Computer Science 010
Introduction to Computer Applications
Siena College
Fall 2010

Topic Notes: Von Neumann Architecture

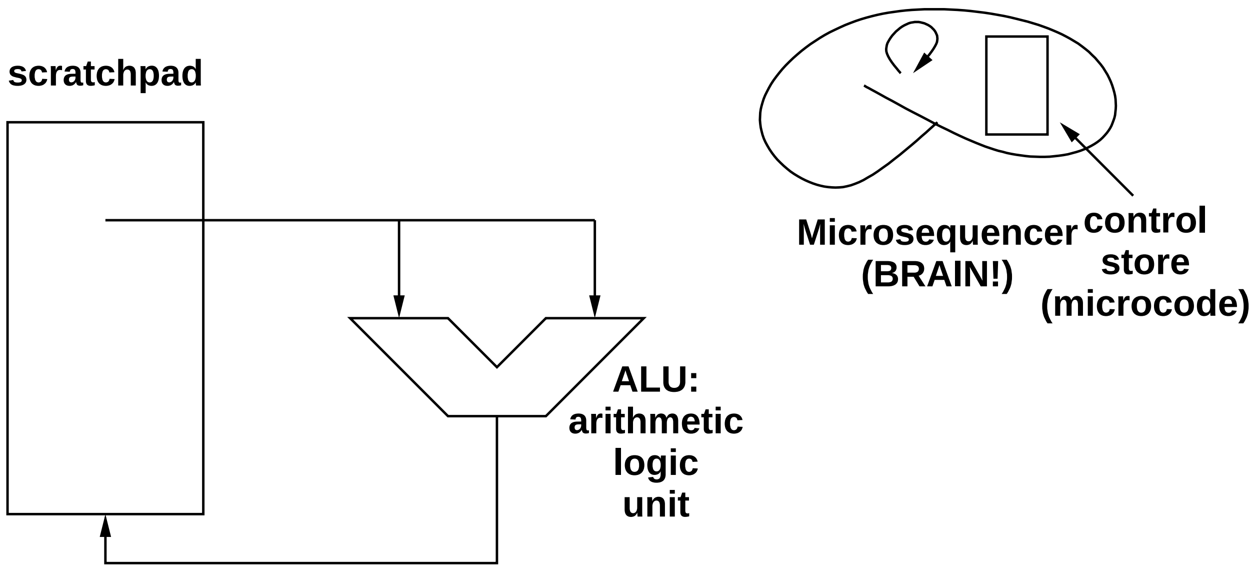
Architecture Basics

Modern computers use the *vonNeumann architecture*.

Idea: a set of instructions and a loop:

1. Fetch an instruction
2. Update next instruction location
3. Decode the instruction
4. Execute the instruction
5. GOTO 1

Basic picture of the system:



The ALU knows how to do some set of arithmetic and logical operations on values in the scratch-pad.

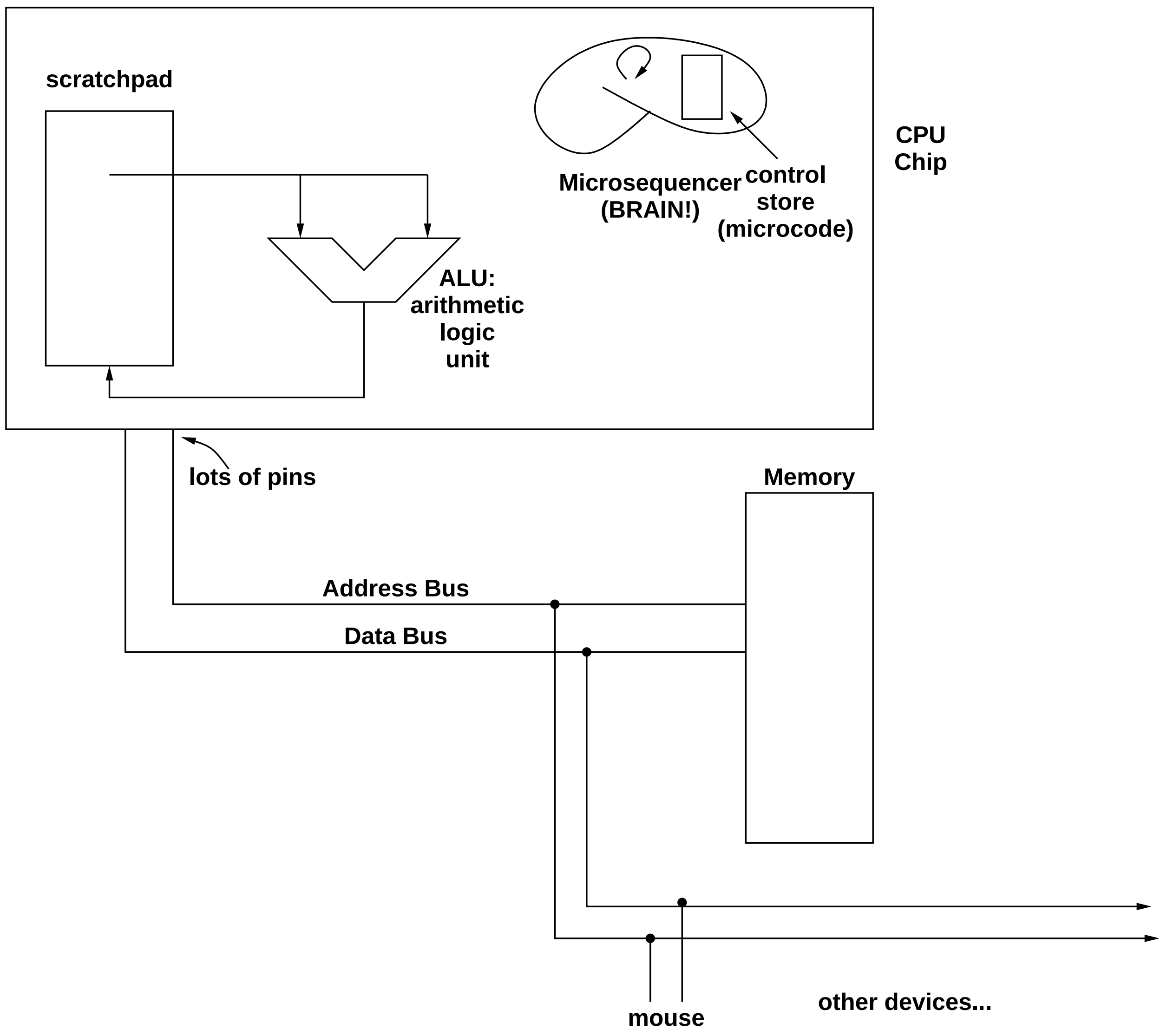
Usually the scratchpad is made up of a set of *registers*.

The micro-sequencer "brain" controls what the ALU reads from the scratchpad and where it might put results, and when.

We will not be concerned about the details of the micro-sequencer.

This is what makes up the *central processing unit (CPU)*.

We can expand this idea a bit to include memory and other devices.



The CPU interacts with memory and other devices on *buses*.

These buses are just wires that carry the electrical signals that represent the data.

If you dig deep down into any modern computer you use, it boils down to this basic process.

The SIENAVAC

The study of a real architecture would go beyond the scope of this course, but we can get the idea by considering a simple hypothetical computer we'll call the SIENAVAC, which stands for

"**SIENA**'s **V**ery **A**ncient **C**omputer".[1]

Note that this name is a play on "UNIVAC", which was the name of many of the earliest commercial computers. There, it was short for "**UNIV**ersal **A**utomatic **C**omputer".

Before we can continue, we need to specify some rules about how SIENAVAC works.

We need to know the set of instructions that SIENAVAC can execute. It has only 8, and we will use 3-bit binary codes to represent the 8 operations.

000	`HALT`
001	`LOAD`
010	`STORE`
011	`ADD`
100	`SUBTRACT`
101	`MULTIPLY`
110	`DIVIDE`
111	`JUMPIFZERO`

Like other computers in the earliest days of computing, SIENAVAC contains a main panel of lights. When these computers ran programs, those lights could tell a trained observer exactly what was happening inside the system at the time. What these lights were showing were the current values in the various computer's registers. These registers were used to hold arithmetic operation results, current program address and operation-code values and any other values that were vital to the computer's operation. Today's computers all have these same types of registers, but computer designers no longer feel it is necessary to display them in lights. Compared to the computers of yesteryear, today's computers have a very boring appearance without those light displays!

These registers on SIENAVAC are:

- `ACCUMULATOR` – an 8-bit value that stores operands and results of instruction execution.
- `PROG ADDR` – a 5-bit value indicating the location of the next instruction to be executed.
- `OPERATION` – a 3-bit value representing the current operation (from the table above).
- `OPERAND ADDR` – a 5-bit value indicating where to find an operand in memory.
- `OPERAND` – an 8-bit value to be used as an arithmetic operand.

There is also memory to store data and programs on SIENAVAC. It is incredibly small by today's standards (or any standards, for that matter): it contains only 32 8-bit values. Each of these values is referred to by an address. Since there are 32 of these locations, we can number them from 0-31 and use a 5-bit number to refer to them.

[1]Thanks to Mr. Gary Cutler, a Siena CSIS-010 instructor, for this example architecture.

MEMORY						
Addr	Contents			Addr	Contents	
00000	001	00011		10000	010	10101
00001	011	00100		10001	110	00101
00010	011	00101		10010	000	00000
00011	000	00001		10011	011	10000
00100	000	00010		10100	111	11111
00101	000	00011		10101	011	01101
00110	101	10101		10110	001	10101
00111	011	11111		10111	010	00000
01000	100	00100		11000	000	10101
01001	011	00100		11001	110	00101
01010	011	00101		11010	000	00000
01011	000	00001		11011	011	10000
01100	000	00010		11100	111	11111
01101	000	00011		11101	011	01101
01110	101	10101		11110	001	10101
01111	011	11111		11111	010	00000

The first seven locations in the chart above constitute a simple *machine language* program which will compute the sum of the integers 1, 2 and 3 (we'll see just how in a minute).

Machine language got that name from the fact that each make and model of computer had its own (unique) instruction set, thus making it the "language" for that "machine". Getting a machine language program **into** a computer like SIENAVAC's memory was a time-consuming operation. Each of the lights on the front panel served two purposes on SIENAVAC they were switches as well mere indicator lights. When an "off" light is pressed, it turns "on". When an "on" light is pressed, it turns "off". The "off" values denote zero while the "on" values denote one.

Each one of those memory values was entered manually, as follows:

1. The desired value is "toggled" in to the `OPERAND` indicator on the panel.
2. The address into which that `OPERAND` value should be stored is entered into the `PROG ADDR` indicator on the panel.
3. The "Store" button is pressed - an action which saves the `OPERAND` value into the `PROG ADDR` location in memory.

This was repeated until the seven desired memory values were saved.

As you can guess, this process was quite tedious and is only reasonable for small programs. Today, computers still execute machine language program, but they are loaded into memory by more automatic means.

The program is executed by entering the starting address of the program 0_{10} (00000_2) - into the `PROG ADDR` field and pressing the "Run"button.

The first instruction is a `LOAD` instruction (001_2). This type of an instruction loads a value from memory into the register called the accumulator so that subsequent instructions may perform arithmetic operations against it. This is quite similar to what you do when you enter a number into the display of a hand-held calculator. In this case, the contents of memory location 3_{10} (00011_2) will be loaded.

Memory location 3_{10} contains the number 1_{10} (00000001_2), and so that value is loaded into the accumulator as SIENAVAC prepares to run the next instruction. Each time SIENAVAC completes the execution of an instruction, it automatically adds one to the `PROG ADDR` value to determine from where it will fetch the next instruction. This means that the instruction at address $1_{10}|$ (00001_2) will be the next one executed.

So after this instruction is executed, the display of lights on SIENAVAC's panel "looks" like this:

```
 ACCUMULATOR  0 0 0 0 0 0 0 1
   PROG ADDR  0 0 0 0 0
   OPERATION  0 0 1
OPERAND ADDR  0 0 0 1 1
     OPERAND  0 0 0 0 0 0 0 1
```

This is just before SIENAVAC increments the `PROG ADDR` in preparation to execute the next instruction.

The instruction at address 1_{10} (00001_2) is the next one executed. That instruction an `ADD` (011_2) will add the contents of the accumulator to the contents of a memory location, leaving the sum in the accumulator, replacing its prior contents.

In this case, the `ADD` instruction will add the accumulator's contents - 1_{10} (00000001_2) - to the contents of memory location 4_{10} (00100_2). Memory location 4_{10} contains the value 2_{10} (00000010_2), so the sum $1_{10} + 2_{10} = 3_{10}$ (00000011_2) will be saved in the accumulator.

```
 ACCUMULATOR  0 0 0 0 0 0 1 1
   PROG ADDR  0 0 0 0 1
   OPERATION  0 1 1
OPERAND ADDR  0 0 1 0 0
     OPERAND  0 0 0 0 0 0 1 0
```

Incrementing the `PROG ADDR` register to a 2_{10} (00010_2), SIENAVAC now prepares to execute the instruction at address 2.

That instruction is another `ADD` (011_2) instruction. In this case, SIENAVAC adds the contents of memory location 5_{10} (00101_2) to the accumulator.

Since memory location 5_{10} contains 3_{10} (00000011_2) and the accumulator contains 3_{10} (00000011_2) the result of 6_{10} (00000110_2) is saved in the accumulator.

ACCUMULATOR	0 0 0 0 0 1 1 0
PROG ADDR	0 0 0 1 0
OPERATION	0 1 1
OPERAND ADDR	0 0 1 0 1
OPERAND	0 0 0 0 0 0 1 1

Next the computer executes the `HALT` (000_2) instruction at memory location 3_{10} (00011_2). The "Stop" light comes on and program execution ceases. The computer's human operator may then read the final result of 6_{10} (00000110_2) from the accumulator.

ACCUMULATOR	0 0 0 0 0 1 1 0
PROG ADDR	0 0 0 1 1
OPERATION	0 0 0
OPERAND ADDR	0 0 0 0 1
OPERAND	0 1 1 0 0 1 0 0

An interesting characteristic of stored-program computers such as these is that the CPU typically makes no distinction whatsoever between a memory location containing data and one containing instructions. Here we saw memory location 3_{10} (00011_2) being used both as a data value (1_{10}) and as a computer instruction (`HALT`). This interesting characteristic actually allowed early programmers to write programs that could re-write themselves when executed! This practice is now discouraged because it makes the diagnosis and repair of program problems almost impossible; in fact, many CPUs today require that memory areas be declared as "data" or "instruction" areas to prevent instructions from being overwritten or data from being executed!