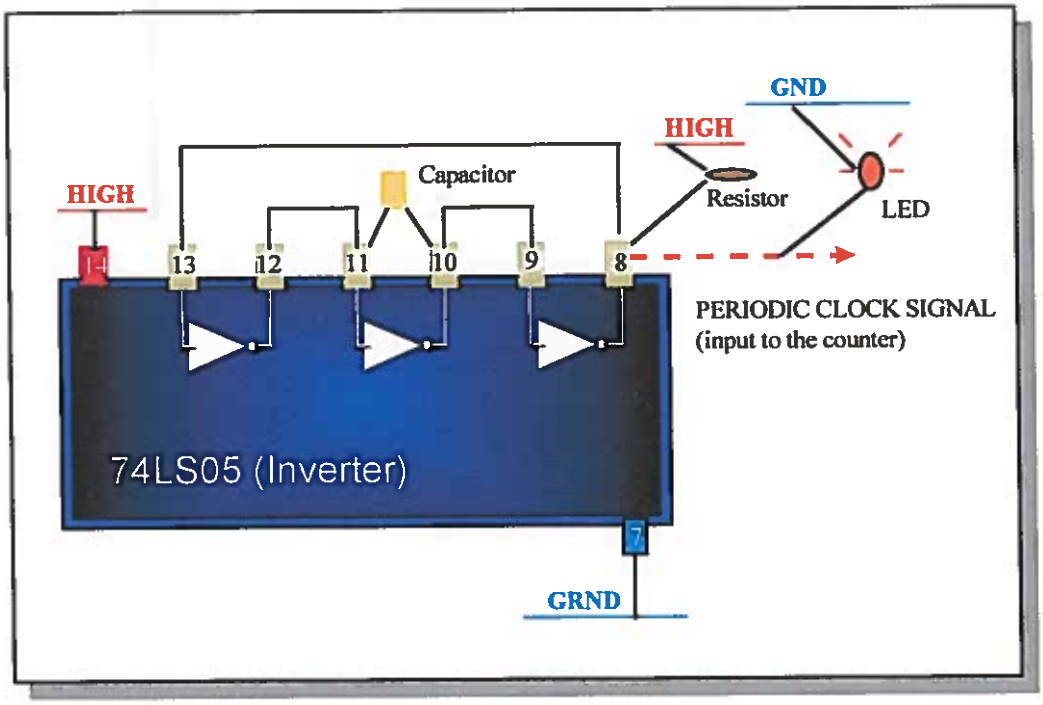


Figure 9. The Clock Circuit.

6.2.2. Counter of the TSG

Figure 10 illustrates the wiring around the counter. The construction procedure follows:

1. **Build the power connections** - Insert the chip into breadboard-1 and connect pins 5 and 10 to HIGH and GND, respectively. Also, interconnect pins 1 and 12 as shown (this is a cascade counter of two parts and this connection makes it a single counter). For the counter to operate properly, pins 2 and 7 must be grounded.
2. **Connect the clock signal** - Connect the output of the clock (pin 8 of TTL 74LS05) to the input pin of the counter (pin 14).
3. **Test the counter's operation via LEDs** - Note that the output pins of TTL 74LS90 for the binary count digits are not in positional-value sequence. For ease of observation, order the LED connections as shown in the figure such that the LED for the binary digit with the lowest positional value (i.e., Out_0) is at the right end and the LED for the binary digit with the highest positional value (i.e., Out_3) is at the left end. With this ordering the LEDs will display the count values as: 0000, 0001, 0010, 0011, ... , 1001.

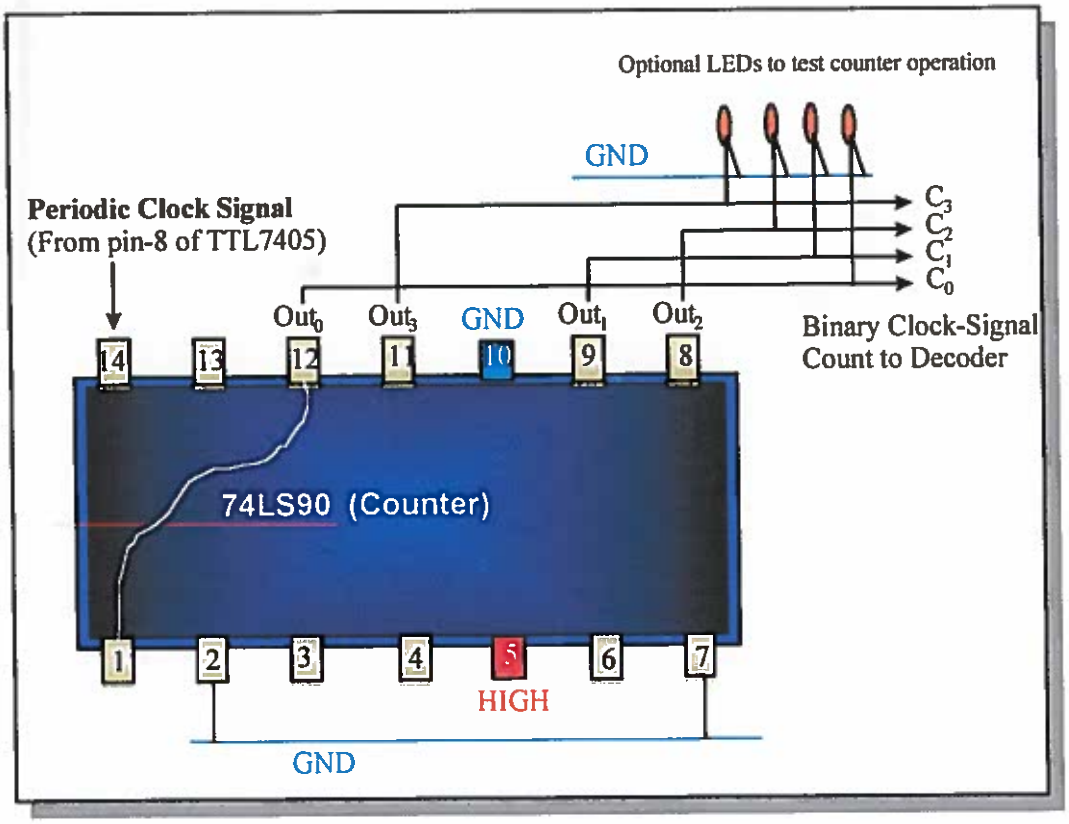


Figure 10. Counter of the TSG.

6.2.3. Decoder of the TSG

Figure 11 illustrates the connections for the decoder.

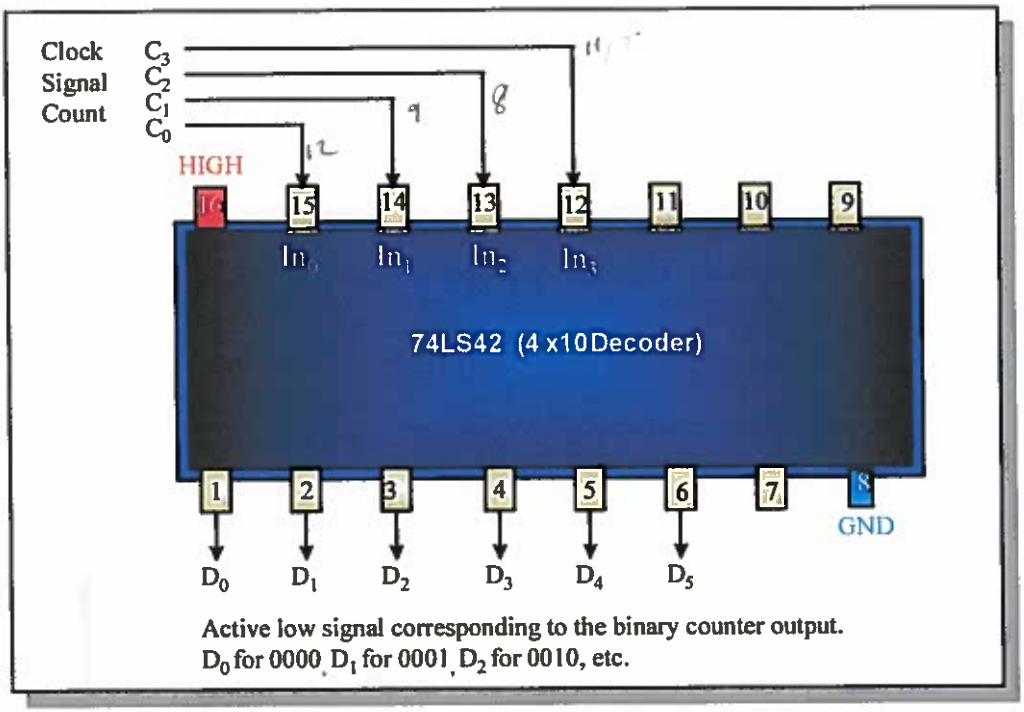


Figure 11. Decoder of the TSG.

The decoder receives its binary input from the counter. Make sure that these connections match the positional values of the pins. In other words, the counter's output-pin C_0 must connect to the decoder's input-pin In_0, the counter's output-pin C_1 must connect to the decoder's input-pin In_1, and so on.

Important note: All through the project, while you do wiring, pay specific attention to matching the positional values of data flow.

6.2.4. Inverter of the TSG

Figure 12 illustrates the final stage of the Timing Signal Generator: the inverter. The gates of the inverter TTL 74LS05 map the active LOW inputs from the decoder into active HIGH timing signals: T_0, T_1, T_2, T_3, T_4, T_5.

While the circuit is in operation, the LED's will blink in a sequence from T_0 to T_5 in cycles. Because we ignore the signals from T_6 to T_9, in each cycle there will be a gap of four clock ticks after T_5 signal.

For 74LS05 to operate properly, each output pin must be connected to the HIGH (+5V) terminal via a pull-up resistor (e.g., 1.6 kohm or 2.0 kohm) in your kit. Not to clutter the diagram, Figure 12 does not show these resistor connections. Figure 9 of the clock circuit contains an example of such a connection.

(Each gate of TTL 74LS05 is independent of the other gates. Therefore, you can choose a gate allocation to input data differently than shown in the figure).

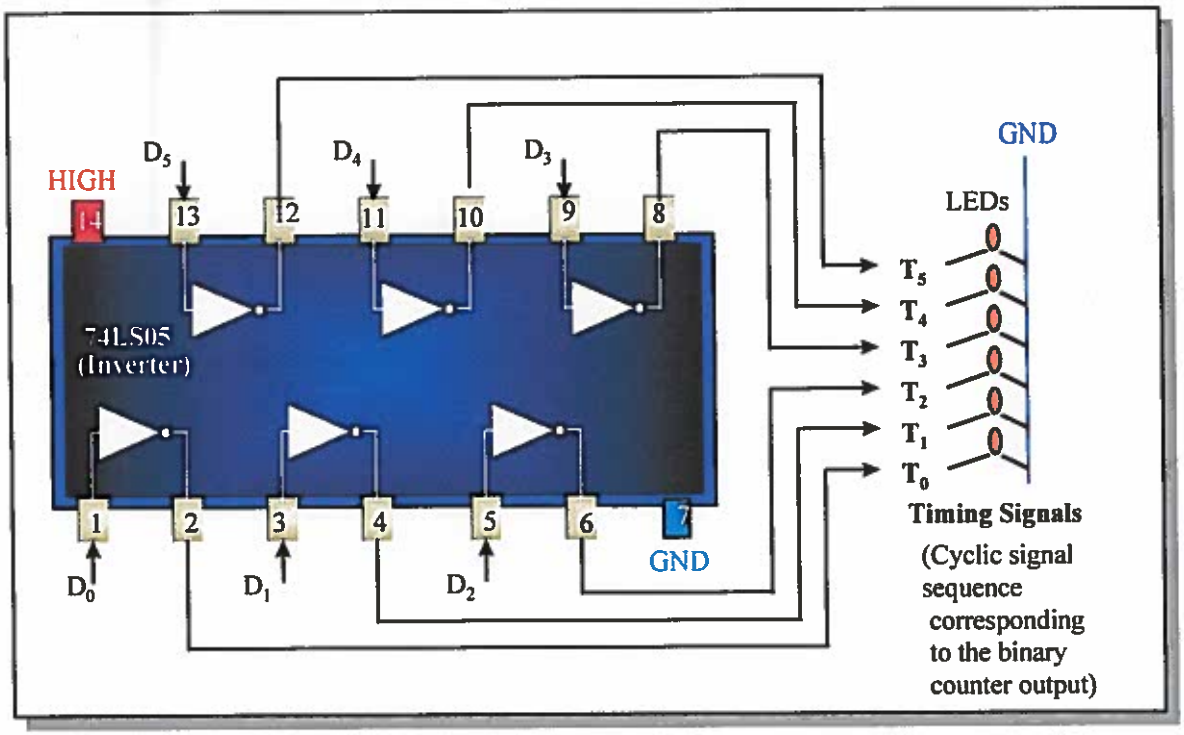


Figure 12. Inverter of the TSG.

6.3. PROJECT 2 : The Program Counter (PC) and The Arithmetic Unit

Top of Figure 13 illustrates the breadboard chip layout for the Program Counter and the Arithmetic Unit (which only performs the "increment" function). The block diagram below the breadboard shows the details. Although the PC and the Arithmetic Unit are functionally independent of each other, both are discussed in this section because the PC uses the Arithmetic Unit for incrementing its contents. This discussion also helps understand how the BUS is used.

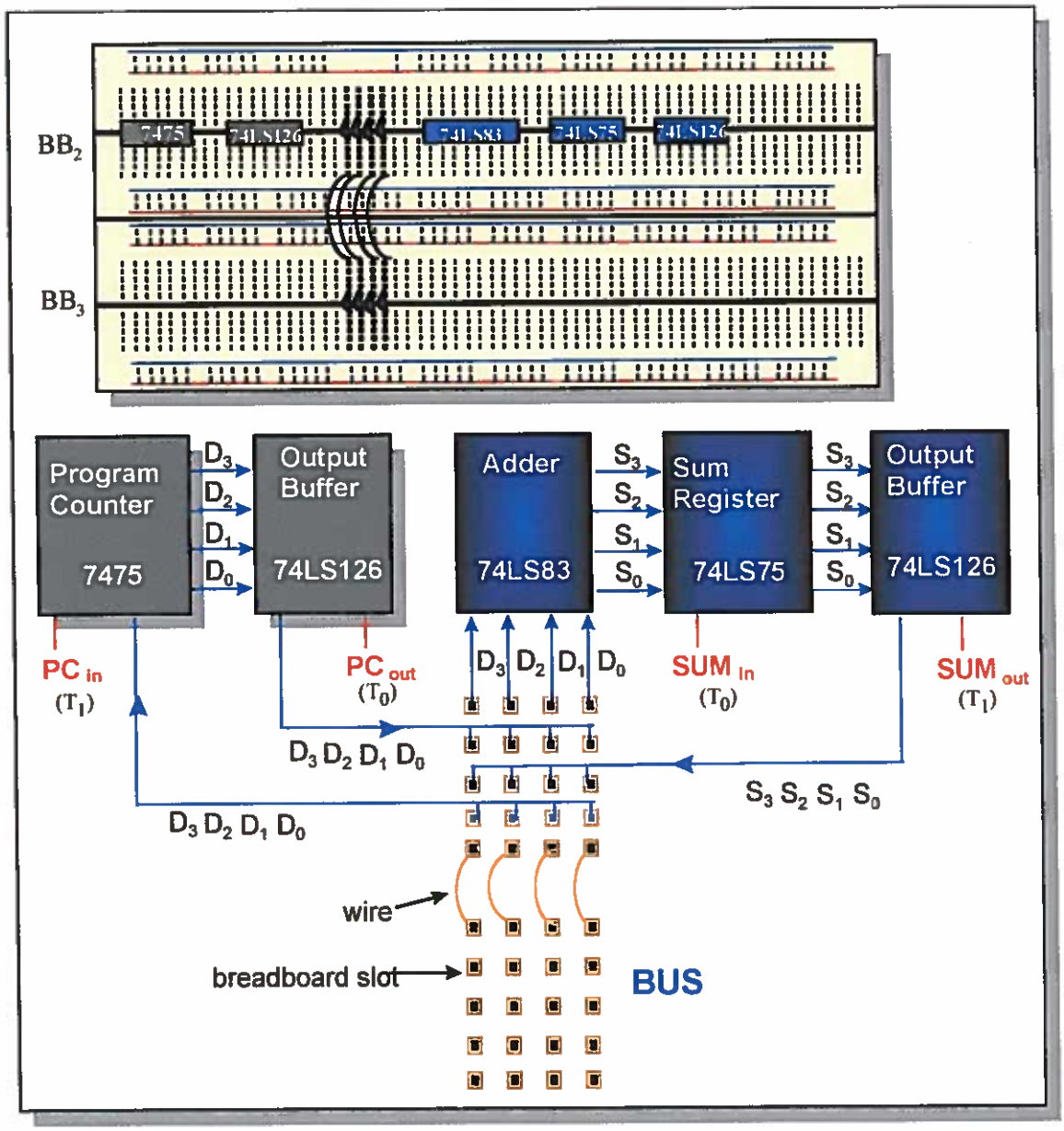


Figure 13. The Program Counter and the Arithmetic Unit.

First let's construct the BUS. The BUS consists of four parallel conductor lines vertically traversing breadboards BB2 and BB3. You construct the BUS by interconnecting the conductor strips on both sides of each breadboard's central groove, and by interconnecting the so formed bus segments across the two breadboards (see the upper part of Figure 13).

The PC and the Arithmetic Unit exchange data through the BUS. Figure 13 shows the required data-flow connections on BB2. Although utilized in the Figure 13, do not yet utilize the top-most slot row of the bus. As Figure 7 shows these slots will be used for extending the bus (through BB1 if you will) to BB5.

As mentioned earlier, the Program Counter is incremented during the first two timing signals of the instruction cycle: T_0 and T_1. The signal T_0 connects to PC_{out} and SUM_{in}. Hence, when the TSG activates T_0, the instruction address stored in PC flows through the BUS into the Adder which increments the address and passes it to the SUM Register. Likewise, the signal T_1 connected to SUM_{out} and PC_{in} opens the gates of the SUM Register's output buffer towards the BUS and enables the PC to accept the incremented address that comes through the BUS.

Here, PC_{out} and SUM_{out} are the output control pins for the 74LS126 chips (see Section 9.10), and PC_{in} and SUM_{in} are the input control pins for the 7475 and 74LS75 chips(see Section 9.6), respectively.

Testing this circuit is simply observing via the LED's that the circuit operates as described above. During T_1 of each instruction cycle the PC is incremented by 1, and, when the PC value reaches 1111, incrementing it by 1 brings its contents back to 0000 (i.e., it wraps around).

6.3.1. Program Counter and Its Output Buffer

Figure 14 illustrates the 4-bit Program Counter (TTL 7475) and its four output buffers in TTL 74LS126. The PC's four output pins labeled D_0, D_1, D_2, and D_3 carry the instruction address. These pins connect to input pins of the buffers in TTL 74LS126, and output pins of these buffers (pins 11, 8, 3, 6) connect to the BUS. Each buffer has a separate control pin which, when activated by a HIGH signal, opens the gate for the bit to flow into the BUS. The figure shows that these control pins (4, 1, 13, 10) are interconnected in a chain so that they all can be activated with a single control signal (i.e., PC_{out}). Please see the Appendix, Section 9.10, for the behavior of a TTL 74LS126 device.

The PC will accept the BUS data (i.e., the incremented address coming from the Arithmetic Unit), only when pins 13 and 4 receive a HIGH control signal (i.e., PC_{in}). To avoid duplication of this signal, please interconnect these pins as shown in the figure. Please observe in the figure that the positional values for data connections match

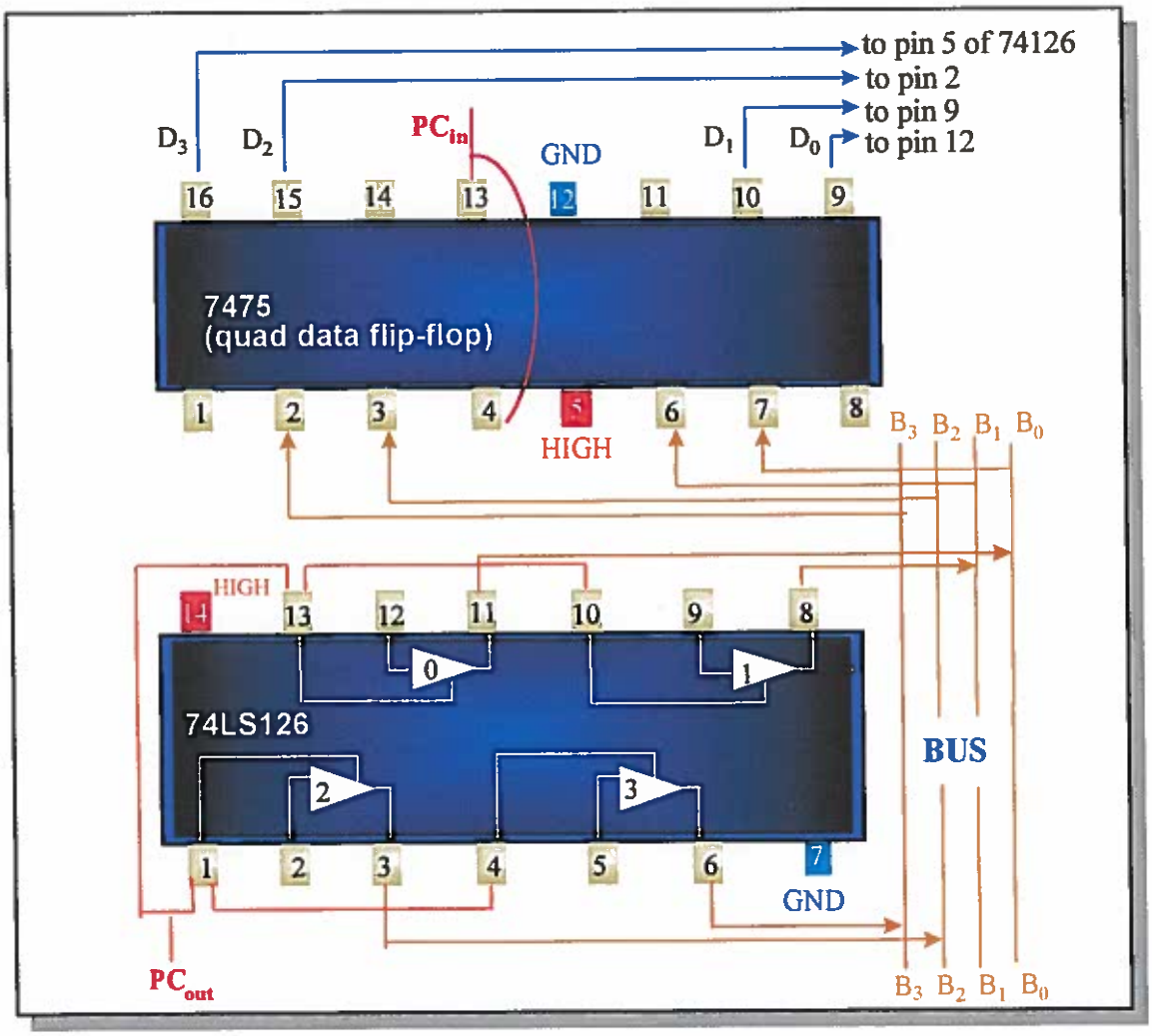


Figure 14. The Program Counter and its Buffer.

6.3.2. Adder of the Arithmetic Unit

Figure 15 shows the connections around the 4-bit adder TTL 74LS83. The adder adds three items: 4-bit "Y" input, 4-bit "X" input, and 1-bit $carry_{in}$. Because our design uses the adder only for incrementing the data incoming from the BUS, the "Y" input set of the chip is connected to GND (i.e., it continuously feeds 0000), the "X" input set is connected to the BUS, and the $carry_{in}$ bit which by default has a value of 1 (i.e., HIGH) is left open. With these input values, the adder increments the "X" input by adding the $carry_{in}$ to it.

The output of the adder are the SUM bits (S_0, S_1, S_2, S_3) and as discussed in the next section you will connect them to input pins of the SUM Register.

led 5 to 13

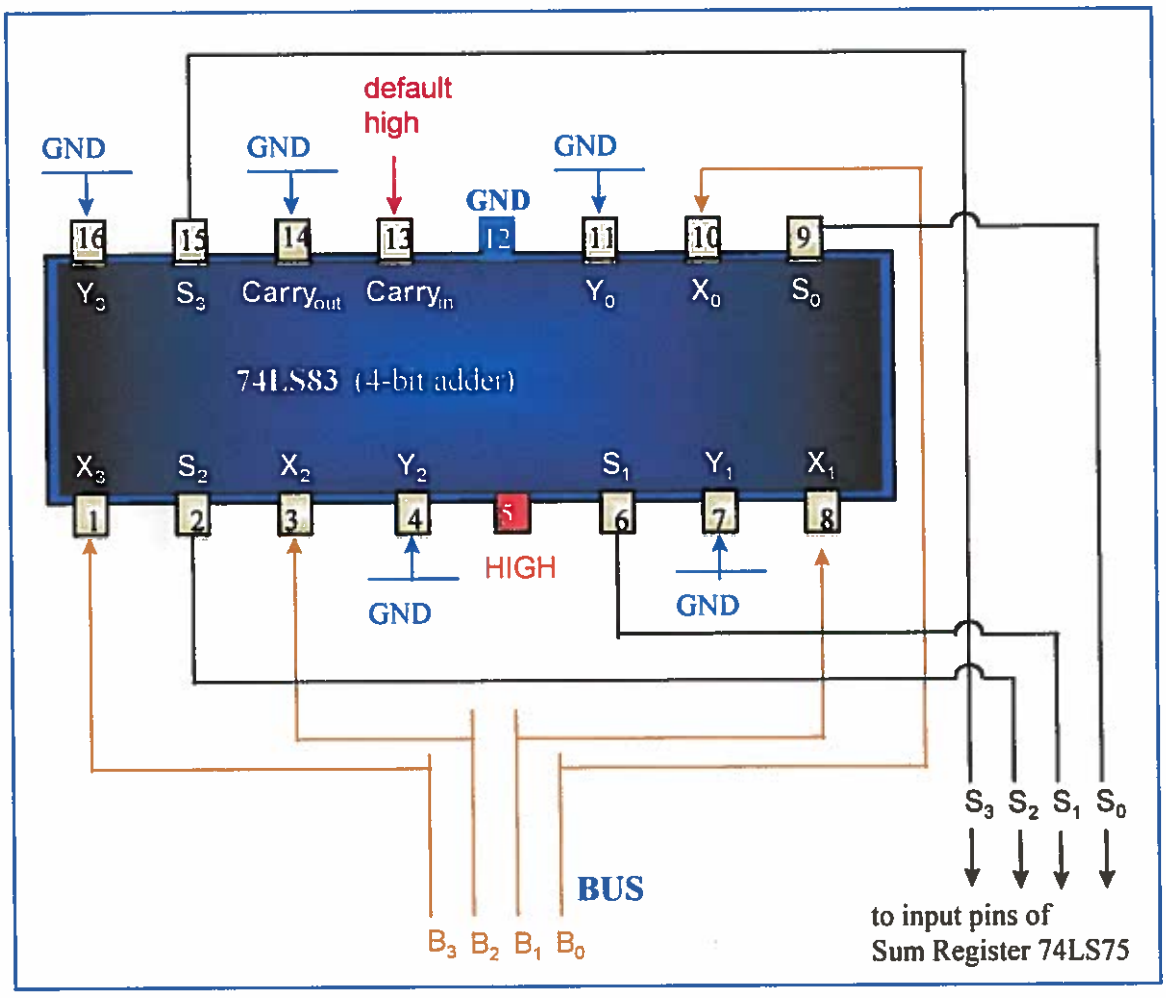


Figure 15. The Adder of the Arithmetic Unit.

6.3.3. Sum Register and Its Output Buffer

Figure 16 illustrates the SUM Register (74LS75) and its Output Buffer (74LS126). The wiring around these chips is similar to the wiring of the Program Counter and its Output Buffer in Figure 14 except that the 74LS75 gets its input from the adder but not from the BUS.

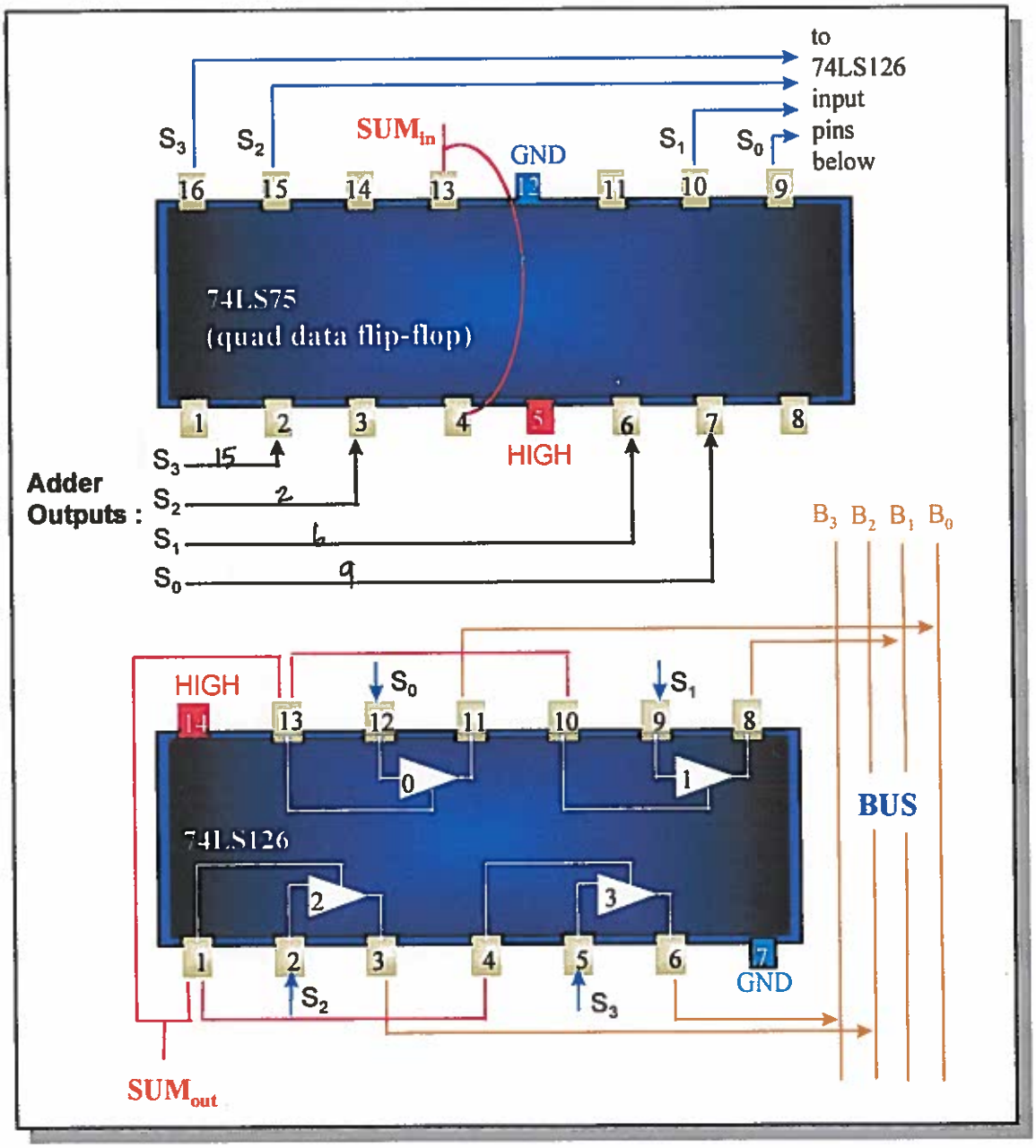


Figure 16. The Sum Register and its Output Buffer.

6.4. PROJECT 3: MAR, Data Register A, and Data Register B

Top of Figure 17 illustrates the breadboard layout for the MAR, Data Registers A, and Data Register B. Each of these registers receives its input directly from the BUS, and puts its output into an associated buffer (74LS126) that connects to the BUS. The BUS-connection wiring for each of these register-buffer pairs is similar to the wiring of the Program Counter and its Output Buffer. The bottom of the figure illustrates this for two of the three mentioned registers.

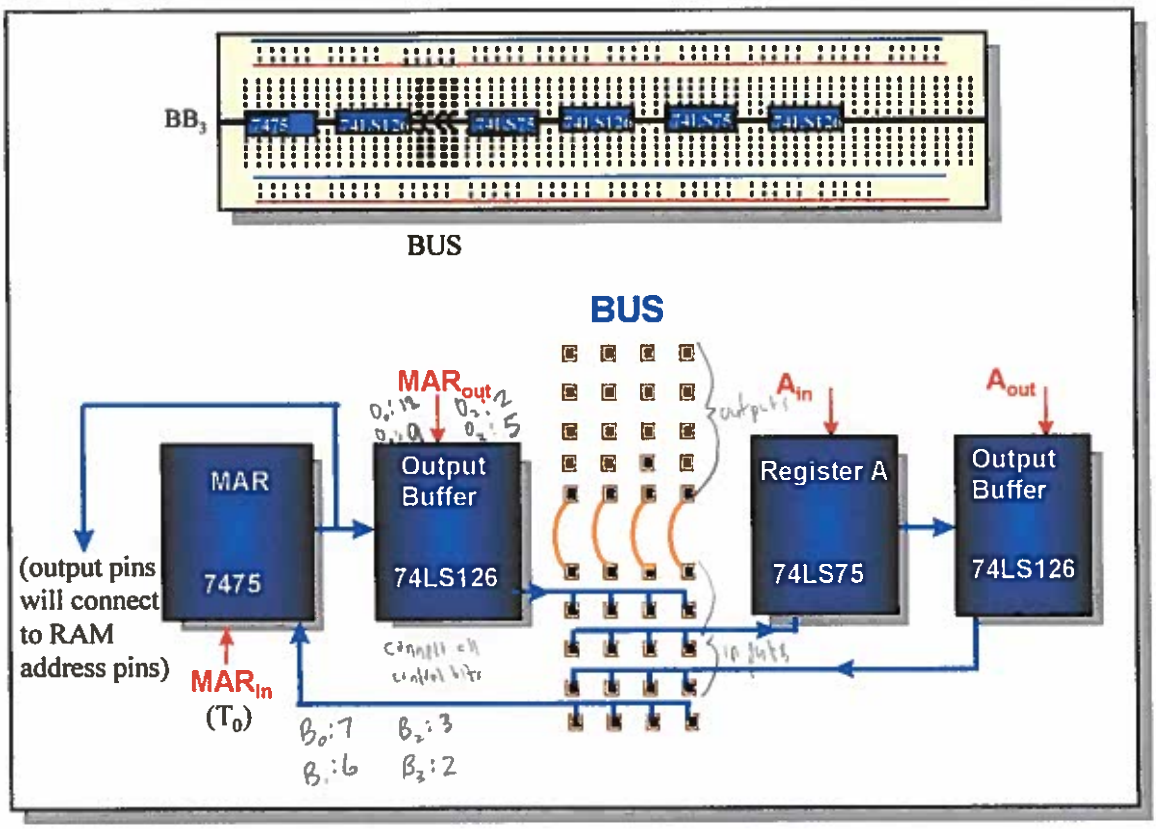


Figure 17. MAR, RegisterA, and RegisterB.

At this point of the project, the control signals MAR_{out}, A_{out}, and B_{out} must be grounded to disconnect the associated buffers from the BUS. Otherwise, the default HIGH value for these control pins will open the gates of these buffers interfering with incrementing the PC. Please see Section 9.10 to prohibit multiple buffers accessing the BUS simultaneously. (Later, these output control pins will be tied to their respective Control-Signal-Generator signals.)

During an instruction cycle, the MAR will contain the RAM address of the currently executing instruction. This is achieved by transferring through the BUS the contents of the PC into the MAR at T_0, before the PC is incremented. Simply connect the MAR_{in} pin to the T_0 timing signal. This way, updating the MAR will be done during the first step of incrementing the PC and it will stay the same during the rest of the instruction cycle.

Before proceeding any further, please observe via LED's that MAR behaves as described above

6.5. PROJECT 4: Random Access Memory (RAM) and the Instruction Decoder

Figure 18 illustrates the chip layout and the flow diagram for the RAM and the Instruction Decoder. Because the TTL 7489 RAM outputs the complement of the stored data, the inverter next to it reconstructs the original 4-bit data. The decoder activates the particular instruction signal corresponding to its 4-bit input, and, the inverter next to the decoder inverts the active LOW output signal of the decoder into an active HIGH instruction signal.

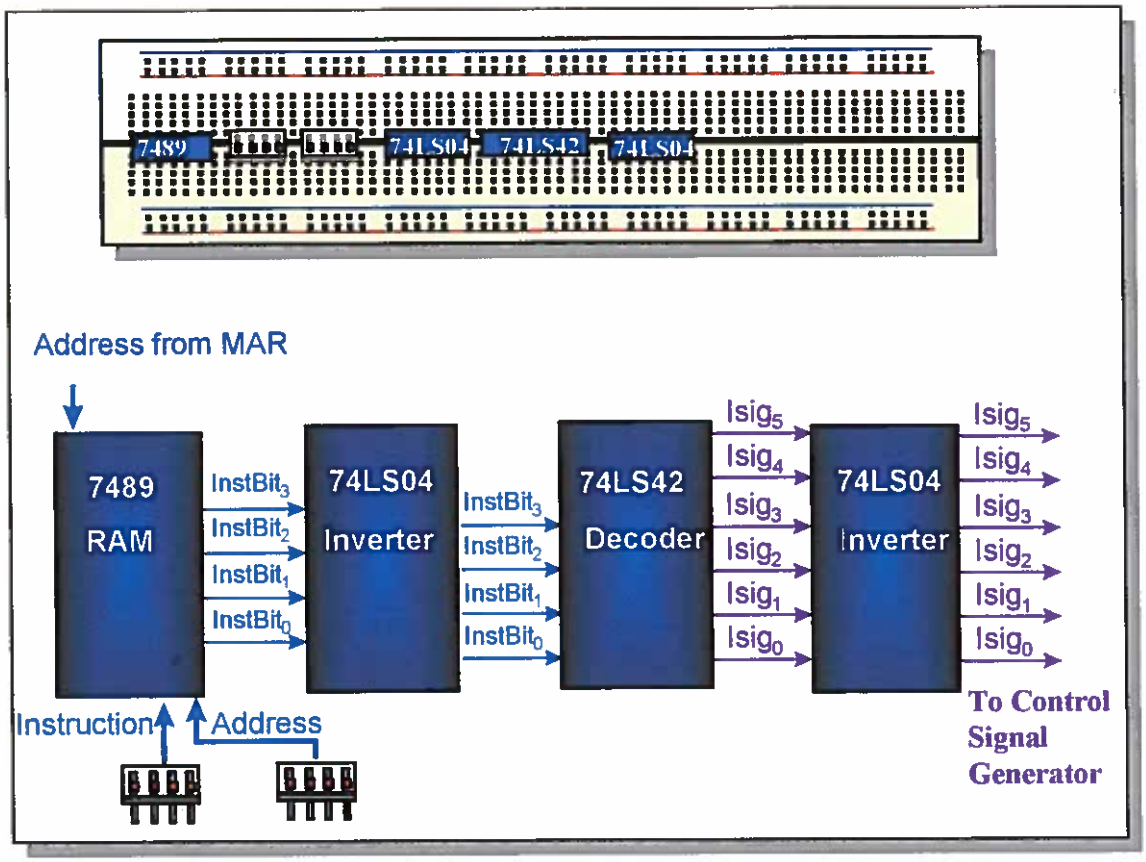


Figure 18. The Random Access Memory and the Instruction Decoder.

While manually programming the RAM, both the instruction and the address values are entered via the illustrated DIP switches. During program execution, RAM will get the instruction address from the MAR. The complement of the 4-bit data at the addressed RAM location is continuously present at TTL 7489's output pins (e.g., $InstBit_0$, $InstBit_1$, $InstBit_2$, $InstBit_3$).

The purpose of the instruction decoder (TTL 74LS04 - TTL 74LS42 - TTL 74LS04) is to activate one of the six one-bit instruction signals (i.e., one of $Isig_0$, $Isig_1$, $Isig_2$, $Isig_3$, ...) for the 4-bit instruction that it receives from RAM. Note that among the sixteen possible instruction codes we only use six codes (i.e., 0000, 0001, 0010, 0011, 0100, 0101) for instruction implementation. The other ten operation codes (i.e., 0110 through 1111) are NULL instructions and will not trigger any operation.

This paragraph is a detailed explanation of why we use the inverters shown in Figure 18. Due to its internal design, each output bit of this RAM is the complement of the stored bit (i.e., 0 is output as 1, and 1 is output as 0). For example, if the instruction you store is 0011, it appears at RAM's output pins as 1100. Therefore, the first chip of the Instruction Decoder inverts the instruction bits to their original values and feeds them to the decoder chip. The decoder TTL 74LS42 maps its 4-bit input to one of the instruction signals. However, the output of TTL 74LS42 is also active low (i.e., it puts a LOW signal on the pin that it chooses) and it needs to be inverted into a HIGH signal. The second inverter chip in the figure does this inversion. The instruction signal so produced is utilized by the Control Signal Generator to generate the appropriate control signals in implementing the instruction.

6.5.1. The Random Access Memory Circuit

Figure 19 illustrates the wiring for the RAM. The figure includes the TTL 7489, two DIP switches, and an optional toggle switch.

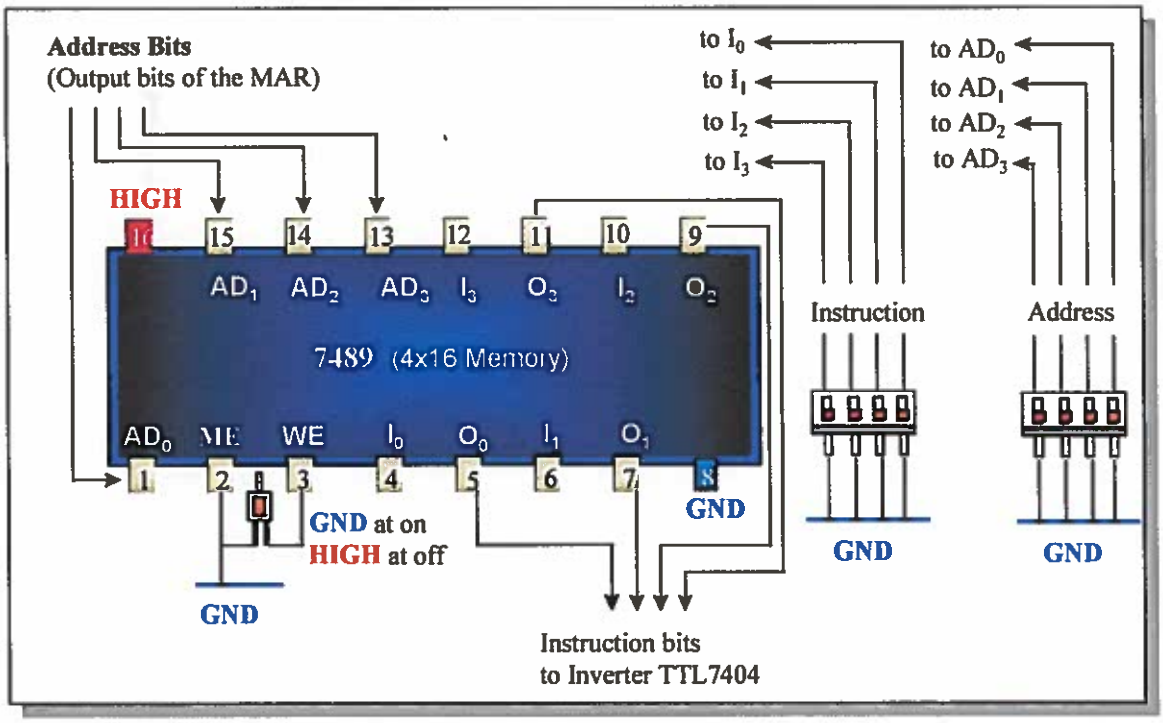


Figure 19. The RAM Circuit.

The address pins (AD_0, AD_1, AD_2, AD_3) and the data input pins (I_0, I_1, I_2, I_3) of RAM connect to the DIP switches for manual entry of program instructions. The address pins of RAM also connect to the output pins of the MAR. It is important to remember that the address from the MAR and the address from the DIP switches cannot be simultaneously active. (Therefore, when manually programming the RAM we shall power up only the RAM board so that only the address DIP switches are in effect. Likewise, during program execution, we shall turn off all the address switch elements so that only the MAR is in effect.)

The pin labeled ME is for enabling RAM and should always be grounded. The WE (write enable) pin enables writing into RAM when it is 0 and enables reading from RAM when it is 1. As in most TTL chips, the default value of this pin is 1. Hence, as soon as it is powered up the chip is in the Read mode - i.e., the contents of the addressed RAM location appears at the RAM's output pins. To write into RAM, momentarily ground the WE pin. For this, you can use a toggle switch as shown in Figure 19 or you can simply connect to pin-3 an open-ended wire which you momentarily dip into GND when you want to write into RAM.

Using the DIP switches is a little bit tricky. A DIP switch element is wired such that it is between a RAM pin and GND. Setting a switch element to OFF breaks the circuit and the RAM pin assumes the default value of ON (i.e., HIGH). Likewise, setting a switch element to ON closes the circuit and sets the RAM pin to OFF. In other words, the ON or OFF value of a RAM pin will be the opposite of the ON or OFF printed-label of its switch element. For example, for the address or data value of 0001, you set the associated switch elements to: ON, ON, ON, OFF.

You can test your circuit by storing various values into RAM locations and monitoring RAM outputs by using one of two techniques:

A. Connect the RAM output pins to an inverter (TTL 74LS04) and monitor the inverter's outputs via LED's. This is a preferred technique because you will have to wire an inverter for the Instruction Decoder anyway.
B. Use an LED and a resistor for each RAM output pin and keep in mind that each RAM output pin outputs the complement of the stored data bit.

6.5.2. The Instruction Decoder

Figure 20 illustrates the connections for the instruction decoder. Note that the input data to pins 1, 3, 5, and 9 of TTL 74LS04 at the top are the RAM output bits, and the output data from pins 2, 4, 6, and 8 of the lower TTL 74LS04 (i.e., the instruction signals) go to the Control Signal Generator. (Note that, not to clutter the diagram, the decoder-inverter pair at the bottom of Figure 20 shows connections for not six but for four instructions only).

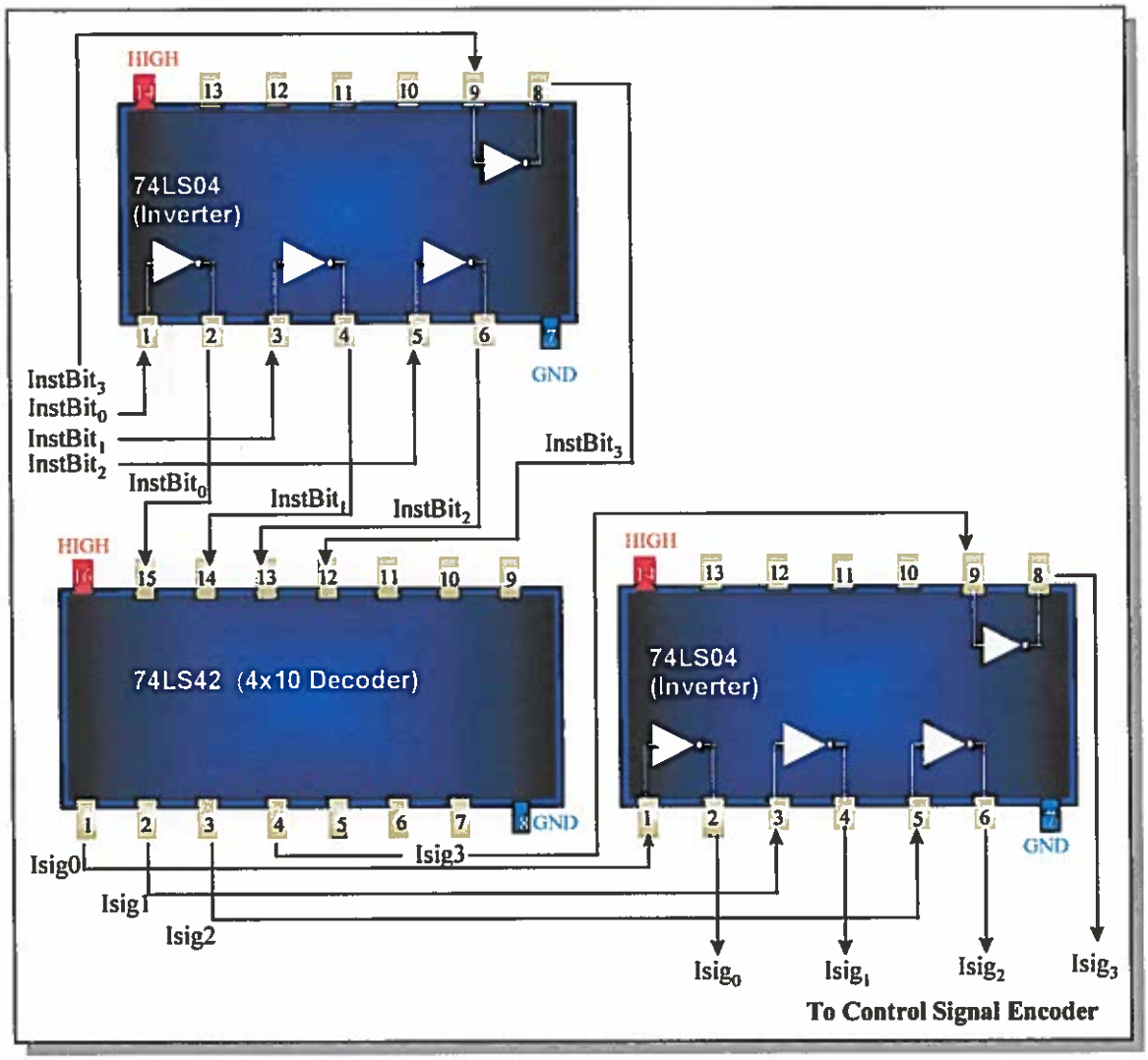


Figure 20. The Instruction Decoder.

Test the Instruction decoder by verifying that it activates the instruction signal for the associated RAM entry. For example, write an instruction into RAM (e.g., 0000 for IncB) by setting the data switch to 0000 and setting the address switch to a RAM location address (e.g., 0000), and momentarily ground the WE pin. If you place an LED for each of the instruction signals coming

out of the Instruction decoder, you should see the LED for IncB emitting light as soon as you write the instruction into RAM. Similarly, test the RAM and the Instruction Decoder with the other instructions.

6.6. PROJECT 5: Control Signal Generator

The function of the Control Signal Generator (CSG) is to generate the micro-operation control signals that will implement instruction fetch and instruction execution. Referring back to Figure 7 we see that the CSG encodes the incoming timing signals and instruction signals to accomplish this. This section shows how these incoming signals are encoded into the control signals.

Figure 21 shows a recommended way of organizing the CSG chips: two TTL 74LS08's (AND gates) on breadboard 4, and three TTL 74LS32's (OR gates) one of which is on breadboard 2 and the other two are on breadboard 1.

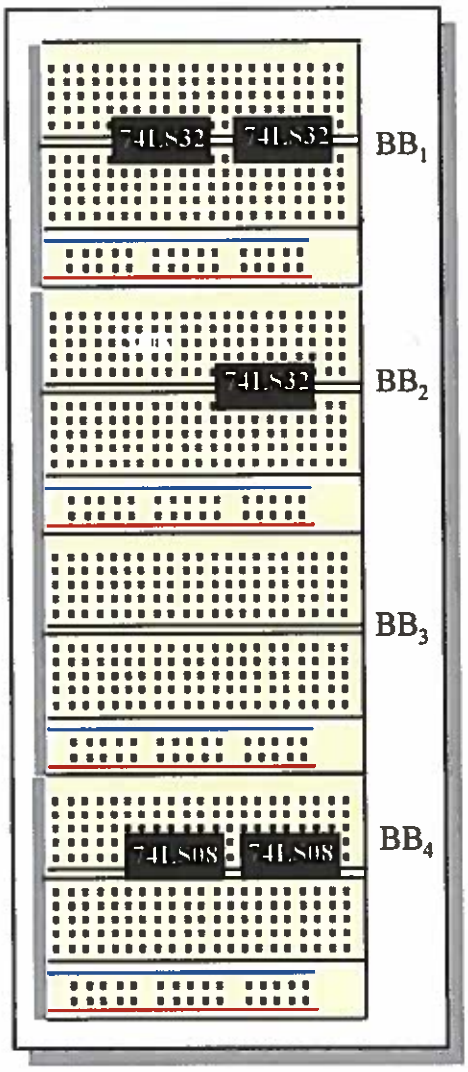


Figure 21. The Control Signal Generator Chips.

Figure 22 shows the logical AND gate encoding of the three instructions symbolically. Figure 23 shows a physical implementation of Figure 22. The following paragraphs explain the control signal encoding for these instructions .

0000

IncB - Referring back to Figure 7, we see that this instruction requires two time steps:

1. Data in Register B flows through the BUS into the Adder and the incremented value is stored in the Sum Register. The control signals for this step are B_{out} and Sum_{in}. These signals are activated at $\mathbf{T_2}$ when the Instruction Decoder output is IncB.
2. The contents of the Sum Register flows through the BUS back into Register B. The control signals are Sum_{out} and B_{in}. These are activated at $\mathbf{T_3}$ when the Instruction Decoder output is IncB. Please see Figure 23 for how these control signals are generated.

0001

MovAB - This instruction is executed in one simple step: data in Register A flows through the BUS into Register B. The control signals are A_{out} and B_{in} and these are activated at T_2 (or a higher time step). Please see Figure 23.

0010

MovBA - Encoding for this instruction is similar to that for MovAB discussed above.

IncA – This instruction is similar to IncB and will not be discussed in the following diagrams. Its implementation is left open as an exercise. However, if your project includes interrupt processing and its associated instructions (e.g., read, return, jump), the resources (e.g., gates and breadboard space) used by this instruction will be needed and you

will have to undo the wiring for this instruction. If you choose to implement this instruction temporarily, you can use 0011 as the operation code.

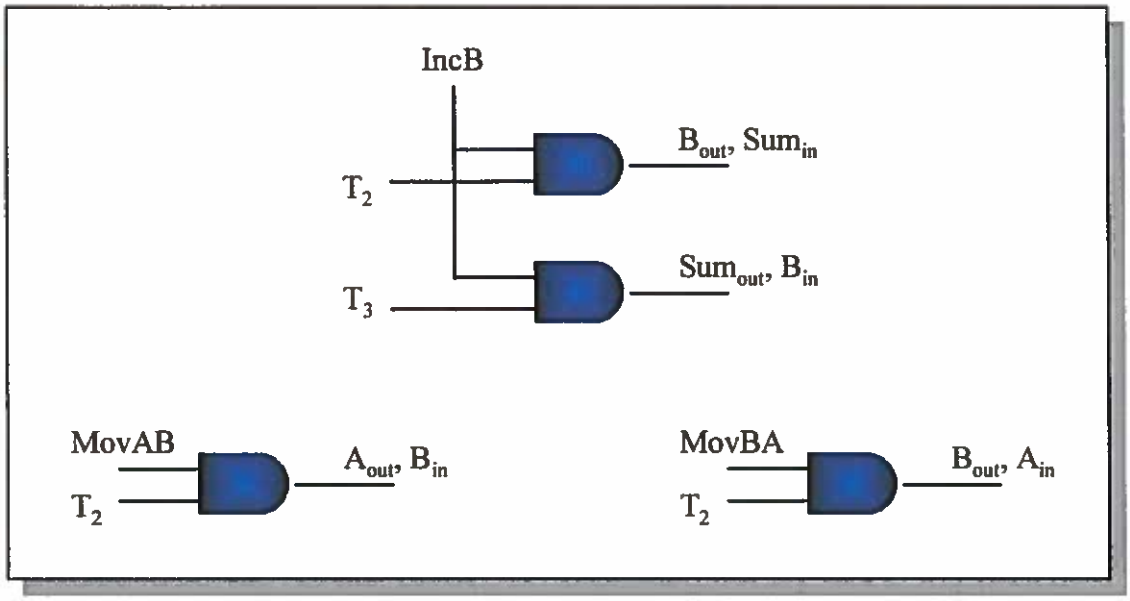


Figure 22. Control Signal Encoding.

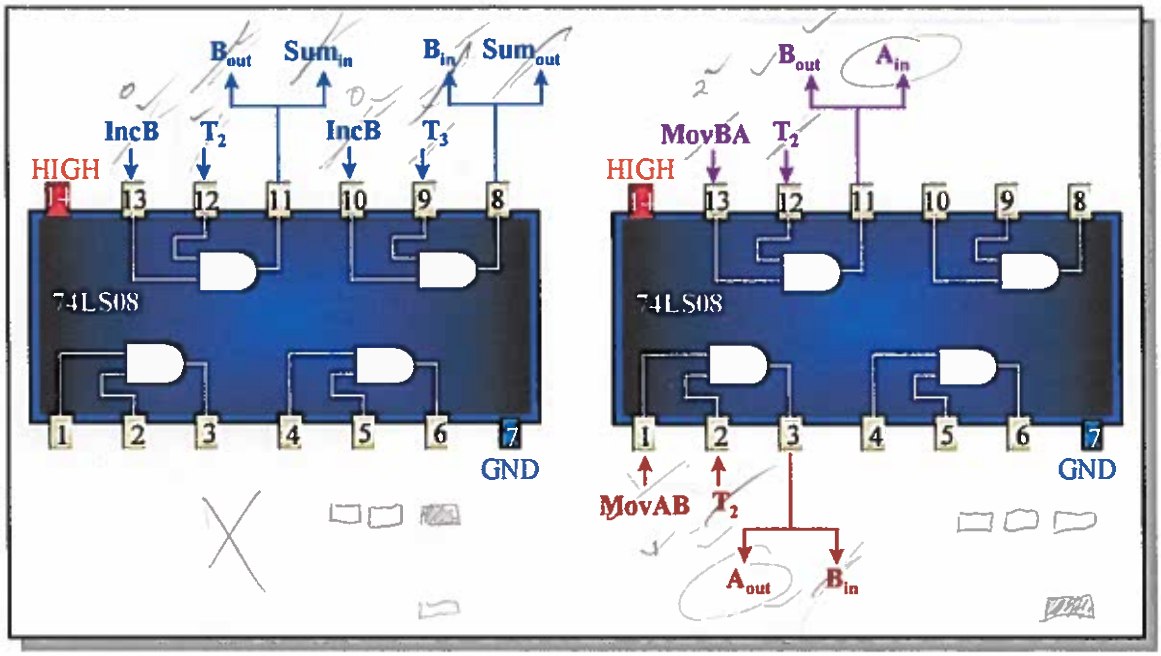


Figure 23. AND gate encoding of the control signals.

An important point to remember is: <u>although a source signal pin can be connected to multiple destinations (i.e., fan-out), the reverse is not true; the input pin of a chip cannot be directly connected to more than one signal source.</u> Please see in Figure 23 that the control signal B_{out} occurs

twice. You need to feed this signal pair into an OR gate and connect the OR gate output to the B_{out} destination pin (see Figure 24). This way, the destination pin is activated by one or the other active signal. You do the same for the B_{in} signal pair of Figure 23. Likewise, the Sum_{in} destination pin now has two incoming signals: the one in Figure 23 and the T_0 signal used during incrementation of the Program Counter. The same applies to the SUM_{out} signal. As Figure 24 shows, these signal pairs are OR'ed before connecting to the destination pins. (Note: if a destination pin had more than two source signals, then they had to be converged through a cascade of OR gates).

A control signal that occurs only once directly connects to its destination pin. So far, the only source for the PC_{out} pin of the Program Counter is T_0, and the only source for the PC_{in} pin is T_1. Likewise, MAR_{in} uses only T_0. Also, if you choose not to implement IncA, each of A_{out} and A_{in} has a single occurrence due to instructions MovAB, and MovBA, respectively. Therefore, these timing signals do not need to be OR'ed and are directly connected to the destination pins. (The MAR_{out} pin is not used in this project but provided for future use).

If you have not connected the MAR output pins to RAM address pins, connect them now. When you complete these connections you can program and run your computer.

It is recommended that you test each instruction after you build it (i.e., do not implement all instructions at once). For testing, you simply have to program your RAM and run your computer (see section 8 - Programming and Running your computer).

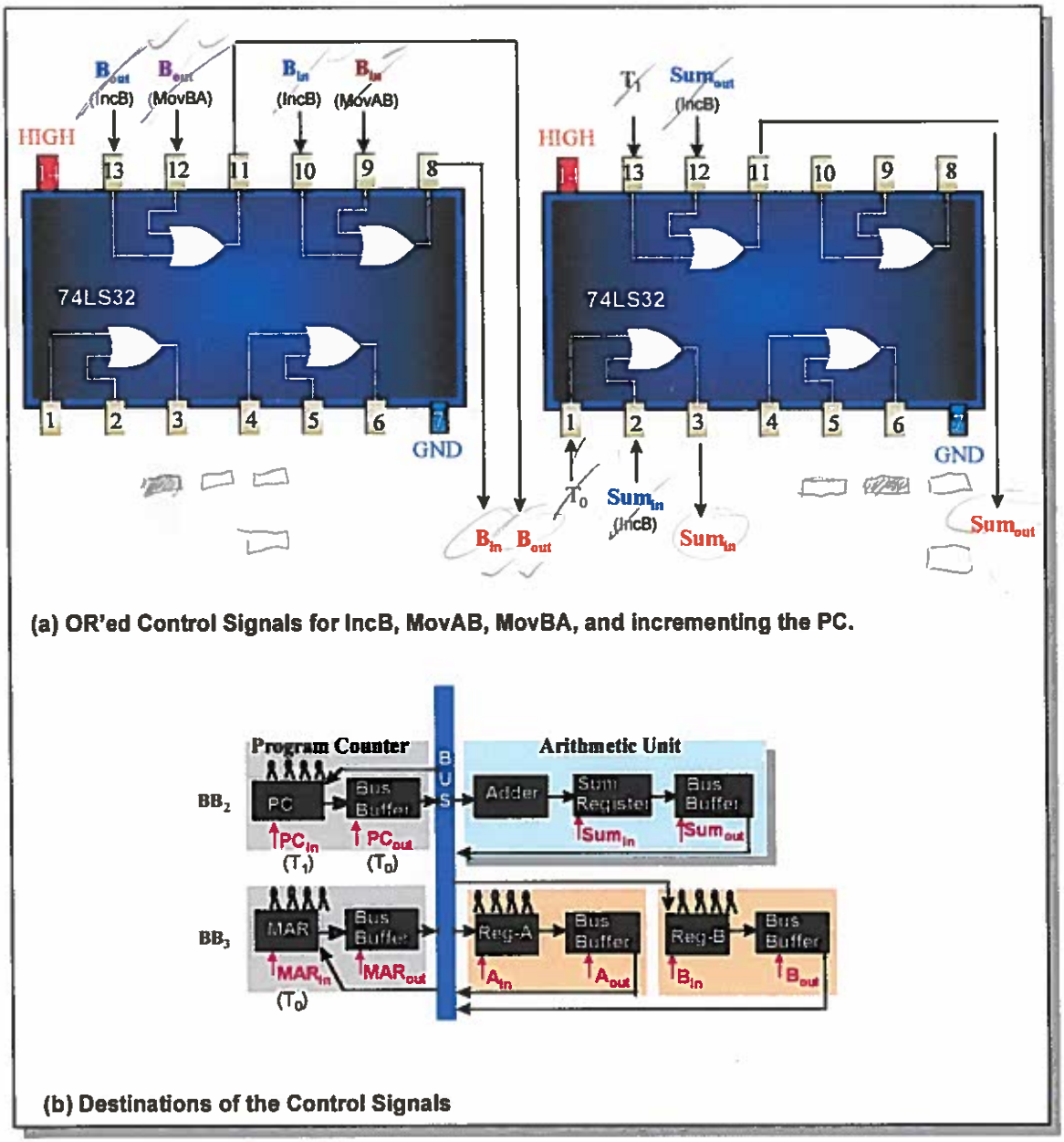


Figure 24. Control Signals.

7. PROJECT 6: Interrupt Processing

In a typical computer, upon encountering an interrupting event (e.g., the user enters input data for the computer to read), the computer saves at a known location the PC value for the running program and executes the interrupt routine for that event. The computer knows the RAM addresses of these routines for they may be hardwired or initialized during system initialization in an array of registers called the interrupt vector. Likewise, an interrupt routine for an event is also initialized in RAM and begins at RAM location pointed by the associated interrupt pointer. A software Return instruction put at the end of an interrupt routine returns execution control to the interrupted program by loading the saved PC value back into the PC.

In this section we shall implement a simple interrupt mechanism for reading input data. Our interrupt vector will contain only one element: the Interrupt Pointer buffer in Figure 25 (i.e., a 74LS126 chip). The RAM address of the interrupt routine will be 1101. We will hardwire the Interrupt Pointer of Figure 25 to contain this value (via connecting its input pins to HIGH and GND terminals).

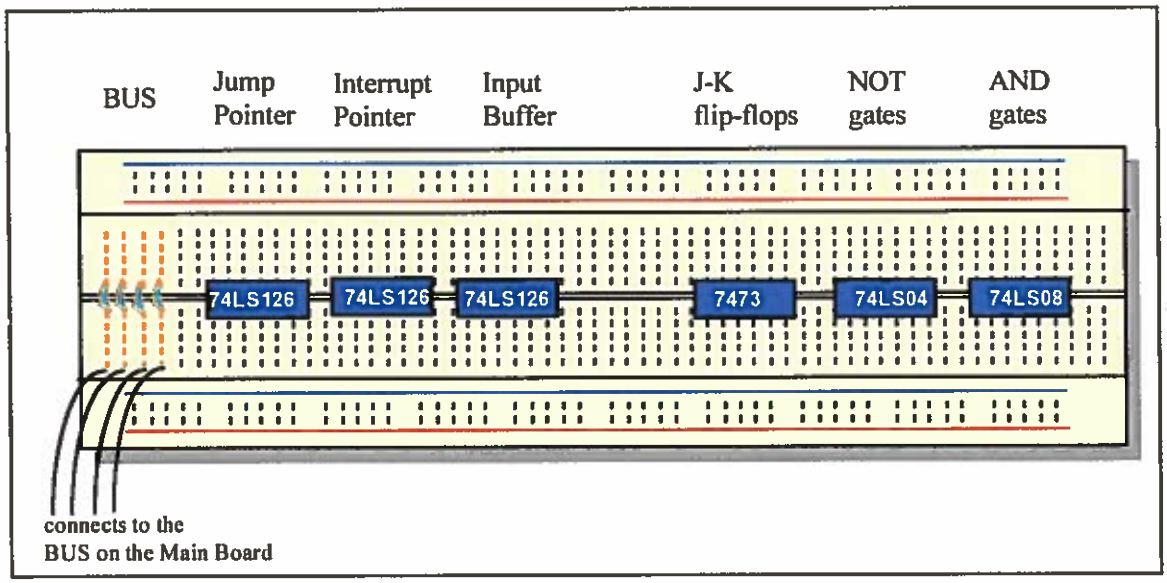


Figure 25. The Extension Breadboard (BB5 shown in Figure 7).

We shall interrupt a program's execution by a signal S, by manually setting a J-K flip-flop output to HIGH (see section 9.11 for a description of J-K flip flops). At time T4 of each instruction cycle, the computer will check the signal S. If it is set, the computer will save the PC value into Register A, and, at time T5 it will load the PC with the interrupt routine address. As a result, the next instruction cycle will execute the first instruction of the interrupt routine.

The input interrupt routine that you will initialize will contain only two instructions: Read (at address 1101) and Return (at address 1110). The Read instruction will move the contents of the Input Buffer of Figure 25 (a 74LS126 chip) into Register B, and the Return instruction will move the contents of Register A back into the PC (i.e., the interrupted program will resume execution).

Before signaling the computer via signal S to read, you will supply the input data in the Input Buffer via connecting its input pins to HIGH and GND terminals (e.g., via manual wire connections or via a DIP switch settings).

We shall implement the signal S via a J-K flip-flop. As the figure below shows, the TTL 7473 contains two independent J-K flip-flops: FF_1 and FF_2. We shall use FF_1 for the signal S (the other one will be used within the interrupt mechanism).

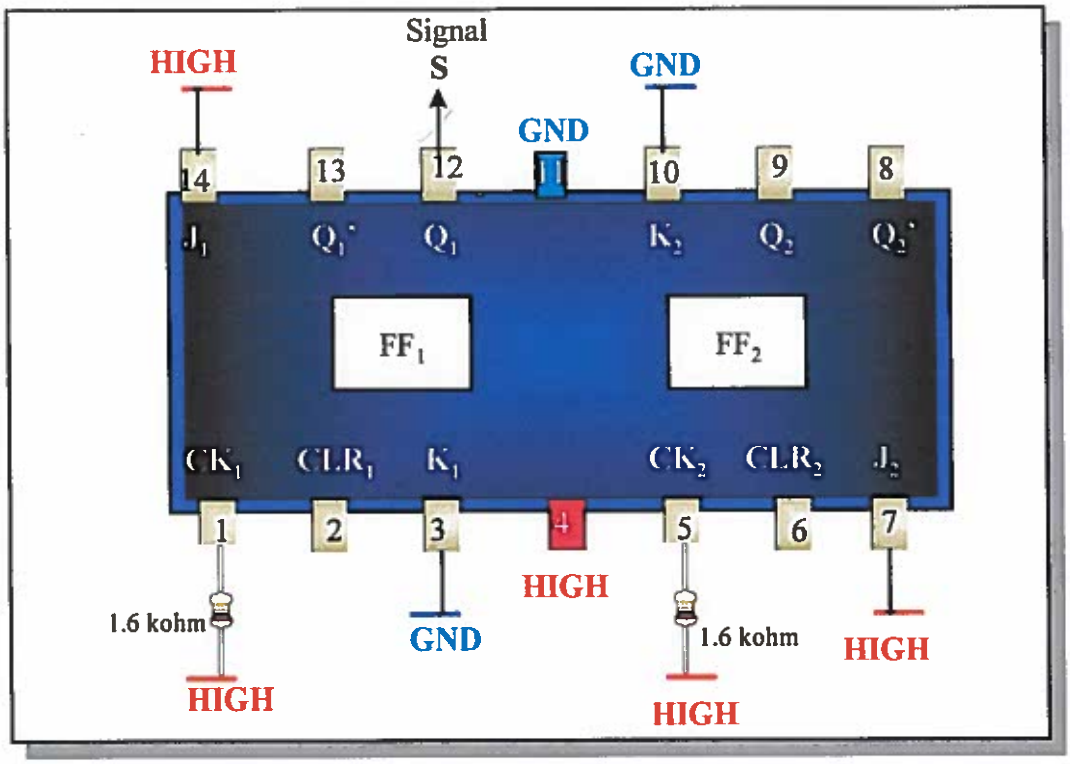


Figure 26. The TTL 7473 chip configuration.

In either flip-flop, when J=HIGH and K=LOW, momentarily bringing CK to LOW sets the Q output to HIGH. Likewise, momentarily bringing CLR to LOW sets the Q output to LOW (i.e., clears the signal).

With both flip-flops (FF_1 and FF_2), please connect the J pins to HIGH terminal and K pins to LOW terminal. In this configuration, when you momentarily ground the CK_1 pin via a piece of wire, the Q_1 output will go HIGH (i.e., the signal S). Hence, you will *manually* signal the computer to interrupt the currently executing program. You can monitor the signal S via an LED. (Likewise, as used in the interrupt mechanism below, momentarily grounding CK_2 will set Q_2 to HIGH).

If you use an LS version of 7473, a problem you might sporadically have is that LOW noise signals (e.g., caused by bouncing pulses while connecting the power supply) might set Q_1 and/or Q_2 at a time when you do not want it. You can prevent this by connecting the CK pins to the HIGH (+5V) terminal via the 1.6 kohm or 2 kohm resistors in your kit. Note that the *regular* 7473 in

your kit is not as sensitive to noise signals and would not require these resistor connections.

Now, you have enough information to construct the interrupt circuit. Figure 27 illustrates how the computer jumps to the interrupt routine upon encountering the signal S.

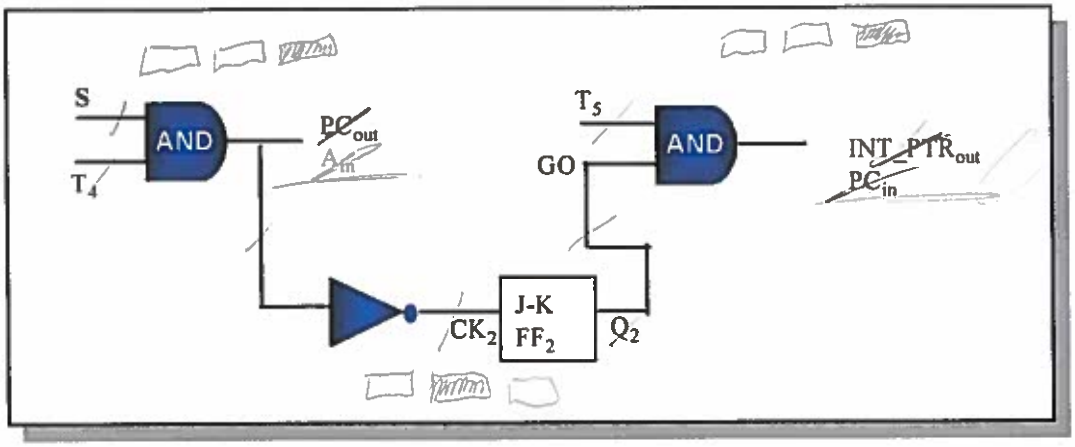


Figure 27. A simple interrupt processing logic.

As the figure shows, at time T_4, the contents of the PC is saved in register A, and the Q output of the flip-flop FF_2 (also labeled as GO) is set to HIGH. At time T_5, the address of the software interrupt routine is moved into the PC. The continuity between these two steps is provided by setting GO to HIGH.

The first software instruction in the interrupt routine is the "**Read**" instruction. You will implement this instruction as in Figure 28.

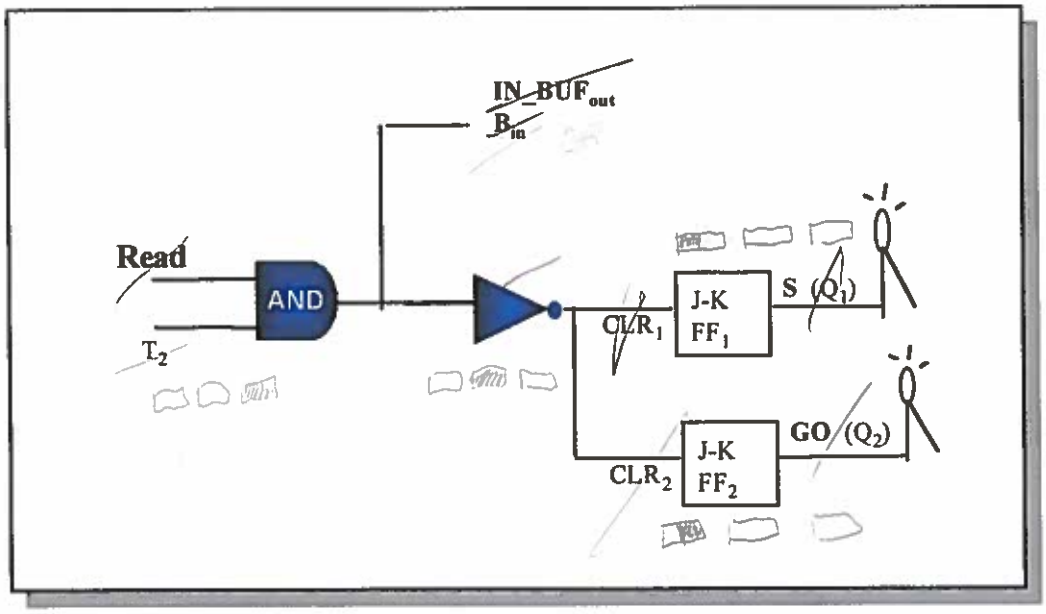


Figure 28. Logic diagram for the READ instruction.

The "Read" instruction will move the Input Buffer contents (which you can configure to any value between 0000 and 1111) into Register B and will clear both flip-flops (i.e., signals **S** and **GO**) to prepare for the next interrupt. Please note that due to the instructions MovAB and IncB, the B_{in} pin of Register B is already wired. Therefore, this new B_{in} has to be OR'ed with the existing control signal.

"**Return**" is a new instruction you will have to implement which, like a MOV instruction, will simply move the return address saved in Register A back into the PC (see Figure 29). This will restart the interrupted program where it was interrupted.

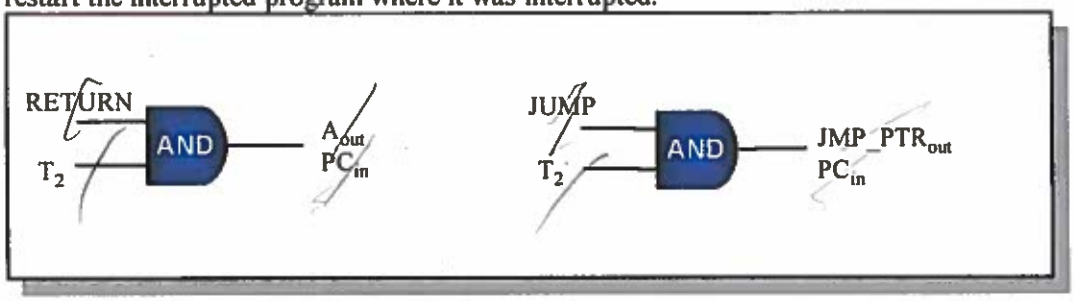


Figure 29. Instructions RETURN and JUMP.

An optional but recommended software instruction to implement is a JUMP instruction that would jump to memory address 0000. The TTL 74LS126 labeled above as the "Jump Pointer" is provided for this purpose. Simply connect the input pins of this buffer to the GND terminal. Then, a JUMP instruction would merely be moving the contents of the jump pointer (0000) into the PC as in a MOV instruction. When used before the interrupt routine, this instruction would facilitate isolating the interrupt routine from the sequential control flow and also will re-demonstrate the execution of your software program.

Again, the new A_{out} signal has to be OR'ed with the existing A_{out}, and the new PC_{in}(s) must be OR'ed with the existing PC_{in}.

(In Figure 25, the output pins of all three TTL 74LS126's connect to the BUS. Do not forget to ground the output-control pins of these chips until they become an integral part of your circuit).

8. Programming and Running your computer

Programming the Computer

1) Connect the power supply to breadboard-4 which contains the RAM. (It is assumed that all power strips are interconnected via HIGH-to-HIGH and GND-to-GND wire connections).
2) Disable all boards except the RAM board. You can do this by disconnecting the power connection between breadboard-4 and the rest of the boards: simply pull out one end of the HIGH-to-HIGH inter-board connection wire.
3) Program the RAM using the DIP switches. To write an instruction into RAM (e.g., 0000 for IncB) set the data switch to 0000, set the address switch to RAM location address (e.g., 0000), and momentarily ground the WE pin. If you place an LED for each of the instruction signals coming out of the Instruction decoder, you should see the LED for the addressed instruction emitting light as soon as you write the instruction into RAM.
4) After you are finished with programming, make sure that all four of the address switch elements are OFF so that they will not interfere with the MAR addresses during execution.

Running the Program

To run your program, restore the power connection you removed in step 2 above. As soon as you do this, the clock will start ticking and the instruction cycles will begin. Sit back and enjoy your successful work.

Contact Bouncing

Ideally, when you insert a power wire (e.g., +5V terminal of the power supply) into a power-strip slot, you expect a clean contact where the voltage goes from 0 to +5V. In reality, there will be an initial sequence of mechanical contact bouncing. You insert the wire (or press the button of a switch) and the contact will open and close many times before finally staying in position. The train of transient spikes so caused might affect the initial configuration of your circuit. For example the counter of the TSG might start at a value higher than zero. Another example, the Q output of a J-K flip-flop (e.g., Signal S) might be set. Yet another example (although not likely with 7475) is that the initial power-up value of the PC may be 0000 or 1111.

Normally, this situation is handled via a contact de-bouncing circuit. For our purposes, the effects of contact bouncing is practically insignificant and, for simplicity, we do not implement a contact de-bouncer. If it happens, you can simply restart your computer. Regardless, you must understand the effect of contact bouncing on the PC. Assuming that your program starts at RAM location 0000, if the PC starts as 0000 your program will run as expected. If the PC starts as 1111 however, MAR will copy this value at T_0 and will point at RAM location 1111 during the first instruction cycle. It will point at RAM location 0000 during the second instruction cycle. If you enter at RAM location 1111 a no-operation code (i.e., any 4-bit value other than a valid instruction code), the net effect is that your program will execute as expected but one instruction cycle later. If the RAM location 1111 contains a Return instruction though, the path your program will take depends on what Register A contains (i.e., the return address).

9. APPENDIX - CHIP DIAGRAMS

The chips that you use in this project belong to the TTL (transistor-Transistor Logic) Family. Following sections describe these chips.

9.1. TTL 74LS04 - Inverter

Figure 30 shows the internal layout of TTL 74LS04. This chip contains six inverter (**NOT**) gates which operate independently of each other but share the HIGH and GND terminals. Each inverter accepts a bit (i.e., 0 or 1) at its input pin, inverts this data (i.e., 0 to 1 or 1 to 0), and puts this inverted data on its output pin.

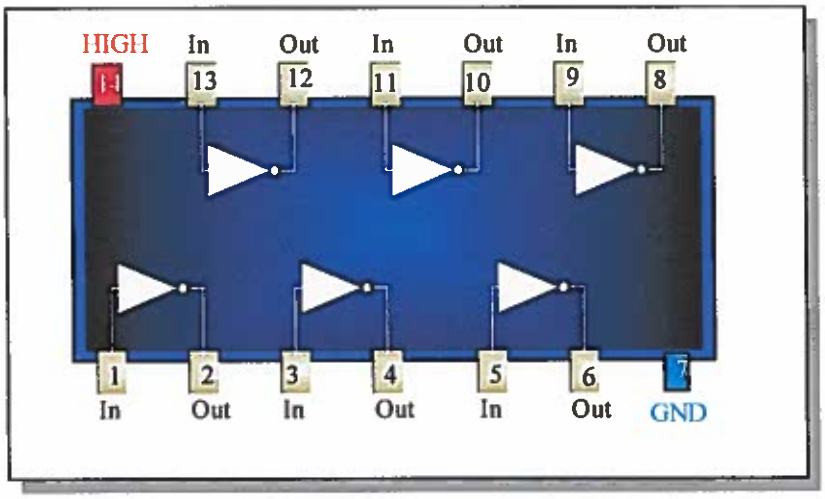


Figure 30. TTL 74LS04.

9.2. TTL 74LS05 - Open Collector Inverter

Similar to TTL 74LS04, this chip contains six independent inverters. The term open-collector indicates that the output transistor's collector is not connected to a positive voltage (i.e., left open). To function, a gate requires an external pull-up resistor connecting the output to the HIGH terminal to complete the output transistor circuit inside the chip.

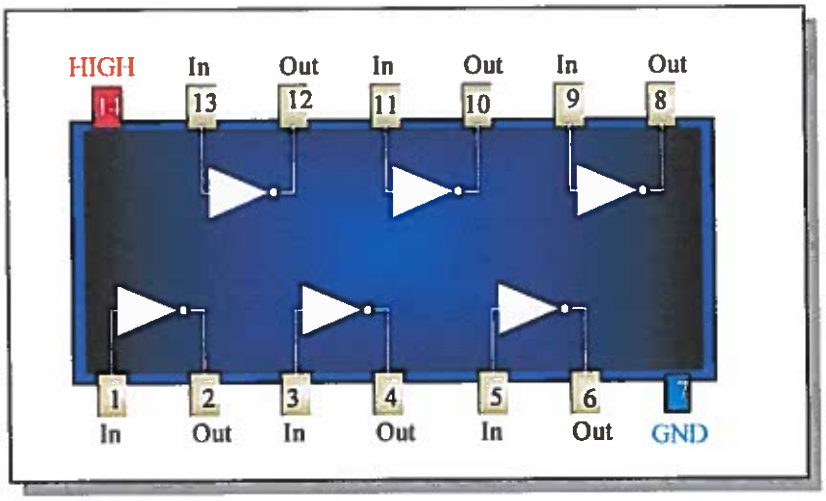


Figure 31. TTL 74LS05.

9.3. TTL 74LS08 - AND

This chip contains four independent AND gates. See section 4 for a description of an AND gate's behavior.

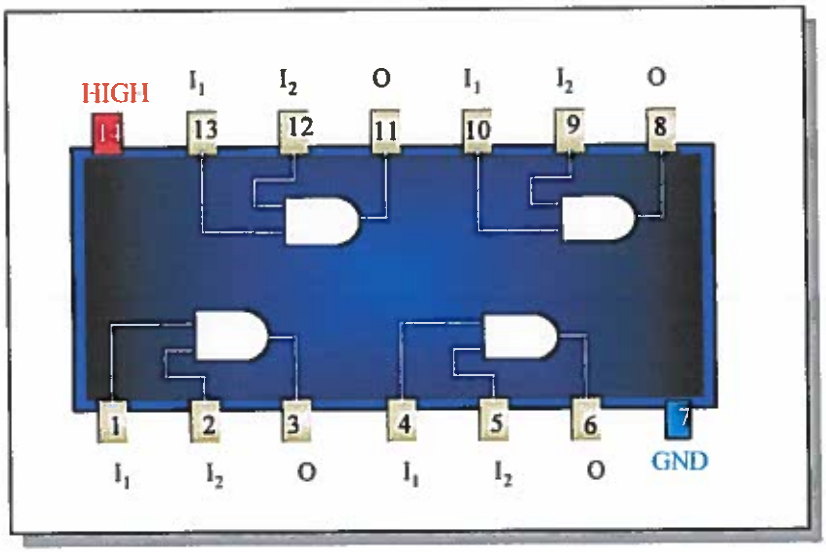


Figure 32. TTL 74LS08.

9.4. TTL 74LS32 - OR

This chip contains four independent OR gates. OR gates are described in section 4.2.

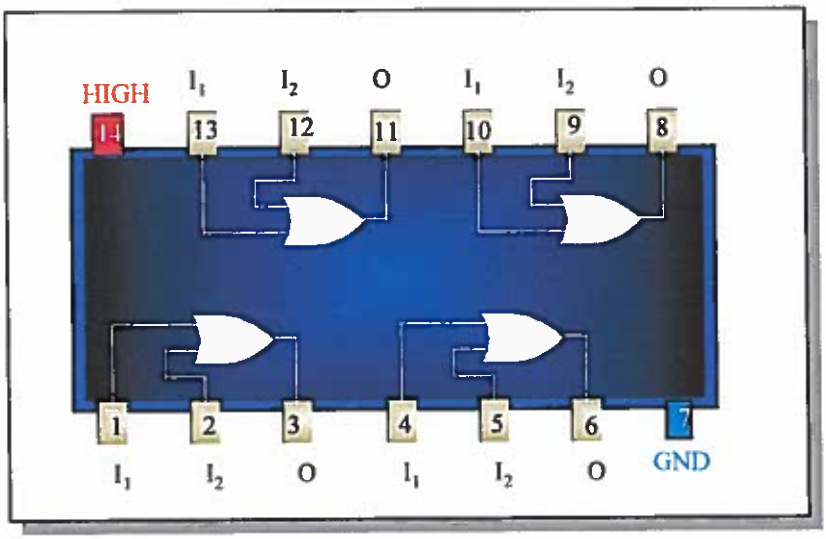


Figure 33. TTL 74LS32.

9.5. TTL 74LS42 - Decoder

Figure 34 shows that TTL 74LS42 has four input pins and ten output pins. The decoder activates (i.e., selects) one output pin corresponding to its 4-bit input data. TTL 74LS42 decodes input values from 0000 (corresponding to Out_0) to 1001 (corresponding to Out_9) by putting an active LOW signal on the selected output pin and HIGH signals on the rest of the output pins. For example, if the input data is 0000 (corresponding to Out_0), it selects pin 1 by putting an active-

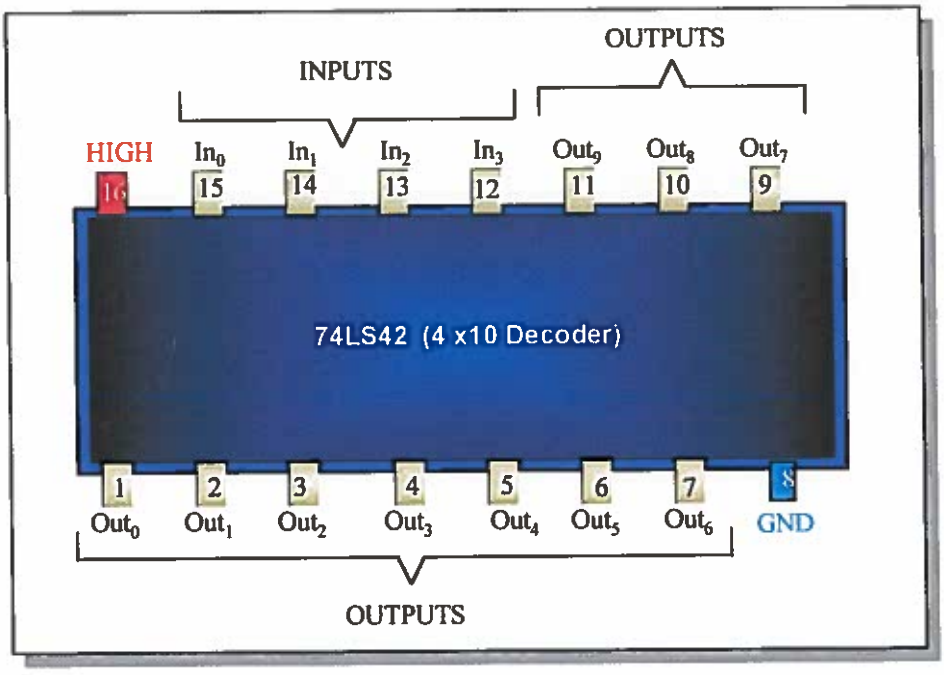


Figure 34. TTL 74LS42.

LOW signal on pin 1 and HIGH signals on pins 2 through 9. If the input value is 0001 (corresponding to Out_1) it puts an active-LOW signal on pin 2, and high signals on the rest of the output pins. And so on … If you need an active HIGH signal (which you will), you feed the active-LOW output signal of the decoder into an inverter (i.e., a NOT gate).

9.6. TTL 74LS75, 7475 - Data Flip-Flops

Figure 35 shows that this chip contains four storage units (also called data flip-flops) each of which stores one bit of information (i.e., 0 or 1). A data flip-flop (described in section 4.5) accepts its input data only when its **Enable** control pin is HIGH (i.e., +5 volts DC). A data flip-flop continuously outputs what it stores. When the Enable control is LOW (i.e., GND), the contents of a data-flip flop and its output is isolated from its input (i.e., changes in input data does not change what the flip-flop stores and outputs). Initial power-up value of a regular 7475 is 0000 and that for 74LS75 is 1111.

Pin 13 enables two of the four flip-flops, and pin 4 enables the other two. You will use TTL 74LS75 as a four-bit data register and interconnecting pins 13 and 4 will allow you to enable all four data flip-flops with a single HIGH signal connected to pin 13 or to pin 4.

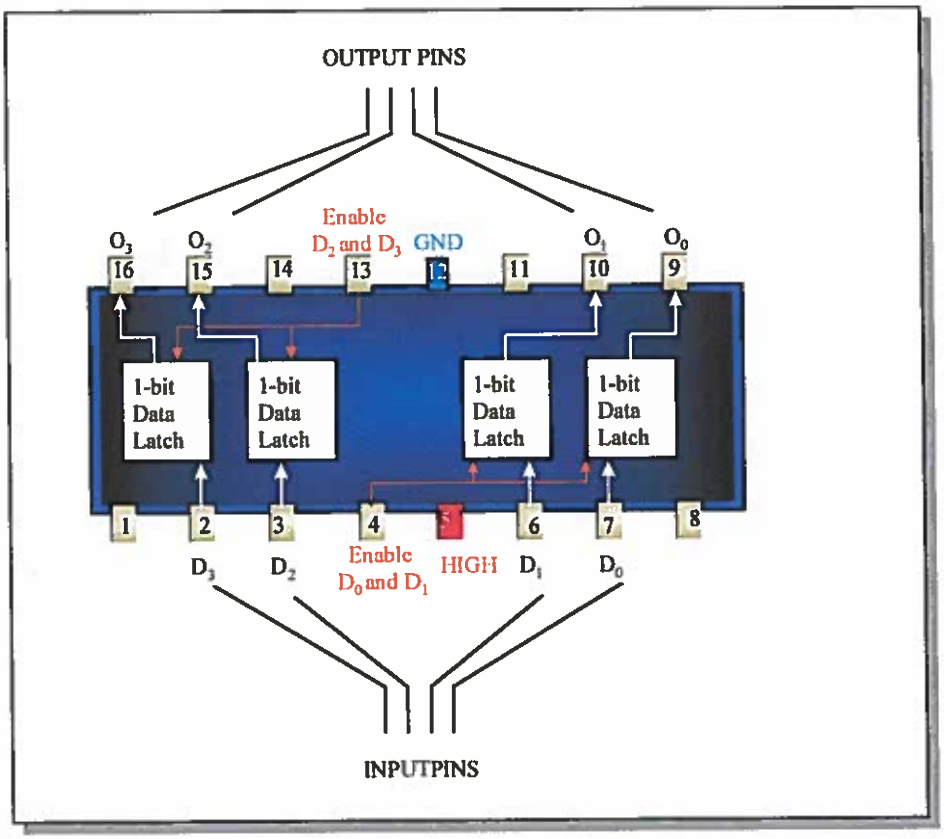


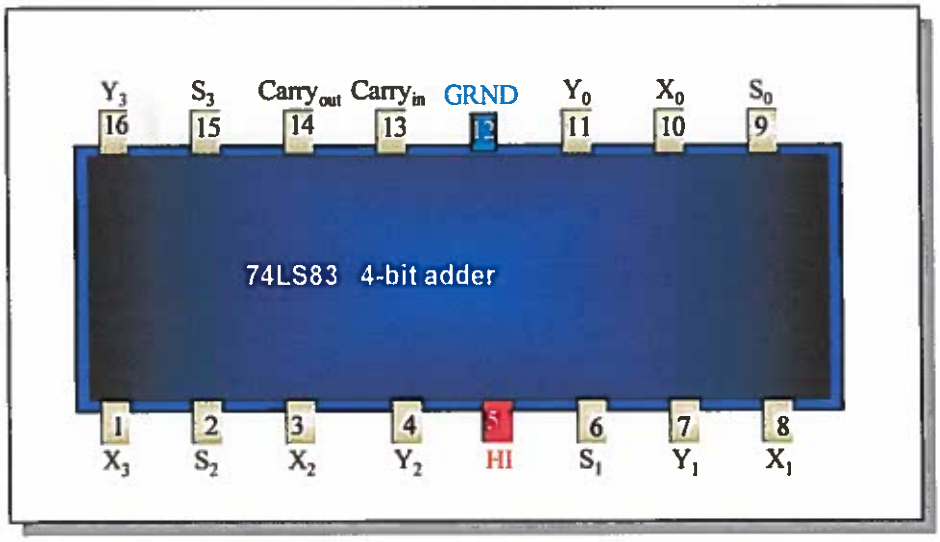
Figure 35 . TTL 7475 or 74LS75.

9.7. TTL 74LS83 - Adder

TTL 74LS83 adds two 4-bit inputs (denoted as **X** and **Y** in Figure 36) and produces a 4-bit Sum (denoted as **S** in Figure 36).

In general, $Carry_{out}$ and $Carry_{in}$ pins are used in cascading these chips two make a larger adder. If the sum of a chip is larger than what four bits can represent (i.e., larger than 15) then the $Carry_{out}$ pin is activated. Likewise the $Carry_{in}$ pin receives the $Carry_{out}$ from the preceding chip in the cascade.

In your project you will use only one TTL 74LS83 for incrementing by 1 only. You will connect the **Y** pins to GND (i.e., **Y** pins will assume the value of 0000) and connect the **X** pins to a data source. The $Carry_{in}$ pin, when open, will assume the default value of HIGH and will be used in incrementing the data coming in via the **X** pins. The incremented value will appear on the **S** pins.



INPUTS: X_1 X_2 X_3 X_4, Y_1 Y_2 Y_3 Y_4 SUM BITS: S_1 S_2 S_3 S_4

Figure 36. TTL 74LS83.

9.8. TTL 7489 - Random Access Memory

TTL 7489 contains sixteen 4-bit memory locations. A location is selected by its address on the address pins (AD_0, AD_1, AD_2, AD_3) for writing into it or reading from it. The data to be written into a location is specified via the input pins (I_0, I_1, I_2, I_3) and the data in a selected location (via the address pins) is continuously present on the output pins (O_0, O_1, O_2, O_3).

Both ME (Memory Enable) and WE (Write Enable) are active LOW pins. The ME must always be connected to GND, and the WE (Write Enable) pin is momentarily brought to LOW to write input data into a location. When not writing into memory, the WE pin is left open (it assumes the default value of HIGH).

The output pins deliver the complement of the data at an addressed location. This 4-bit output data can be inverted to its original input value by using a TTL 74LS04-Inverter.

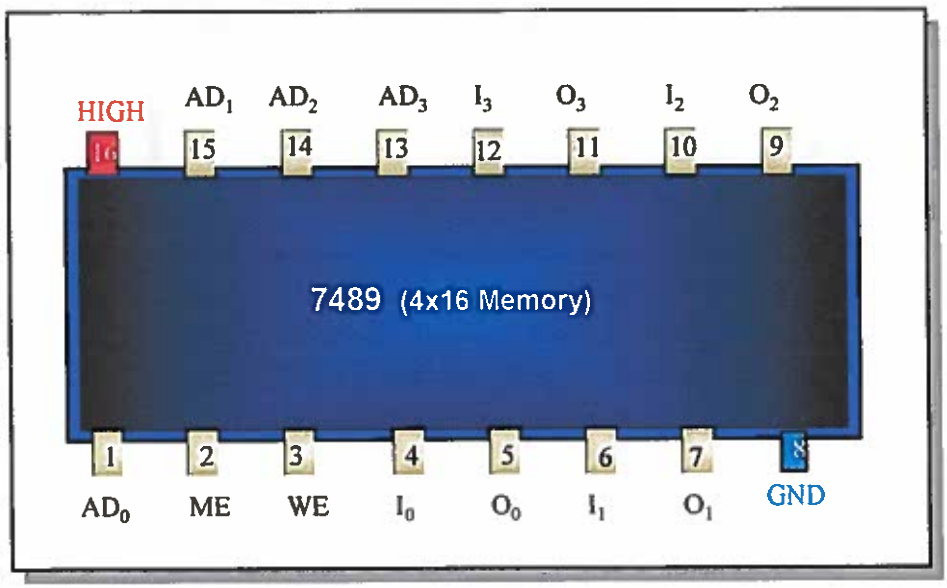


Figure 37. TTL 7489.

(e.g., TTL 74LS75) to a BUS. Only one device must output to a BUS at a time. When the control inputs are LOW, the outputs of the buffers are disconnected from the BUS. Note that, by default, the "Cntrl" pin of a buffer assumes a HIGH value (i.e., if this pin is left open, the buffer will output its input data). Therefore, when a buffer is connected to a shared BUS, it is a good practice to ground its control pin (until it is connected to a source and becomes an integral part of the circuit). This way, the buffer output will be disconnected from the BUS, and it will not interfere with another device outputting to the BUS.

9.11. TTL 7473 : Dual J-K Flip-Flop

TTL 7473 contains two independent J-K flip-flops.

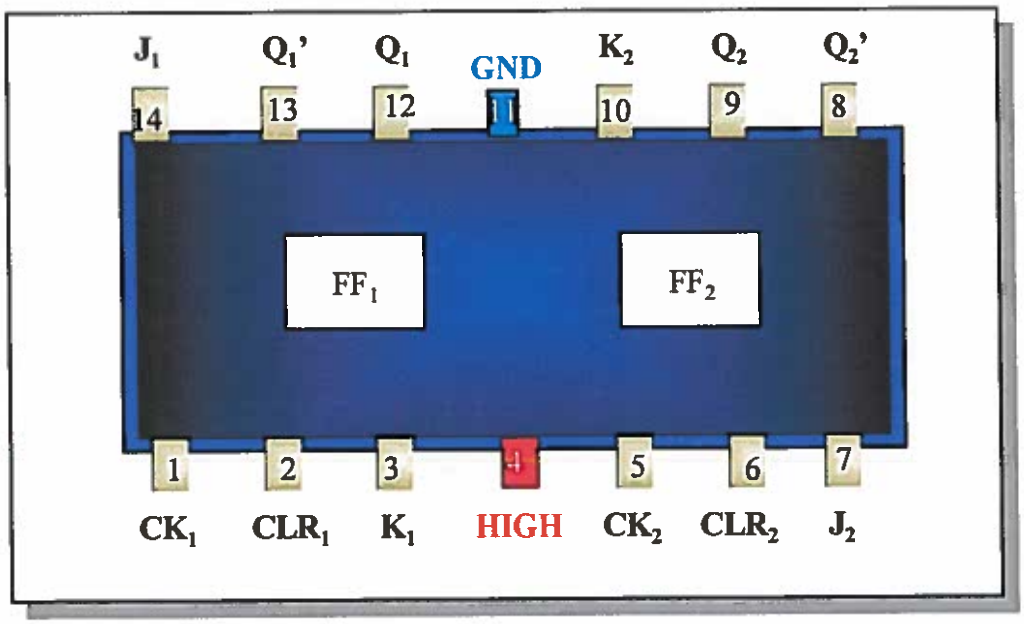


Figure 40. TTL 7473.

In either flip-flop, when J=HIGH and K=LOW, momentarily bringing CK to LOW sets the Q output to HIGH (e.g., signal **S** or **GO**). Likewise, momentarily bringing CLR to LOW sets the Q output to LOW (i.e., clears the signal).

In this project, Q_1 pin of flip-flop FF_1 will be used for the interrupt signal **S**, and the Q_2 pin of flip-flop FF_2 will be used for the intermediate signal **GO** within the interrupt circuit.