Assignment

Day 9

Monday, 13 September 2023

By: Ibrahim Tarek

SV G2 / Intake #3

Table of Contents

1. Executive Summary

This report discusses each of the following topics:

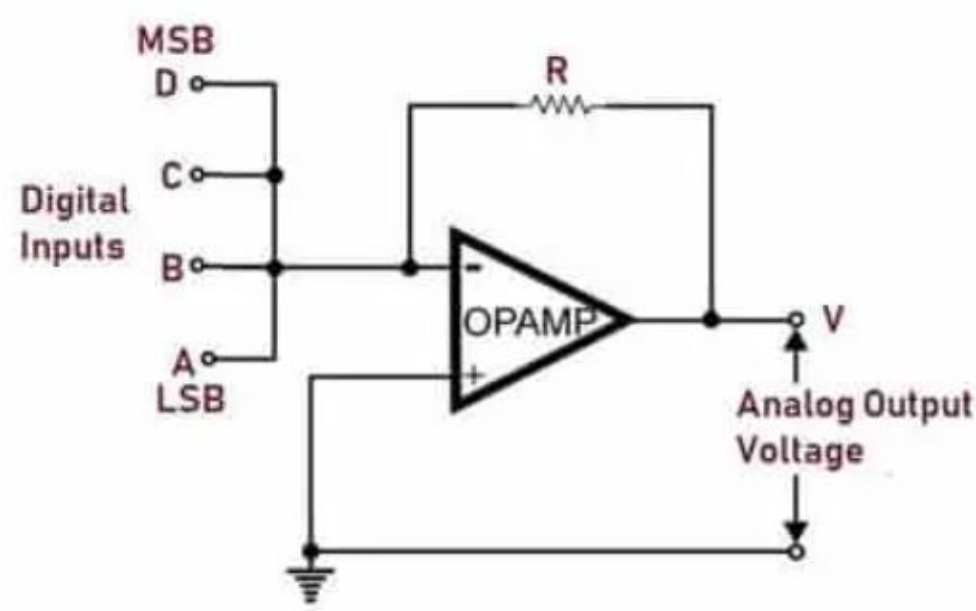
1- Types of DACs.
2- Memory layout of C program.

2. Types of DACs

A Digital to Analog Converter (DAC) consists of a number of binary inputs and a single output. In general, the number of binary inputs of a DAC will be a power of two.

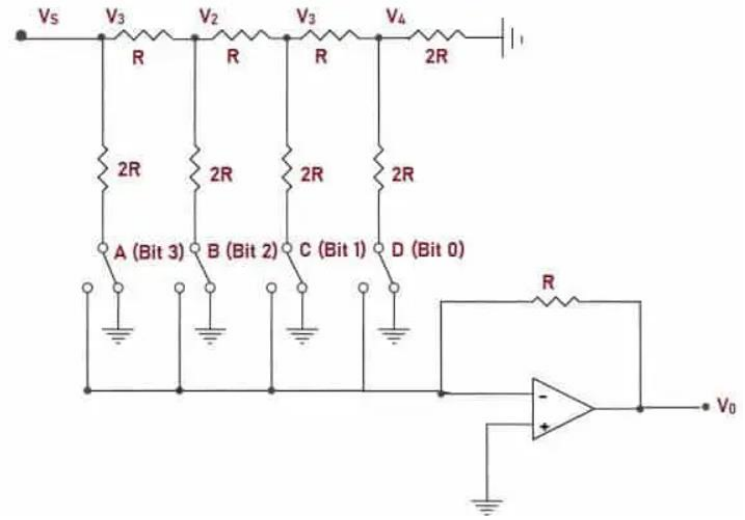
2.1. Weighted Resistor DAC

In this type of Converter, every digital input bit that needs to be converted, requires one resistor or current source. These resistors are connected across the inputs and the summing point. The output is generated through this Summing Amplifier Circuit. Fig. 3 below shows a typical Binary Weighted Resistor Converter Circuit which consists of an Op-Amp, four resistors which are connected at the input terminal of Op-amp along with the Feedback Resistor.



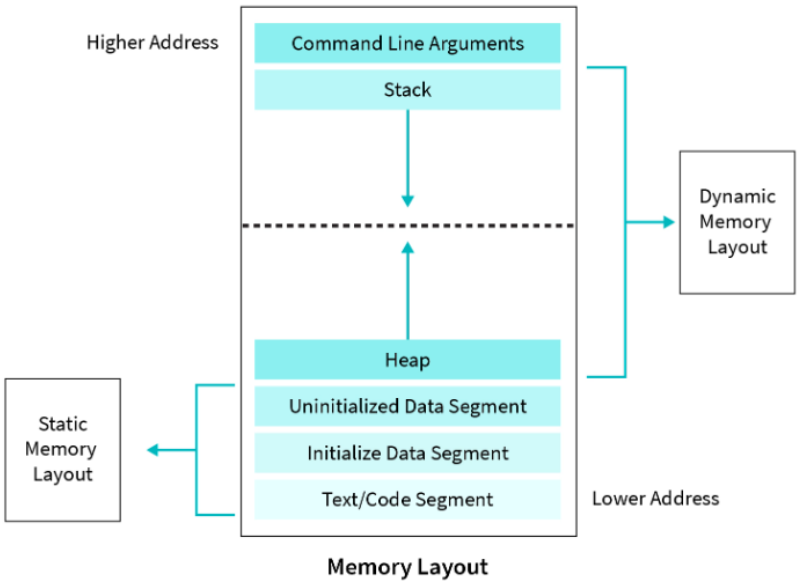
2.2. R-2R Ladder DAC

This type of Converter has only two values of Resistors, R and 2R. The conversion speed reduces in this type of DAC due to parasitic capacitance. It is the simplest type of DAC where the switch between ground and inverting input of the Op-amp is controlled by the input bit.



3. Memory layout of C program

When we execute a C program, the executable code of the file loads into RAM in an organized manner. Computers do not access program instructions directly from secondary storage because the access time of secondary storage is longer when compared to that of RAM. RAM is faster than secondary storage but has a limited storage capacity, so it is necessary for programmers to utilize this limited storage efficiently. Knowledge of memory layout in C is helpful to programmers because they can decide the amount of memory utilized by the program for its execution.



Memory Layout

3.1. Text Segment

After we compile the program, a binary file generates, which is used to execute our program by loading it into RAM. This binary file contains instructions, and these instructions get stored in the text segment of the memory.

Text segment has read-only permission that prevents the program from accidental modifications.

Text segment in RAM is shareable so that a single copy is required in the memory for frequent applications like a text editor, shells, etc.

3.2. Initialized data segment

Initialized data segment or data segment is part of the computer's virtual memory space of a C program that contains values of all external, global, static, and constant variables whose values are initialized at the time of variable declaration in the program. Because the values of variables can change during program execution, this memory segment has read-write permission. We can further classify the data segment into the read-

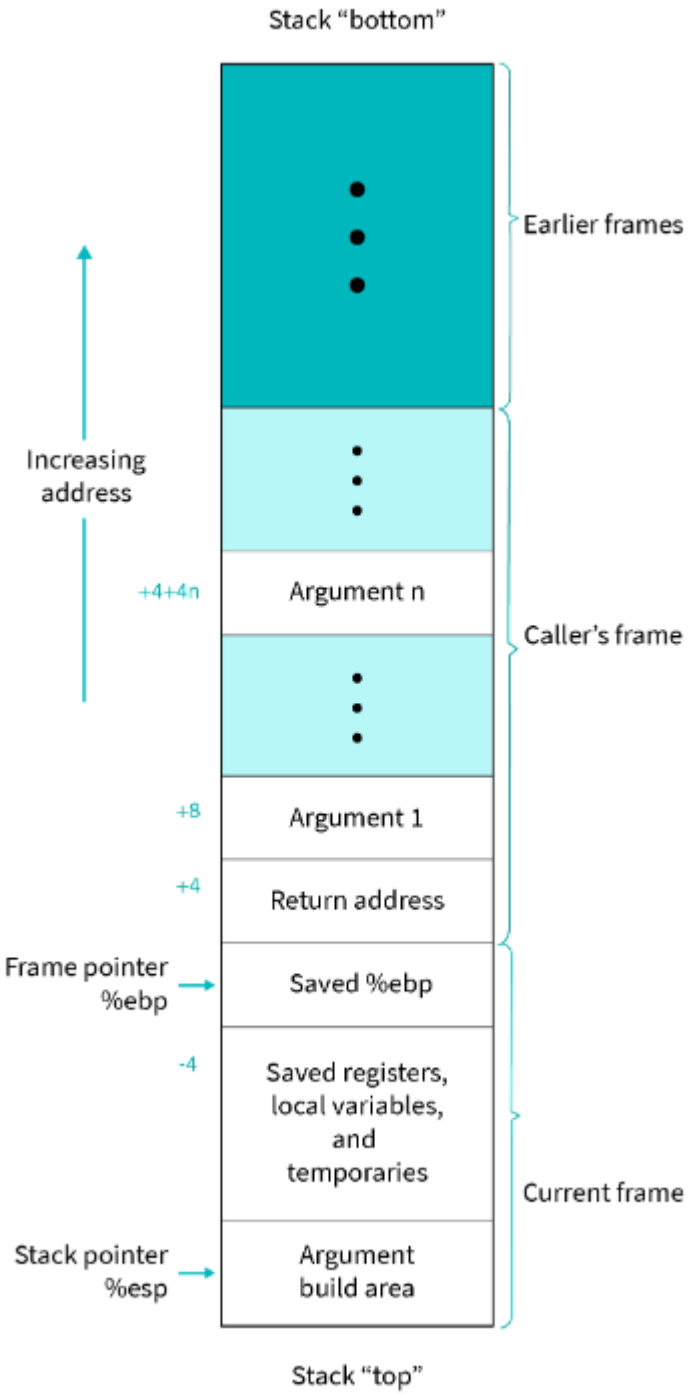
write and read-only areas. const variable comes under the read-only area. The remaining types of variables come in the read-write area.

3.3. Uninitialized data segment

An uninitialized data segment is also known as bss (block started by symbol). The program loaded allocates memory for this segment when it loads. Every data in bss is initialized to arithmetic 0 and pointers to null pointer by the kernel before the C program executes. BSS also contains all the static and global variables, initialized with arithmetic 0. Because values of variables stored in bss can be changed, this data segment has read-write permissions.

3.4. Stack

The stack segment follows the LIFO (Last In First Out) structure and grows down to the lower address, but it depends on computer architecture. Stack grows in the direction opposite to heap. Stack segment stores the value of local variables and values of parameters passed to a function along with some additional information like the instruction's return address, which is to be executed after a function call.



Stack pointer register keeps track of the top of the stack and its value change when push/pop actions are performed on the segment. The values are passed to stack when a function is called stack frame. Stack frame stores the value of function temporary variables and some automatic variables that store extra information like the return address and details of the caller's environment (memory registers). Each time function calls itself recursively, a new stack frame is created, which allows a set of variables of one stack frame to not interfere with other variables of a different instance of the function. This is how recursive functions work.

3.5. Heap

Heap is used for memory which is allocated during the run time (dynamically allocated memory). Heap generally begins at the end of bss segment and, they grow and shrink in the opposite direction of the Stack. Commands like malloc, calloc, free, realloc, etc are used to manage allocations in the heap segment which internally use sbrk and brk system calls to change memory allocation within the heap segment. Heap data segment is shared among modules loading dynamically and all the shared libraries in a process.

3.6. Command Line Arguements

When a program executes with arguments passed from the console like argv and argc and other environment variables, the value of these variables gets stored in this memory layout in C.

4. References

4.1. https://electricalfundablog.com/digital-to-analog-converter-dac/

4.2. https://www.scaler.com/topics/c/memory-layout-in-c/