Arbadell Computer

by Shane Avery

1. Introduction

Arbadell is a computer designed completely from scratch. The purpose of the project is to design a complete computer system to test and develop the skills acquired as a computer engineering graduate at Cal Poly. Also, the project will provide a way to show my skills to an interviewer. Arbadell contains a CPU, RAM, loader, chipset, keyboard input, RS232 interface, and VGA out. The chipset will be design by Elohim Holguin as her senior project.

2. CPU

The CPU will be a CISC like computer. Implementation of the CPU will be done in Verilog. A Xilinx FPGA was chosen for the CPU for a few reasons. First, Xilinx development software is free. I managed to get a hold of the student edition from Professor Sandiage's book. Secondly, I managed to find a development kit made by Xess that contained a Xilinx chip.

The design of the CPU contains an instruction register to hold the current instruction, an alu (arithmetic and logic unit) that performs add, logical and, logical or, invert, increment, and decrement, a PC (program counter), a stack, a scratch register, a status register, and a control unit.

The CPU is an 8 bit CPU (meaning it has an 8 bit data bus). The address bus is 16 bits. Therefore the CPU can access 65536 bytes of data. The CPU is designed to be run at 4 MHz. Upon reset the CPU will begin executing at address 0x0000. There are 24 instructions. Their names, machine code, and a short description are as follows.
Note: Literal values are 8 bits and memory addresses are 16 bits.

movel	literal value	0x00	move the literal value into the scratch register
movem	memory address	0x01	move the byte in the memory address into the scratch register
moves	memory address	0x02	move the value of the scratch register into the memory address location
inv		0x03	invert the value of the scratch register
inc		0x04	increments the value of the scratch register
dec		0x05	decrements the value of the scratch register
shiftr		0x06	logical shift right of the scratch register
shiftl		0x07	logical shift left of the scratch register

addl	literal value	0x08	add the literal value to the value of the scratch register
andl	literal value	0x09	and the literal value to the value of the scratch register
orl	literal value	0x0a	or the literal value to the value of the scratch register
addm	memory address	0x0b	add the value in the memory address to the value of the scratch register
andm	memory address	0x0c	and the value in the memory address to the value of the scratch register
orm	memory address	0x0d	or the value in the memory address to the value of the scratch register
call	memory address	0x0e	calls the subroutine at memory address and the current address will be pushed onto the stack for a return
bra	memory address	0x0f	branches to memory address
braz	memory address	0x10	branches to memory address if the zero bit has been set else just continues to next instruction
ret		0x11	return from call; will pop of the next address on stack and put it in the PC
reti		0x12	return from interrupt
iorecv	literal value	0x13	I/O receive from chipset; the value will determine the input received
iosend	literal value	0x14	I/O send from chipset; the value will determine what the chipset will do with data given to it
setie	literal value	0x15	this will enable interrupts; keyboard interrupt enable will be the value of the first bit; serial interrupt enable will be the value of the second bit; the interrupt will

			be enabled if the bit is high and cleared if low; all other values will be ignored
ti0	memory address	0x16	test interrupt0(keyboard) and branch to memory address if set
ti1	memory address	0x17	test interrupt1(serial in) and branch to memory address if set

note: All arithmetic and logical operations store the result into the scratch register.

Interrupts will be handled as follows. The chipset will assert an interrupt pin when it receives data from either the keyboard or serial port. If the interrupts are enabled the CPU will finish its current instruction, save the current memory address and branch to 0xfff0 if a keyboard interrupt and 0xfff8 for a serial interrupt. The CPU will then continue executing instructions and will not interrupt again even if an interrupt pin is asserted by the chipset until the reti instruction has been executed. Alternatively, the user may poll for interrupts using the ti0 and ti1 instructions. However, it is strongly recommended the polling not be used because the chipset will block until interrupts are serviced.

3. Chipset

The chipset was developed by Elohim Holguin as a senior project and was implemented with a PIC microcontroller. It will be involved in most input and outputs to the Arbadell computer. The chipset will be responsible for keyboard in, serial in, serial out, passing data to the video system, and latching the LED array. Whenever the chipset happens to receive a byte from serial or the keyboard it will interrupt the CPU. The interrupt cannot be cleared until the CPU requests to service the interrupt from the chipset.

The communications protocol between the CPU and chipset is a simple two signal protocol. The chipset will look at the IOADDR pin that comes from the CPU. When that pin is asserted the chipset will look at the lower three address lines to determine the action to take. The lower three bits and their action is described as follows.

0x00	keyboard in
0x01	serial in
0x02	serial out
0x03	video out
0x04	latch LED array
0x07	asks the chipset to respond with 0xa5

The last action 0x07 is way for the CPU to determine if the chipset is there and responding. It is meant to be used during the POST (power on self test).

Once IOADDR is asserted the chipset will perform the action and then assert ACK. This will tell the CPU that the operation is complete. If the chipset has put a byte on the data bus the CPU will grab it at this time. The CPU will then clear the IOADDR line telling the chipset that the CPU has received the ACK and, if applicable, has grabbed the byte from the data bus. The chipset in turn will clear ACK and the process

will start over. The CPU cannot set the IOADDR line again until the chipset has cleared ACK.

By default the chipset will always have the data bus as input until the IOADDR line is asserted the action is keyboard in, serial in, or respond with 0xa5. Once IOADDR has been cleared the chipset must then switch to input on the data bus before clearing ACK.

4. RAM

Main system memory is a CMOS NVSRAM part. This is so data can be retained even after powerdown.

5. Loader

Somehow the memory has to be initially loaded. An AVR microcontroller will load the RAM with the programs and data before turning the CPU loose on the RAM. The AVR will have an RS232 interface and will connect to the PC via the serial port. The AVR will then be able to accept commands like a dummy terminal. Commands will allow users to load specified block of RAM, read specified blocks of RAM, and pull the CPU out of reset to execute the instructions in RAM.

6. Assembler

The assembler will be written in C. A more detailed document on assembler syntax and usage will be included in the zip file along with the source code.

7. VGA Chip

It would be too complicated to make the chipset directly control the VGA interface. Instead there will be separate VGA chips. The VGA chips will be a CPLD written in verilog and a PIC microcontroller. It will accept commands from the CPU to display characters on a standard VGA monitor.