CS61c: Introduction to Synchronous Digital Systems

J. Wawrzynek

October 8, 2007

Reading: P&H, Appendix B

1 Instruction Set Architecture

Among the topics we studied thus far this semester, was the following big idea. Our high-level language programs (C, or C++ for instance) are converted to a list of assembly language instructions which in term are converted to a set of machine instructions.

C	Language constructs/data structures
↓	
assembly language	list of CPU instructions (with memory references)
↓	
machine language	list of binary coded machine instructions
↓	
?	

How are these instructions carried out? What happens to the machine instructions? One thing we can do is to read them into a simulator (such as MARS or SPIM). However, most of the time, we are interested in actually running the instruction on real hardware. How does the hardware execute our instructions? That is the topic of the remainder of the semester.

The set of opcodes that a processor can carry out, along with a description of the processor "state" (all the user visible registers), is called the *Instruction Set Architecture (ISA)*.

Software

Instruction Set Architecture

Hardware

It is an abstraction of the hardware for the benefit of the software. More precisely it is a 2-way contract between the hardware and the software.

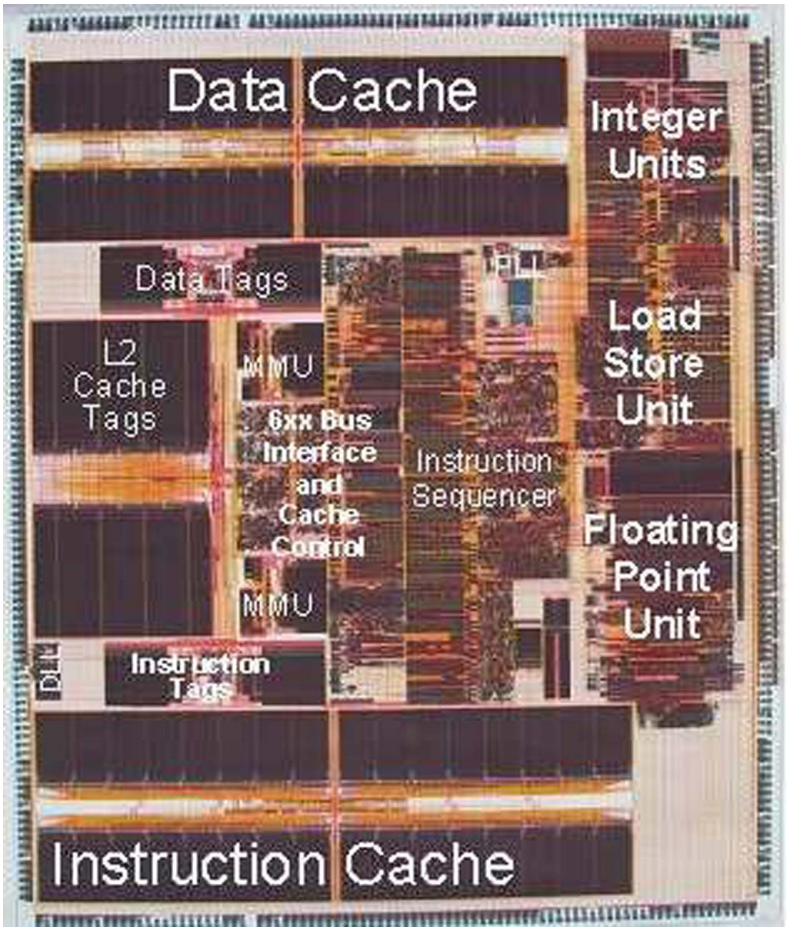
> The hardware promises that it will faithfully implement the ISA and the software promises to only try to do things supported by the ISA.

Software designers can write operating systems, compilers, and applications without fear that their software will not work on new versions of the hardware. Also, the software really doesn't know (nor should care) about how the hardware implements the ISA. Therefore the hardware designer is free to design and optimize the implementation of the processor as he/she sees fit. It can be tailored to the latest technology, or to some particular market need (a particular price or performance point for instance), as long as the design meets the contract by faithfully implementing the ISA. As new, more advanced, implementations can come along, all our old programs and operating systems will still function.

The ISA is an abstraction and doesn't specify in detail how the instruction execution should be implemented. Consequently their are many different ways to design the processor. We will study a few of the basic concepts behind processor design, leaving many of the performance enhancements for CS152.

2 The Physical Hardware

We will start by looking at a photograph of a microprocessor. This one is from a few years ago, but still is representative of today's processors.



This microprocessor happens to be a powerPC (similar to what user to be inside Apple computers). However, most processors, even ones with different instructions set, for instance the MIPS, look very similar to this picture.

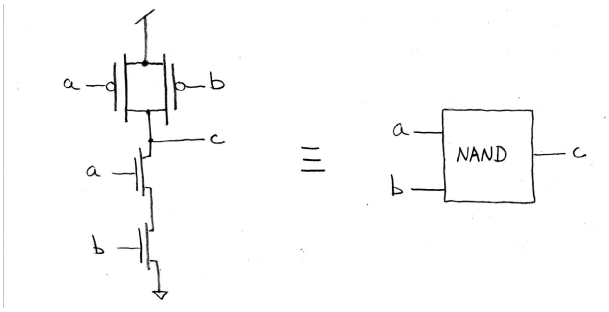
Notice the various blocks labeled on the photograph. We will be discussing these later this semester. Here are the specifications for this particular processor:

- "Superscalar (3 instructions/cycle)"
- 6 execution units (2 integer and 1 double precision IEEE floating point)
- 32 KByte Instructions and 32 KBytes Data L1 caches
- Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)
- External L2 Cache interface with integrated controller and cache tags, supports up to 1 MByte external L2 cache

The picture gives you a good idea of the relative complexity of the various sub-blocks.

In the world of integrated circuit (chip) design $area \equiv cost$. Bigger chips are much more expensive to make than smaller chips. Chip real estate is a precious commodity. Designers work hard to make each function as physically small as possible without hurting overall performance.

If we zoomed in with a microscope, each subpiece would look very similar to the others, because they are all made of the same basic stuff—wires and transistors. (In fact, we call these devices "integrated circuits" because they put together multiple transistors and the necessary wires to connect them together. Early chips (non-ICs) had a single transistor or two without the necessary wires. These days state-of-the-art ICs have transistor counts in the 10's to 100's of Millions). We are not going to go into the details of how transistors operate. Suffice it to say that small collections of transistors are joined to make simple functional blocks which in turn are joined to make bigger blocks, etc. We will stop our investigations at the level of the simplest function units, logic gates and flip-flops. Below is an example of a simple unit. We will only consider the representation on the right, not the transistor-level details. EECS40 covers transistor-level implementations in detail.



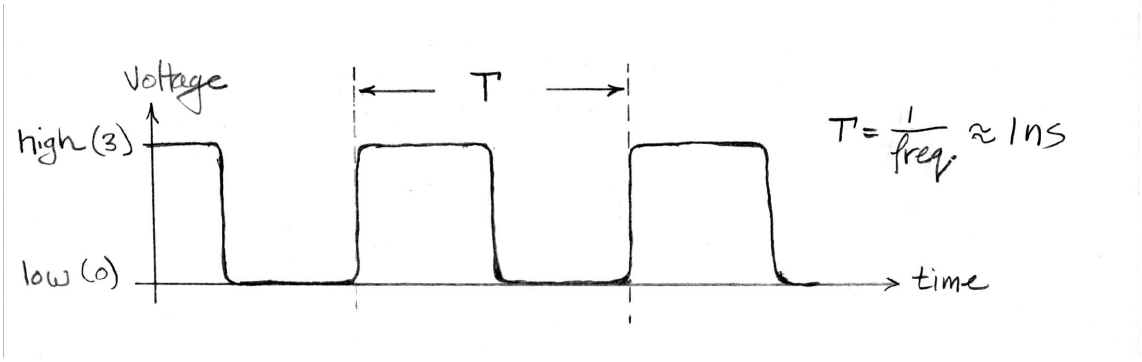
Surrounding the inner part of the chip—the core—is a set of connections to the outside world. Usually these connect through some wires in the plastic or ceramic package to the printed circuit board (PCB). In the case of most computers this PCB would be the *motherboard*. Some of these connections go to the main memory and the system bus. A fair number of the pins are used to connect to the power supply. The power supply takes the 110 Volt AC from the wall socket (provided by PG&E) and converts it to low voltage DC (usually in the range of around 1 to 5 volts, depending on the particular chip used).

The DC voltage is measured relative to ground potential (GND). Power connections to the chip from the power supply are of two types; GND, and DC Voltage (labeled "Vdd").

The energy provided by the power supply drives the operation of the processor. The energy is used to move electric charge from place to place in the circuits and in the process is dissipated as heat. These days a processor my use on the order of 10 watts—like a not-so-bright light bulb.

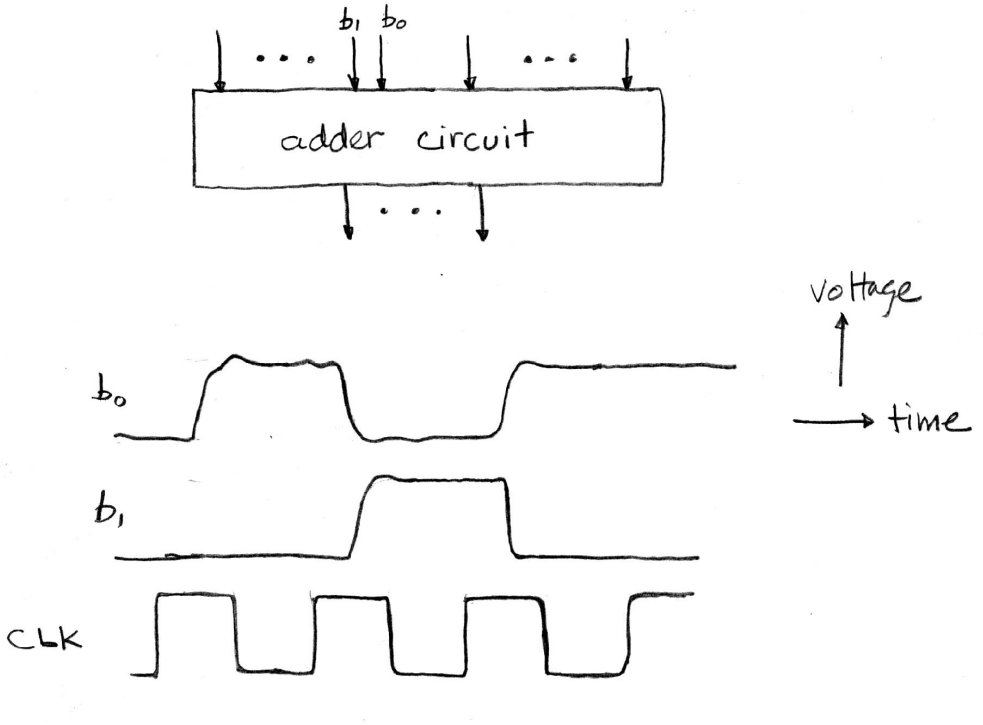
3 Signals and Waveforms

Another special connection to the chip is the clock input signal. The clock signal is generated on the motherboard and sent to the chip where it is distributed throughout the processor on internal wires. The clock signal is the heartbeat of the system. It controls the timing of the flow of electric charge (and thus information) through out the processor and off-chip to the memory and system bus. If we had a very small probe we could examine the signal on the clock wires, by looking at the voltage level on the wire. We would see a blur, because the clock signal is oscillating at a very high frequency (around 1 billion cycles/second or 1GHz, these days). If we could record it for a brief instant and plot it out we would see something like:



Note, the clock signal is very regular. It is just a repeating "square wave". One clock cycle (tick), is the time from any place on the waveform to the same place on the next instance of the wave form. This semester, we will almost always measure the clock period from the rising-edge of the signal to the next rising-edge. (The idea of rising and falling edge comes from the idea of imagining that the signal might start at a low voltage then it rises to the high voltage, stays there for half the period, then falls to the low voltage, where it stays for half the period, etc.) As we shall see later, the clock signal is a very important signal and is present in all synchronous digital systems.

If we looked around with our microscope we could look for an another interesting place to put our probe to look at other signals. For instance, we could look around for an adder circuit—no doubt there is an adder or two in the processor someplace. Suppose we found an adder circuit (how would we know its an adder?). We could put our recording probe at its input and might see a waveform as shown below. If we had two probes we could look at 2 signals simultaneously. Let's imagine that we probe the two least significant bits of one of the inputs to the adder.



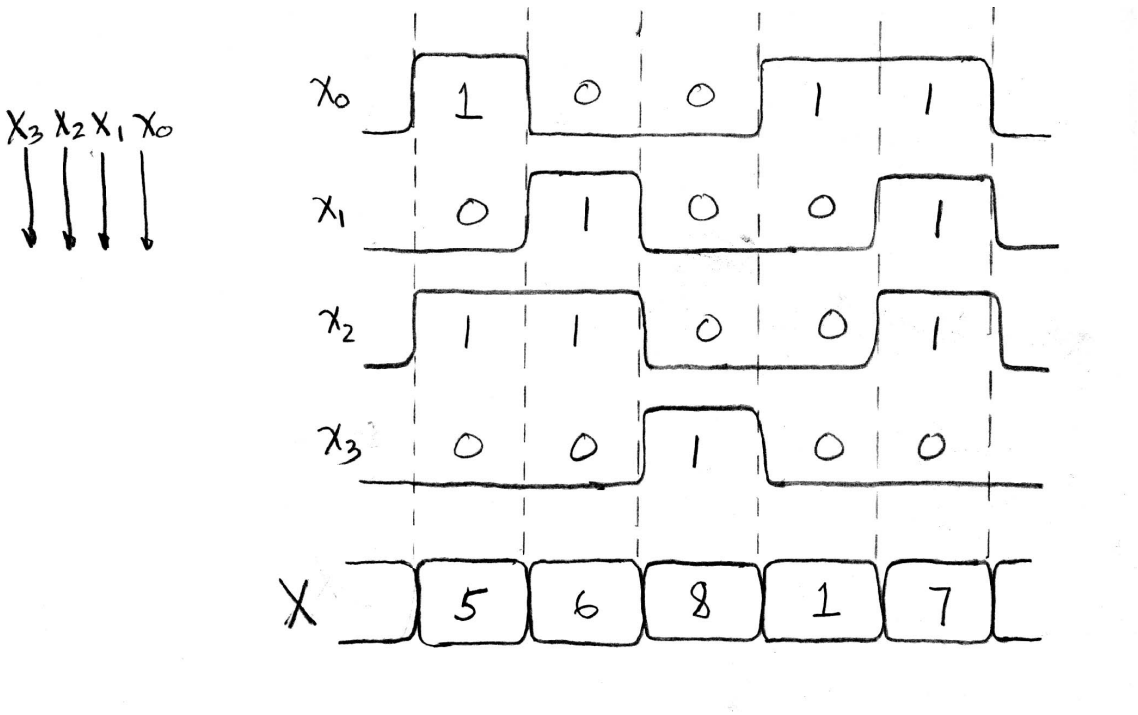
We can make the following observations based on the recorded waveforms.

1. Most of the time, the signals are in one of two positions: low voltage or high-voltage.
2. When the signals are at the high or low voltage levels, sometimes they are not all the way to the voltage extremes.
3. Changes in the signals seem to correspond to changes in the clock signal (but they don't always change on every clock cycle).

Here are the facts that explain the observations:

1. By convention, low voltage is used to represent a binary 0 and high voltage is used to represent a binary 1. At different points in time the particular bit position of the input to the adder has a value of 0 and at other times has a value of 1.
2. The transistor circuits that make up the adder and its subcircuits are robust against small errors in voltage. If the signal is close to high-voltage it is interpreted as a 1 and similarly if it is even close to 0 volts, it is interpreted as a logic 0. Therefore errors (or deviations from ideal voltages) do not accumulate as signals move from circuit to circuit. The circuits have the property that they take input signals that may not be ideal, in the sense of being exactly at the voltage extremes, and produce outputs at the voltage extremes. This restoration property is a critical property of digital circuits.
3. In synchronous systems, all changes follow clock edges. As we shall see later, the clock signal is used through out the system to control when signals may take on new values.

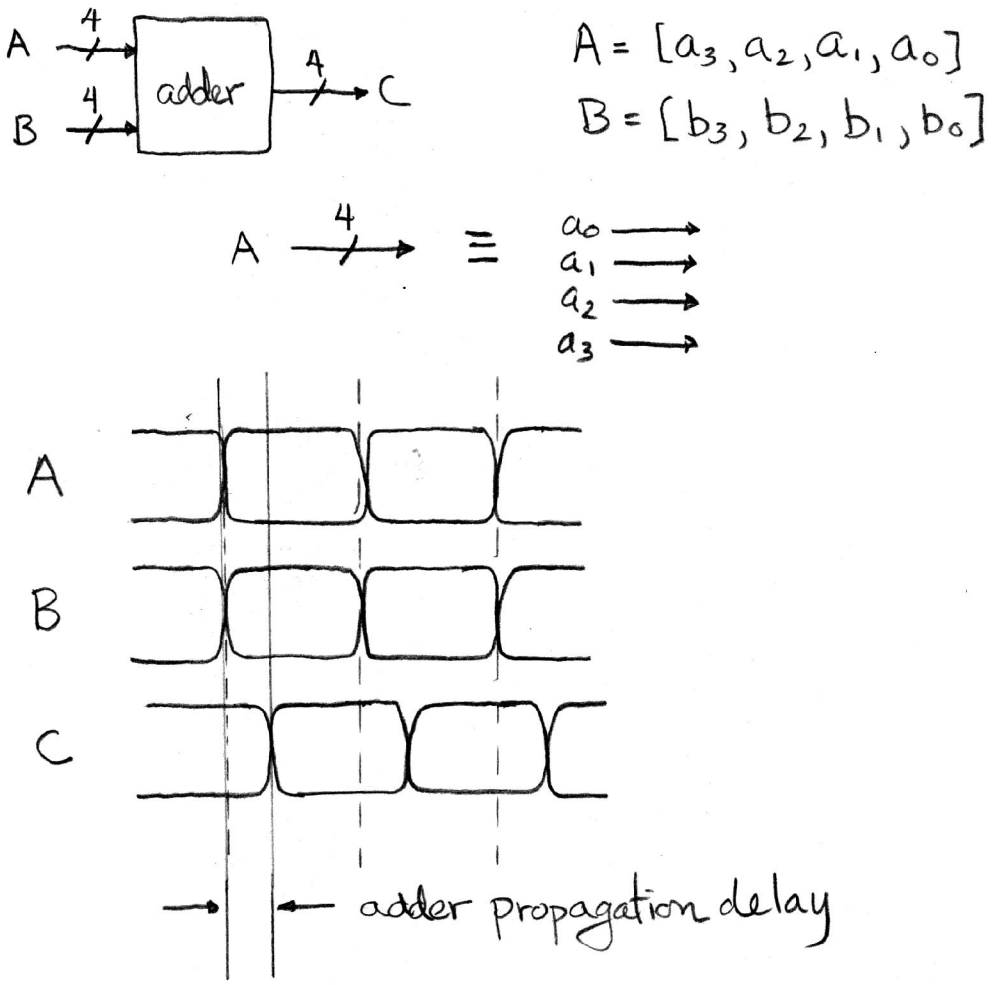
Signal wires are often grouped together to transmit words of data, as illustrated below:



The drawing shows a collection of 4 wires used to transmit a 4-bit word. The waveforms show how the signals on the wires might change over time. Lower-case letters are used to label the individual wires (and signals) and the corresponding upper-case letter is used to label the collection of wires (sometimes called a bus). Drawing a single waveform that represents all the signals on the bus is difficult because at any instant each wire could have a different value. If we need to show the timing of the changes of the signals on a bus, we can do what is shown in the figure. Here we draw the bus signal as being simultaneously high and low and label each closed section with the value on the bus at that instant. In addition to the values on the bus, the important information shown in this sort of drawing is the timing of the changes.

4 Circuit Delay

Suppose we have a circuit that adds 2 4-bit numbers, as shown below:



Another important property of digital circuits is that they cannot produce a result instantaneously. The adder shown above cannot produce the sum instantaneously, so if we examined the input and output waveforms, we would notice a small delay from when the inputs change to when the output has the new result. The delay is present because it takes time for the electrical charge to move through the wires and the transistors. A measure of the delay from input to output, in general, is called the *propagation delay*, and in this case is called the adder propagation delay. The absolute value of this delay number is dependent on many things, including the particular IC technology used to fabricate the chip, details of the internal circuitry of the adder, even the value of the input numbers (some numbers are quicker to add than others). However, the propagation delay of a circuit, like the adder, is always less than the clock period.

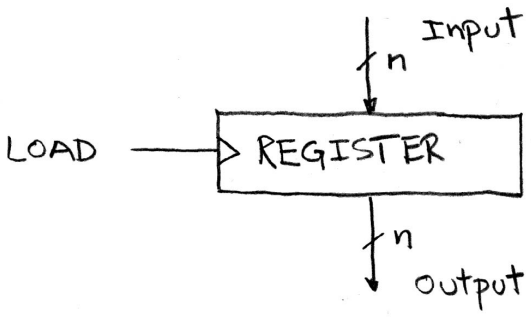
5 Types of Circuits

The adder circuit is an example of a type of circuit we call "combinational logic" (CL). These circuits take their inputs and combine them to produce a new output after some small delay. The output is a function of nothing other than the inputs—like a mathematical *function*. Combinational logic circuits have no memory of their previous inputs or outputs. Other examples of combinational logic circuits that you might recognizeinclude, value comparitors (takes two words and outputs a signal indicating if they are the same or different), data selectors (passes one of several inputs to its output), and all sorts of arithmetic circuits.

Combinational logic circuits are used throughout processor implementations. They provide all the necessary functions needed to execute instructions, including all the arithmetic and logic operations, incrementing the program counter value, determining when to branch, calculation of new PC target values, etc. Not counting the area consumed by caches, most of the area on microprocessor chips is used for combinational logic blocks.

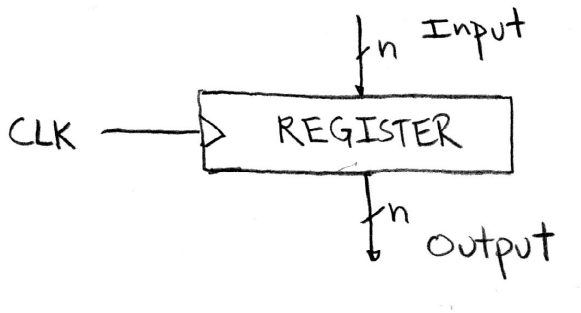
The other type of circuit used to make processors (and other hardware systems) are circuits that remember their inputs (in CL, inputs are only used to generate outputs, but never remembered). An important example of this type of circuit is the memory circuit used to implement the general purpose registers within the MIPS. Take register $1 for instance. Somewhere on the MIPS chips is a register that gets written to when the destination of an instruction is $1 and gets read from when the source operand for an instruction is $1.

Below is shown what register $1 might look like:



Under the control of a special input, called LOAD (don't confuse this with the MIPS instruction LD), the register captures the value of the input signal and holds onto it indefinitely, or until another value is loaded. The value stored by the register appears at the output of the register (a very short time after it gets loaded). So, if the processor puts a value in $1 at one point in time, it can come back later and read that value. Even if different values appear at the input to the register in the interim, the output value of the register will not change, unless instructed to do so by signaling with the LOAD input. So as long as the processor has control over the signaling on the LOAD input, it has total control over what gets saved and when.

Often the system clock signal (CLK) is used as the LOAD signal on a register:



This configuration means that for this register a new value is loaded on each clock cycle. We will see next time why this configuration is valuable.