CS 428 Lab 1: URUART

Design Review: Friday April 1 (Sign up for a time)
Working Hardware Due: Friday April 8
Lab Books Due: 4 P.M. Friday April 8

READING

Read the EIA RS-232-C handout, but don't panic as you will not have to implement the whole thing. This document will give you some good scope as to where your actual project fits into the scheme of things. It is also good practice to read this kind of spec, as you will be doing a lot of this if you are ever possessed enough to become a professional designer.

PURPOSE

Build a circuit similar to a UART, which is a very real part of any computer system. This lab is also intended to help bridge the gap between the previous labs in 427 which were intended to help you learn certain design techniques and circuit phenomena, and the "real world" (whatever that is) of digital circuits in computer applications. The project fits into the overall framework of designing an entire computer during the scope of this course. Your computer will need to communicate with a terminal. Most terminals produce signals which are governed by a particular protocol, the most common is an RS-232-C ASCII protocol. A device which you can buy that sends and receives this protocol is called a UART (Universal Asynchronous Receiver Transmitter). There are several UART's around, but just buying the part doesn't teach you much, and therefore you will design one. Your UART will allow you to communicate with a standard terminal at 4800 baud (4800 bits/second), and is functionally simplified from the commercially available type.

It will hopefully also be fun.

THE TERMINAL

The terminal sends RS-232-C ASCII information in a bit serial fashion at certain standard voltage levels (known as EIA levels). Usually these EIA levels have to be conditioned to be compatible with the logic levels of the rest of URSYSTEM. Fortunately for you, the terminals we will be using will provide TTL signal levels to URUART which will be built from the lab's CMOS inventory. The pinout for the terminal, which will plug in to a standard 16 pin socket is:

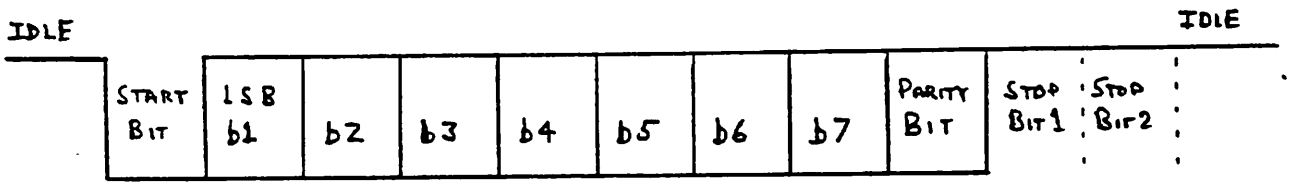
RCV (data out)	pin 16	
XMT (data in)	pin 12	
Signal Ground	pin 7	
Chassis Ground	pin 1	(use both)

-- warning -- make no connections to other pins. Voltages are present at other pins which will damage the equipment.

Most terminals provide switches to specify:

1. Half or Full Duplex
2. Odd or Even Parity
3. Baud Rate
4. 1 or 2 Stop Bits
5. Some other stuff which you can ignore for now

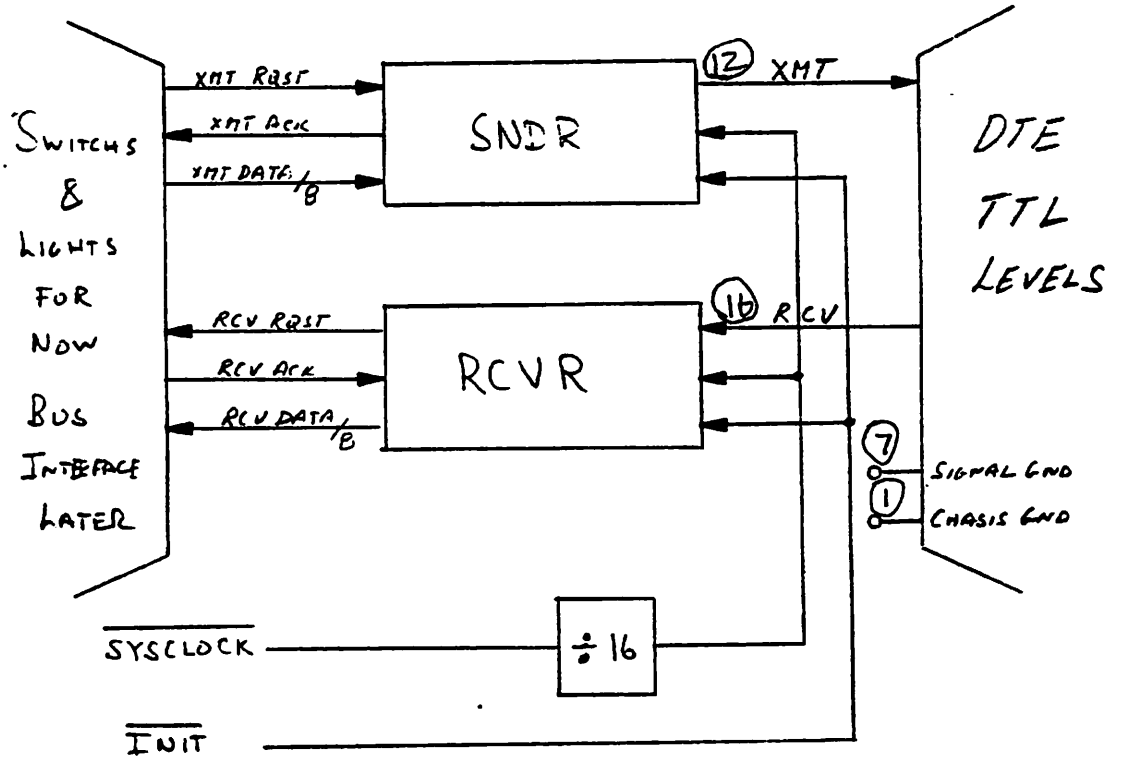
We will assume for now that your UART will work at 4800 baud. We will assume that parity is handled elsewhere and you are passing 7 bit ASCII with parity to yield an 8 bit wide character. The signal is passed in bit serial fashion in the following format:



When the line is idle it remains in a high state. When it drops low it appears as a 0 bit, followed by 8 data bits, followed by 1 or 2 stop bits, (which are high bits). The bits after the start bit are sent and received at regular intervals according to the baud rate. Some UART type circuits are smart enough to detect the baud rate but here we will assume 4800 bits per second. The most general protocol is to transmit 2 stop bits to the terminal, and assume that the terminal is sending only 1 stop bit to you.

PROJECT DESCRIPTION

You are to build a UART to communicate with a terminal as described above. You circuit will basically look like:



The SNDR (Sender) and RCVR (Receiver) parts should each be controlled by separate state machines and use separate data registers. The SNDR upon seeing a XMT.REQ will take the 8 bit ASCII code from the XMT.DAT lines and transmit them to the terminal over the XMT line. When the SNDR has taken the data from XMT.DAT it will raise the XMT.ACK line. Sometime after the XMT.ACK is gone high the XMT.REQ will be pulled low. When the SNDR has transmitted the data, and when the XMT.REQ is low, the SNDR will pull the XMT.ACK low, thus completing the four-cycle handshaking convention.

When the RCVR sees the RCV line drop low, it will receive the 8 bit message and place the information on the RCV.DAT lines. It will then raise the RCV.REQ, when it sees the RCV.ACK line go high, it will drop the RCV.REQ, wait for RCV.ACK to go low and then look for another character from the terminal.

You should use a single system clock to drive your UART which is 8 times faster than the 4800 baud rate, and you should sample the received signal from the terminal as nearly in the center of the bit as possible.

The last page of this lab is a copy of the ASCII codes which will be transmitted. Note however that your UART doesn't care what the code is, but most likely your computer will care later.

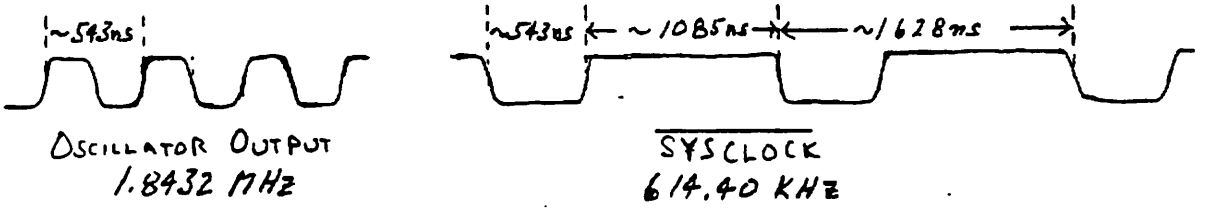
Design your UART such that the receive and transmit sections can be working simultaneously on the two serial RCV and XMT lines.

HINTS

Since you will be keeping your circuit and building other stuff onto it later - DESIGN IT PROPERLY OR YOU WILL GET SCREWED BY IT LATER!!

The packaging of this quarter's project will be a major consideration, so you should spend a bit of time on each design to make it as small as possible. In this lab DO NOT USE EPROM's for your state machines as you will need them for the CPU. Also design URUART so that it fits on a single board. This means that for the entire quarter you should put chips as close to each other as is reasonably possible. Ten rows of four 16 pin chips each is about the best that you can do.

For your clock use the crystal oscillator circuit. The crystal will produce a 1.8432 MHz clock which will be too fast for URSYSTEM. You should design a clock generator circuit which will divide the clock down to yield a 614.4 KHz clock. The clock should be asymmetric as shown below. Your clock circuit should be stoppable and should include a single step capability. Also include a push button controlled master clear (INIT') generator as part of the clock unit. Divide the system clock by 16 to get the 8x4800 baud clock for URUART. Locate the clock on a separate board from URUART - this will become the maintenance panel board - your next project. SYSCLOCK' will be connected between boards on pin V on the edge connector. INIT' will appear on pin 15.



DOCUMENTATION STYLE

This quarter the emphasis will be placed on very systematic design and documentation procedures. For everything that you build you will be required to have the following documentation sections:

1. Introduction describing the method you have used in your design and a description of the way in which you have decided to solve the problem and why. This section should contain a block diagram of the circuit and a description of the function of each block. The block diagram should show all of the communication signals between blocks, and additionally you should provide any timing diagrams which are necessary to explain the timing assumptions made for these signals.

2. A Data-Flow diagram showing all of the components of your circuit and showing the control points. Naming convention will be Name for signals which are asserted as high levels and Name' (or overbar) for signals asserted by low levels.

3. A Control-Flow diagram which will be a State Table or Diagram showing how the control points in your data-flow diagram are generated and what sequences are specified. All state assignments and output and input encodings should be specified.

4. Circuit diagrams - on vellum, with pinouts labeled and a board layout specified in the normal manner.

5. Wire lists on the provided forms.

6. Descriptive material explaining your circuit diagrams and referring to a particular diagram by page number. This material will be used by the TA in an attempt to understand your circuit.

7. Conclusions - a personal evaluation of how well you think your circuit was designed and including comments on what you would do differently if you were to redesign it.

Except for the vellums, wire lists and board layouts, which will be kept in a separate 3 ring binder, all documentation will be in a lab notebook. (Preferably the spiral bound 22-157 Computation book).

You should use your lab notebook not only for the final documentation of each project, but also as a record of the design and debugging work that goes into the project. Save a page or two at the beginning of the book to use as an index to indicate where the formal, final documentation for each lab is located. This portions of the book should be neat, legible and orderly. Other sections need not be. By keeping <u>all</u> of the work that goes into the project in one place, you won't lose it, and you will have a record both for yourself and your TA of the process by which you arrived at a design and how debugged it.

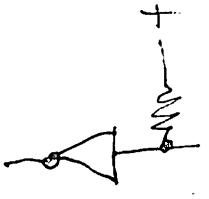
There is no point in gluing copies of the lab handouts into your lab book. The TA's have read that already (presumably) and are not apt to be impressed by your ability to use a glue stick. If you want a copy of the lab available for reference, stick it in your ring binder.

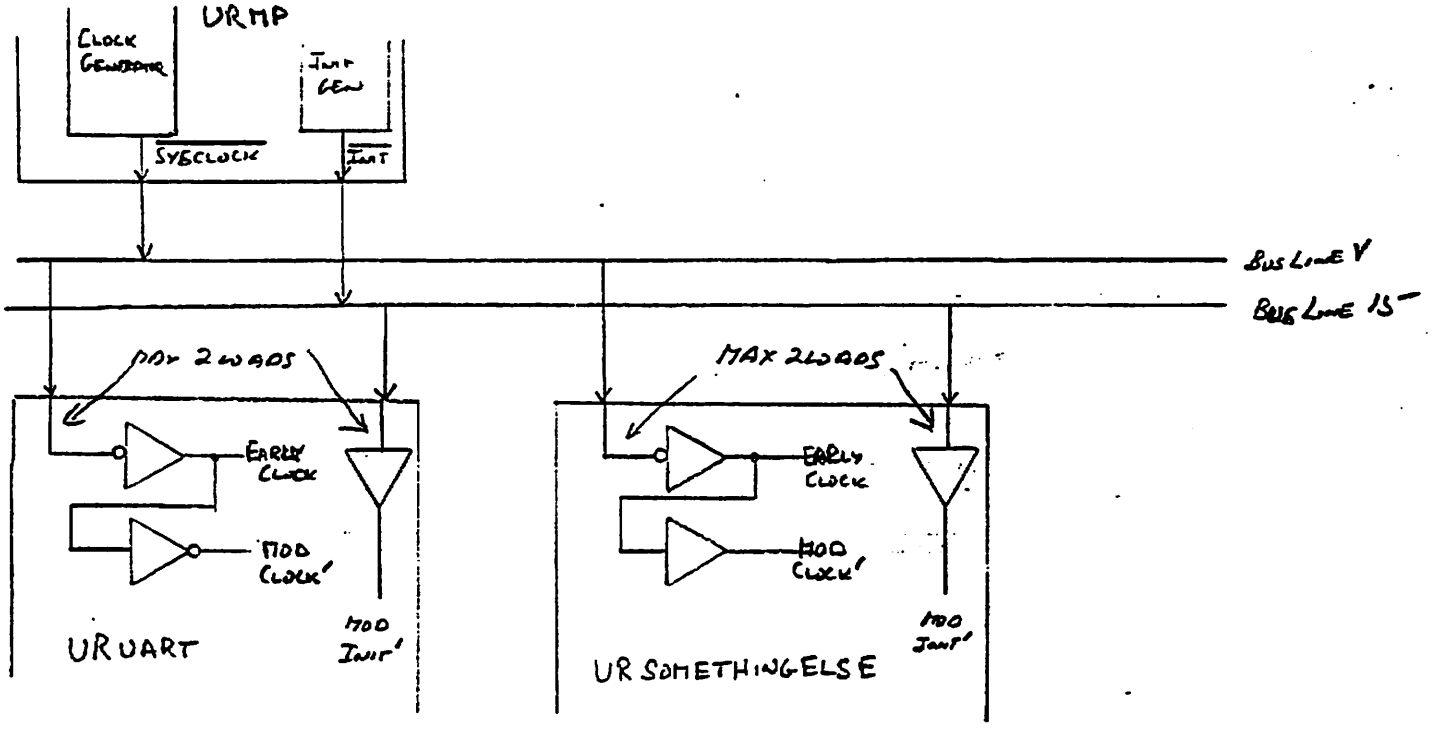
Bus Signals

The signals that should be distributed on the bus from the Clock generator board to the UART board are:

SYSCLOCK' — A negative going 614.4 KHz clock with a 1/3 duty cycle (low for approx. 543 nsec and high for approx. 1085 nsec). This signal will be inverted and buffered in each module of URSYSTEM to become EARLYCLOCK which in turn will be reinverted and buffered to become MODCLOCK', the "normal" clock for the module. This signal will appear on pin V of the edge connector. Each module will be allowed to impose a maximum of 2 CMOS loads on this signal.

INIT' — The master clear signal for the machine. This signal will appear on pin 15 of the edge connector. Each module will be allowed to impose a maximum of 2 CMOS loads on this signal.





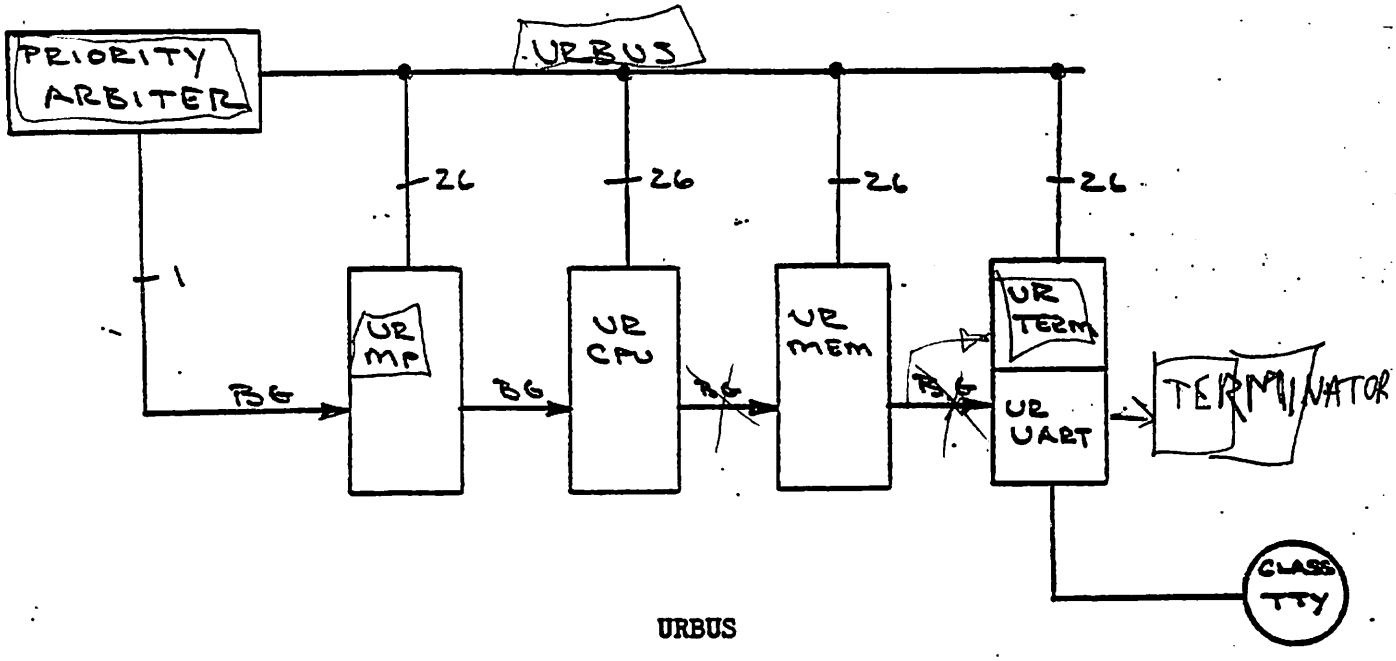
CS428 Lab 2: URBUS & URMP

Design Review: Friday April 15 (Sign up for a time

Working URBUS Hardware Due: Friday April 22 All Lab 2 Hardware Working: Thursday April 28 Lab Books Due: 4 P.M. Friday April 29)

INTRODUCTION

In the previous lab you built a device which allows a standard terminal to communicate with an 8 bit parallel path, and in the next lab you will build a processor and a memory. In this lab you will build a communication channel which will allow these three devices to communicate with each other via a particular protocol. In addition you will build another device which will serve as a maintenance panel for your system to be. The communication channel will hereafter be referred to as URBUS, and the maintenance panel as URMP. You will also be given a handout on the UNIBUS. URBUS is actually a scaled down UNIBUS with smaller data and address paths, and with only a single priority level. The protocol by which URBUS will allow communication is exactly the same as that of the UNIBUS. The UNIBUS is the main communication path in the tremendously successful DEC PDP-11 series. The UNIBUS concept has had a major influence on modern computer architecture. You should be familiar with the Unibus from CS322, but we will briefly go over the Unibus handout in class to refresh your memory. A block diagram of URSYSTEM is:



URBUS

NOTE: Since we will be using a wired-OR feature of the CMOS Open Drain 74C906 parts to deal with the major bus control signals, it is advantageous to use low signal levels to indicate a particular action (i.e. negative logic). In order to be compatible with the handout we will adopt a negative logic

descriptive style. That is, "assert signal X" means to pull X low. Similarly "negate signal X" means to pull it high.

URBUS consists of a number of bus wires which are connected to all of the devices on the bus as shown in the previous figure. One particular wire, BG for "Bus Grant" is daisy chained. It will be seen that this daisy chain technique actually creates a priority for the bus devices based on the physical position of the device in the daisy chain. URBUS consists of 27 wires which functionally fall into 3 categories.

1. INITIALIZATION - The INIT signal asserted low when asserted will master clear every device in URSYSTEM. This will be a normal CMOS output which will be generated by the maintenance panel.

2. DATA TRANSFER -

   a. 11 address lines - These lines will be CMOS Tri-State. Because the Tri-state drivers that you will be using are non-inverting, the address lines will be asserted high (positive logic). The address map will be as follows:

      - 0---------- Addresses in RAM
      - 10000000000 Terminal Status
      - 10000000001 Terminal Data
      - 101-------- URMP
      - 110-------- URCPU

   b. 8 data lines (Tri-State CMOS, asserted high) - these lines allow an 8 bit data item to be transferred in parallel.

   c. CONTROL (74C906 Open Drain assterted low) - true indicates data is to be transferred from master to slave, and false indicates data is to be transferred from slave to master.

   d. MSYN (74C906 Open Drain asserted low) - the master synch pulse

   e. SSYN (74C906 Open Drain asserted low) - the slave synch pulse

3. PRIORITY ARBITER -

   a. BBSY (74C906 Open Drain asserted low) - Bus busy

b. BR(74C906 Open Drain asserted low) - Bus request

c. BG - Bus grant normal CMOS signal asserted high

d. SACK (74C906 Open Drain asserted low) - Selection acknowledge

The URBUS basically consists of two activities:

1. Selection of the bus master, and
2. Transfer of data between established master and slave pairs.

To select a bus master the following protocol must be followed:

1. Any device may assert BR at any time it wants to become bus master. DON'T LOOK FOR SACK, LOOK FOR BG.
2. Any time SACK is negated and BR is asserted the Priority Arbiter of URBUS asserts BG.
3. Any device which receives a BG when its BR is asserted knows that it will be the next bus master and indicates this by asserting SACK and negating BR. Such a device must not allow the BG signal to pass through to the next device on the daisy chain.
4. After seeing the SACK, the Arbiter negates BG.
5. When a device knows that it will be bus master (namely that it is asserting SACK) then when BBSY is negated it may then assert BBSY and then negate SACK.
6. The master may then use the bus for as long as is desired. When the device is done with the bus it negates BBSY.

Notice that this scheme allows master selection and data transfer to happen in a concurrent pipelined fashion. This overlap can greatly enhance the performance of URSYSTEM.

Data is transferred over URBUS between the master and a slave. The master uses the address lines to select the slave device (indicated as the two high order bits of the address lines). The master drives the CONTROL line when it drives the address lines. When CONTROL is asserted (low), the master is sending data to the slave as follows:

1. The master asserts CONTROL and drives the address lines with the proper address.

2. The master drives the data lines with the proper data.

3. The master asserts MSYN.

4. The slave upon seeing MSYN and CONTROL asserted , latches the data and asserts SSYN.

5. When the master sees SSYN asserted, it negates MSYN and removes the data.

6. When the slave sees MSYN negated, it negates SSYN.

7. This cycle may proceed as many times as necessary. Note however, that only 1 data byte may transfered per cycle.

When CONTROL is deasserted (high) the master gets data from the slave as follows:

1. The master deasserts CONTROL and drives the address lines with the proper address to select the slave.

2. The master asserts the MSYN.

3. When the slave sees MSYN asserted and CONTROL high, it drives the data lines and then asserts SSYN.

4. When the master sees SSYN asserted, it latches the data and then negates MSYN.

5. When the slave sees MSYN negated, it removes the data and negates SSYN.

6. The whole cycle may occur all over again.

NOTE: All of the lines except BG are tristate lines.

Examples of timing diagrams of both data transfers and arbitration are shown in the UNIBUS handout. You are to build URBUS and the associated priority arbiter.

1 URMP

You are to build a maintenance panel for URSYSTEM which will be a device on URBUS and have the following switches and lights.

1. A master clear switch which is tied to INIT on URBUS.
2. 11 address switches for driving the address lines.
3. 8 data switches for driving the data lines.
4. A Read/Write' switch which when high indicates URMP wants to read the indicated address. When it is low it indicates that the data on the switches is to be placed into the indicated address.
5. On a URMP read the read data is placed in a latch which drives the 8 data lights.
6. A go switch which on a positive edge indicates to URMP to do the indicated action.
7. A RUN light which is actually MSYN' and can be used to show active transfers.

Note that for URMP to work properly, it will be necessary for you to buil a controller which will interface it to URBUS and the associated protocols.

As described, URMP is a master only device, that is all data transfer involving the maitenance panel are controlled by URMP. If you are bored an would like a little something extra to do, you can implement a slav capability for URMP. There will be one read/write register, which should have address 10100000000. Reading this address gives the switch data while data written to this address will appear on the lights. What you will have implemented is a control panel.

2 URTERM

Build a controller which will sit between URUART and URBUS to allow a terminal to communicate with the other devices in URSYSTEM via URBUS. The controller should conform to the specs of this lab and convert the request and acknowledge signals from URUART into URBUS signals for transfer over the bus. One of the addressed locations is the terminal status register which in its low order bit will have DRDY (Data Ready) which is actually the RCV.REQ from URUART. The next to low order bit will be TBE' (Terminal Buffer Empty NOT) which is actually the XMT.ACK. Writting data to the terminal data register

will generate XMT.RQST and reading data from it gives RCV.ACK. Design your controller so that in transmitting a character from the terminal to another submachine the bus will clear (i. e. SSYN will negate) before XMT.ACK goes low, in this way the whole URBUS will not have to wait on URUART to complete its transfer of serial I/O.

3 Cable Formats

In order to have some standardization the following cable formats will be used to transport the URBUS between your cards. Note the labels on the cards. You will not actually be using the cables but the edge connector pins that correspond to the listed cable position. Examine your circuit boards and you will see the correspondence.

3 - Data7
4 - Data6
5 - Data5
6 - Data4
7 - Data3
8 - Data2
9 - Data1
10 - Data0

URBUS CABLE #1

C - Addr10
D - Addr9
E - Addr8
F - Addr7
H - Addr6
J - Addr5
K - Addr4
L - Addr3

11 - Addr2
12 - Addr1
13 - Addr0
14 - Control
15 - INIT
16 - MSYN
17 - SSYN
18 - BBSY

URBUS CABLE #2

M - BR
N - BG IN
P - SACK

R - unused BOUT
S - unused
T - unused
U - unused
V - unused for URBUS but you will use to pass SYSCLOCK'

By using these standard connections you should be able to hook your processor to someone elses MP and run things!

Ah the old Computer Science Adage "Let's get loaded and link"! !

4 Final Remarks

All of the Bus Lines other than address and data lines are NEGATIVE logic (a 0 means do it), and the address and data lines are normal positive logic.

This is not as bad as it looks, but it is a serious design so start now, ask questions, and make sure that your design is quite solid by the design review. Take advantage of the TA hours and get in several discussions on your design before next Friday.

URTERM and URMEM will never be bus master and will always be used only as slave devices. URCPU and URMP may be either a master or a slave device.

Implement your controllers as synchronous finite state machines so that your design can be modified as easily as possible. Experience has shown that you will have to fix a couple of major screw ups before you get this one working - so give yourself the benefit of the doubt and DO NOT WING IT!!!!!! If you do a seat of the pants design on this one, you will undoubtedly take it in the seat of the pants.

Try to minimize board space again, if possible try to fit URMP and URBUS on a single card ~~with the URUART and its bus interface controller on another~~ card. After this lab we will take a survey of how much board space and component inventory has been consumed.

This is a real problem of almost professional caliber so try to have fun, and don't let confusion get you down - deal with it by discussion with other class members, TA's, or anybody else who will listen to you.

CS428 Lab 3: URCPU and URSTORE

May 5-6: Design Review: Complete URSTORE design, preliminary URCPU hardware and microcode design, including partitioning of URCPU onto two boards.

May 12-13: Design Review: Complete URCPU hardware and microcode design.

May 13: Demonstrate that URMP can read and write to URSTORE as well as to the terminal.

May 20: Demonstrate that your microcontrol can fetch, interpret and execute a single macroinstruction.

May 27: Demonstrate that URCPU can get a character from the terminal, add 1 to the ASCII code, put it in memory and send a copy back to the terminal. Successive terminal input characters should be stored in sucessive memory locations. Also show that the value can be read from memory by URMP.

June 1: Demonstrate that your machine can execute a simple test program that will be given to you.

June 3: Kits turned in and Lab Books due. Details on how the kits should be turned in will be given to you later, but the procedure is the same as it was last quarter.

INTRODUCTION

In this lab you will finish off URSYSTEM by adding a URBUS compatible processor and memory. We will of course call them URCPU and URSTORE (nothing like flogging a dead horse). URCPU is similar in structure to the simplified 11/04 that you wrote microcode for in CS322 last quarter, but has an 8 bit wide CPU. The instruction set is adequate (hopefully) but substantially simplified from what you would find in a real microprocessor. The URSTORE unit will be a 1024 by 8 bit storage unit which is addressed in the normal URBUS convention.

URSTORE

URSTORE will be a slave-only device on the URBUS and will consist of 1024 8 bit words. You will use two 2114 memory chips to implement the store - they are 1K X 4 chips. TRY to get the memory on the URTERM board. Space is going to be at a premium.

URCPU

URCPU will be a simple 8 bit microcoded processor which will communicate over URBUS in the standard way. URCPU has a single accumulator and only does absolute addressing. The only other registers which must be in the machine

are:

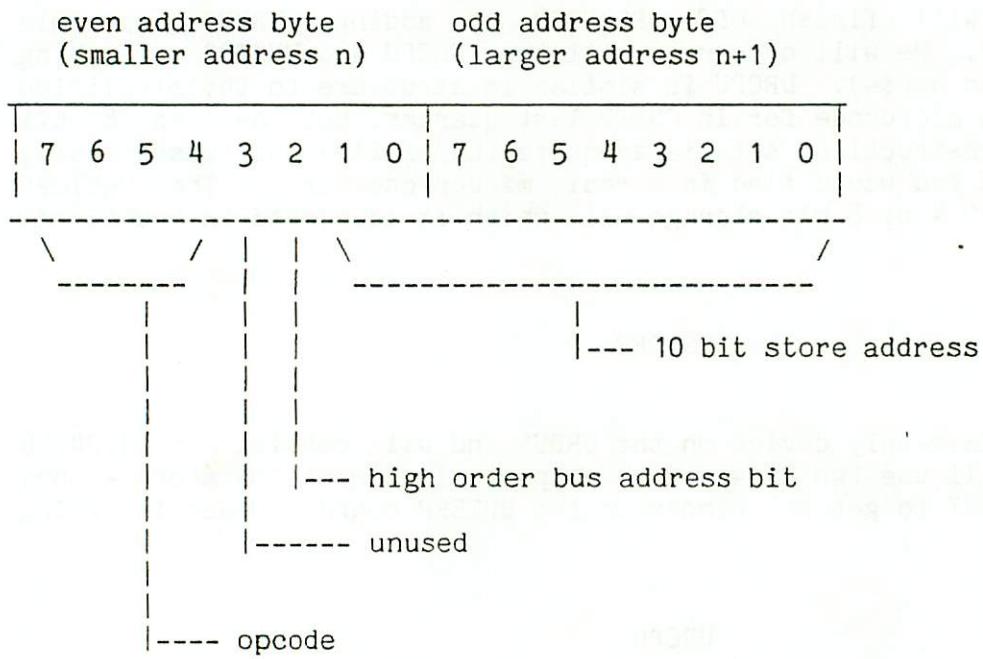
- IR - an instruction register which is used to hold the current opcode which is being executed.
- AR - a register which is used to hold the bus address for bus read and write operations.
- PC - a register which is used to hold the address of the next instruction which will be executed.

It is suggested that two additional registers be included:

- T - a temporary register
- PS - a register which holds status bits

URCPU on an INIT signal from URMP must set the PC to 000 (hex) and begin executing program from that location.

The instruction format is always a 2 byte instruction beginning on an even numbered location. The format is:



Note that certain instructions don't need an address, and in this case the second byte is just wasted space.

The instruction set and opcodes are:

C BIT 0
N BIT 1

Hex opcode	Symbol	Meaning
0	LDA	Load accumulator with addressed data
1	STA	Store accumulator to addressed location
2	LDI IM	Load accumulator with value in the address field
3	STI INDIR	Store accumulator to the address in the location(s) addressed by this instruction
4	ADD*	Add the addressed data to the accumulator
5	AND*	Logical and addressed data to the accumulator
6	NOT*	1's complement accumulator
7	ASL*	Shift accumulator 1 bit left, shift 0's into low order bit, high order bit shifts into carry
8	ASR*	Shift accumulator 1 bit right, the current carry is shifted in from the left, and the low order bit is shifted into the carry
9	CLC*†	Clear carry
A	STC*†	Set carry
B	BRA	Branch to address
C	BRC	Branch if carry = 1
D	BRZ	Branch if accumulator = 0
E	BRI	Branch indirect
F	JMS	Jump to subroutine. Store PC at addressed locations, start execution at addressed location + 2

All of the arithmetic is done 2's complement.

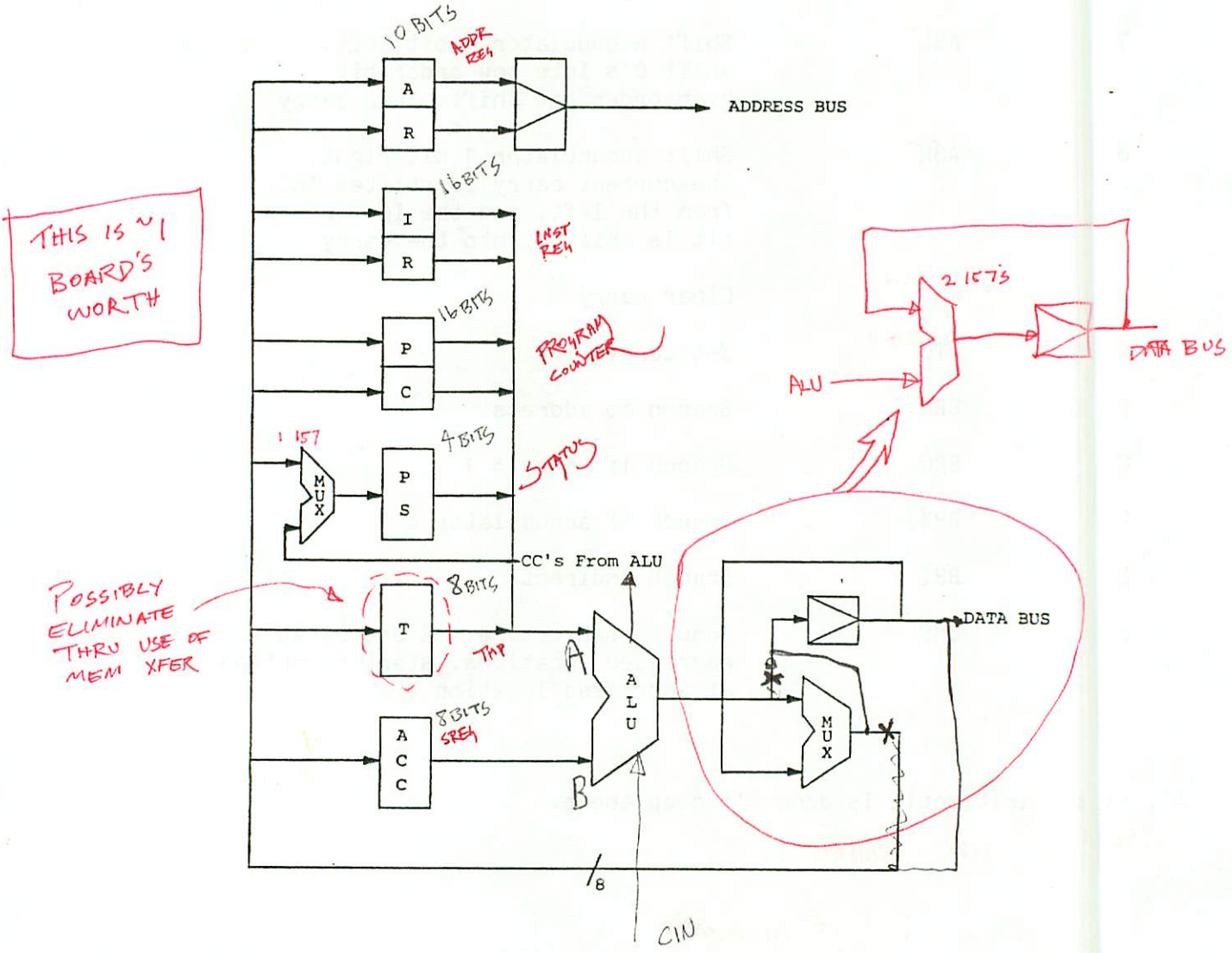
NOTE 'OR' INST DROPPED

† LEAVE N ALONE
SET CC'S

SUGGESTED ARCHITECTURE

Shown below is a suggested dataflow for URCPU. This is only a suggestion, you are free to rearchitect the machine if you like. At minimum, you should determine that the suggested dataflow is capable of implementing the required instructions. AR, IR, and PC are 2 byte registers with a high and low byte. PS need only be a half byte register. The status bits that you probably need to keep track of are:

- C - The carry (out of and into) the ALU.
- Z - This is the F=0 output from the ALU.
- N - This is the sign bit of the ALU output (the most significant bit).
- SI - This is the single instruction bit. It comes from a switch on URMP and indicates that URCPU should stop after the execution of each macro instruction.



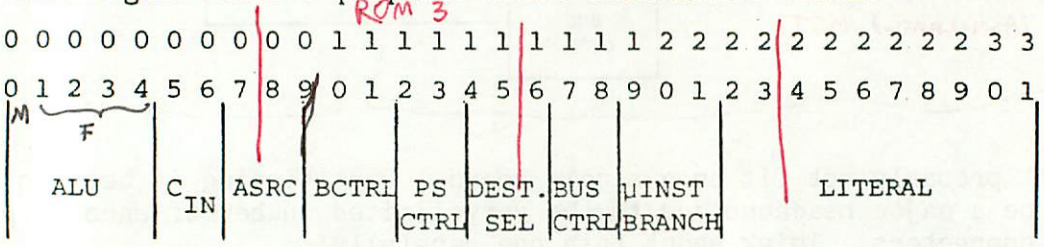
The PS register can be loaded either from the URCPU internal bus or from the condition bits generated by the ALU and URMP.

T is the temporary register. It is 1 byte. The ALU A input registers will have to be implemented such that only one of them is enabled to the bus at a time. The outputs of the PS register and the upper byte of the IR register will need to be available to Micro controller all of the time.

The Accumulator is the single register on the ALU B input. It will need to have the ability to shift right (high -> low). You will probably want to have the PS C bit shift into the high order bit of the B register.

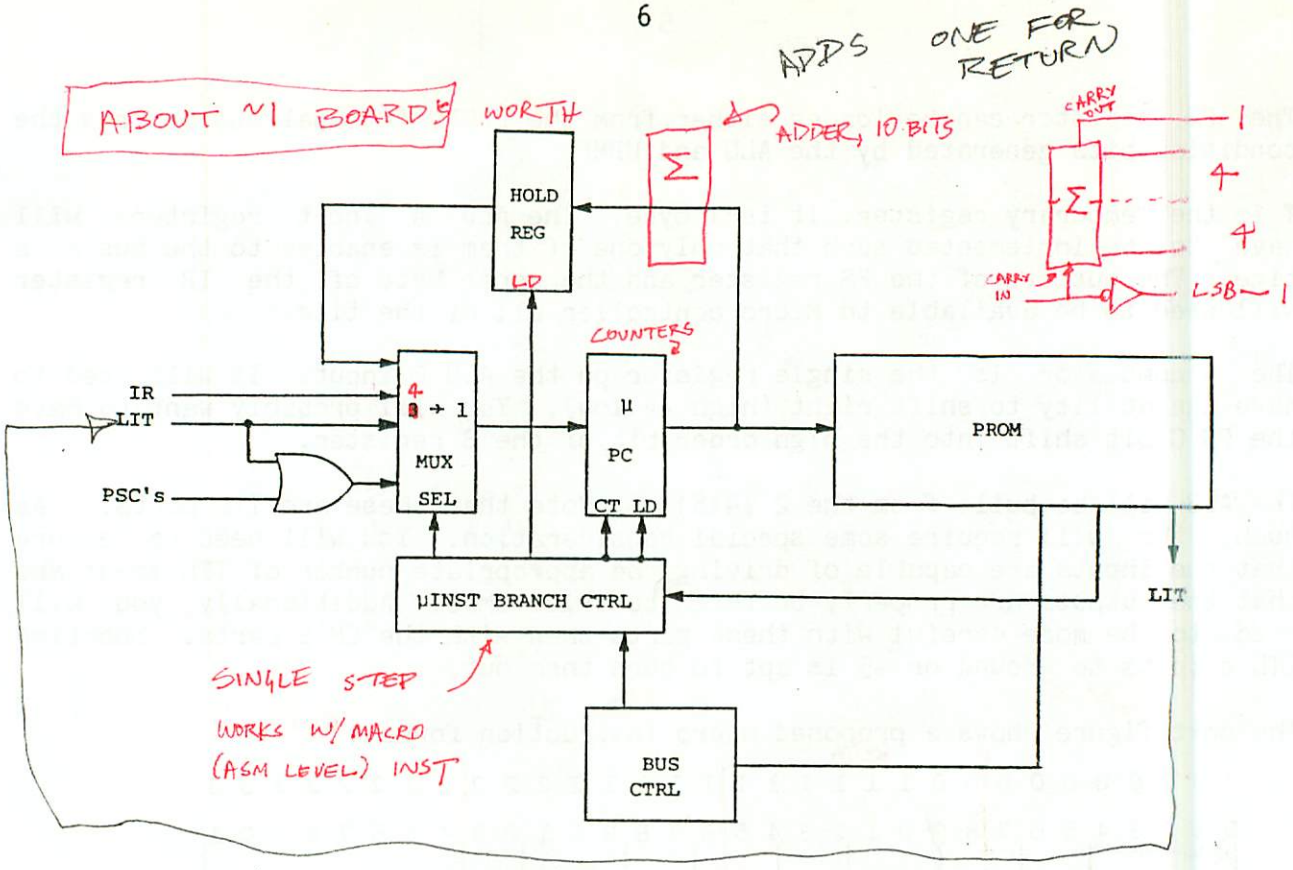
The ALU will be built from the 2 74181's. Note that these are TTL parts. As such, the will require some special consideration. You will need to be sure that the inputs are capable of driving the appropriate number of TTL loads and that the outputs are properly buffered to drive CMOS. Additionally, you will need to be more careful with these parts then with the CMOS parts. Shorting TTL outputs to ground or +5 is apt to burn them out.

The next figure shows a proposed micro instruction format.



- ALU - Controls 181's. See table in datasheet.
- Cin - choice of Cin = 0, 1, PS C bit, or PS C bit'.
- Asrc - choice of IR high or low, PC high or low, PS, or T.
- Bctrl - nop, load, or shift.
- PSctrl - nop, load from bus, or load from ALU CC's.
- Destination - AR high or low, IR high or low, PC high or low, T, or nothing.
- Bus Control - nop, read, or write.
- u-inst. Branch Ctrl - inc pc, go to lit,go to lit saving PC in HR, go to lit or'ed with PS bits, go to IR, go to HR.
- Literal - 10 bit branch address.

The final figure shows a block diagram of a micro controller. The u-inst. Branch Control interprets the appropriate field of the micro instruction to generate the address of the next micro instruction. The signal from the Bus Control unit suspends the operation of the CPU during bus read and write operations until the bus operation has completed.



The CPU will probably not fit on a single board. Partitioning it between two boards will be a major headache due to the very limited number of unused pins on the edge connectors. Think about this one carefully!