PSI3441 – Arquitetura de Sistemas Embarcados

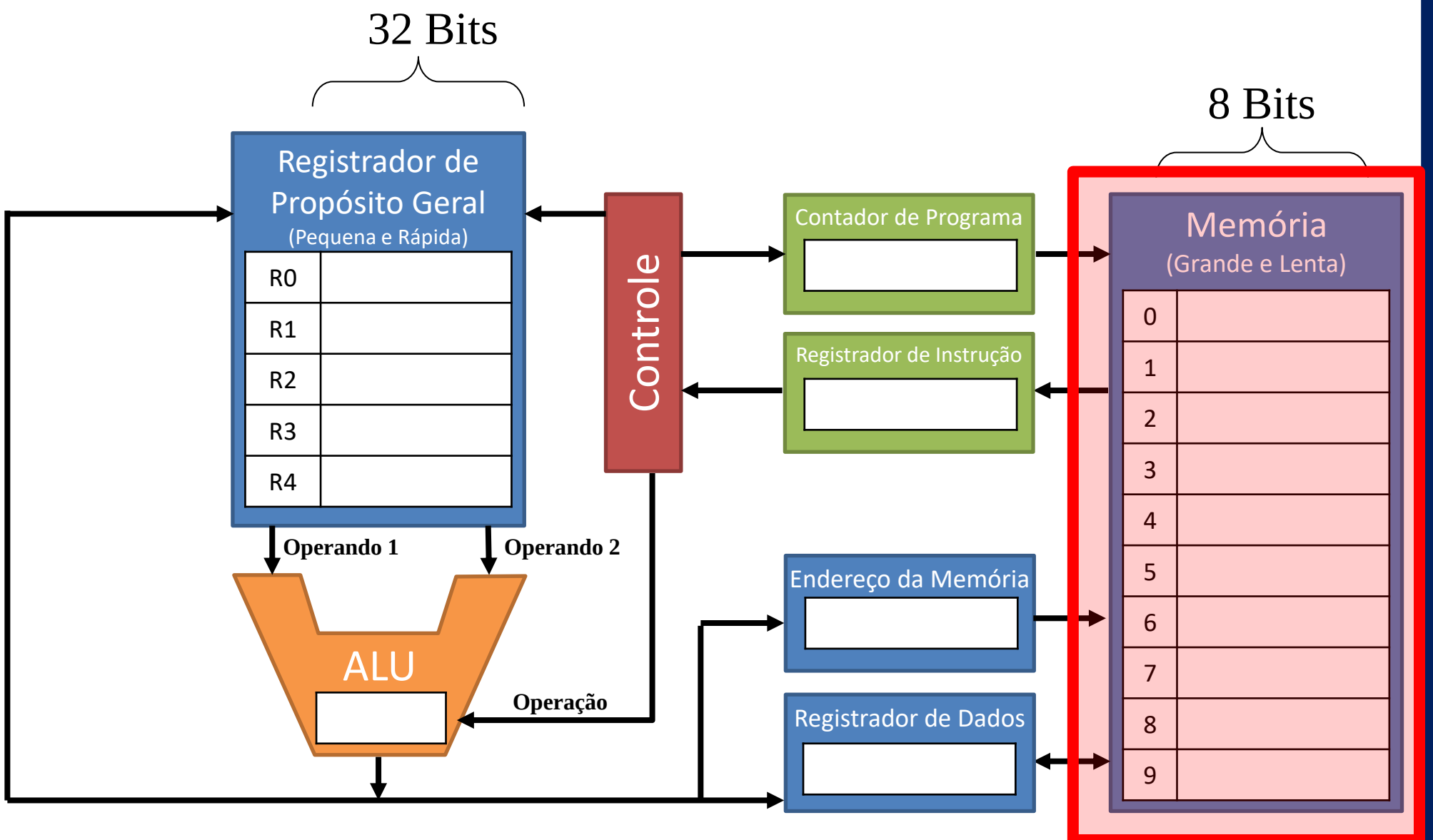
- Memória

Escola Politécnica da Universidade de São Paulo

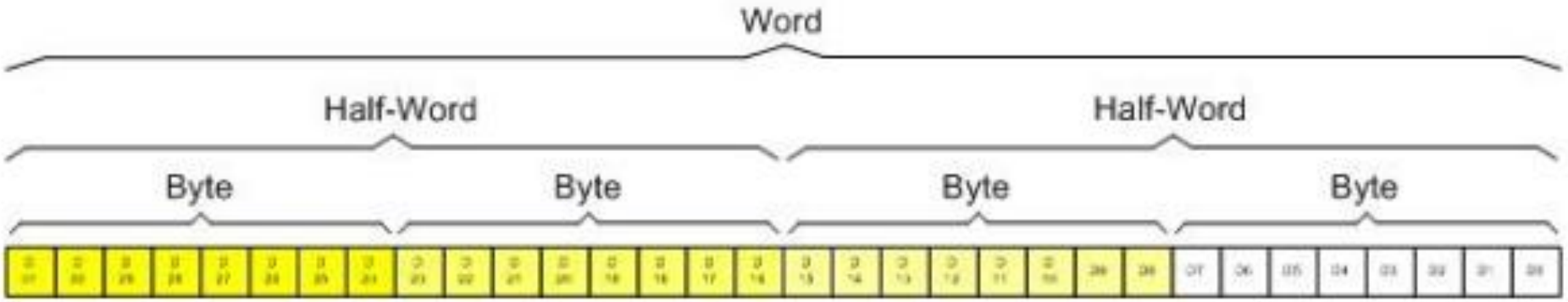
Prof. Gustavo Rehder – grehder@lme.usp.br

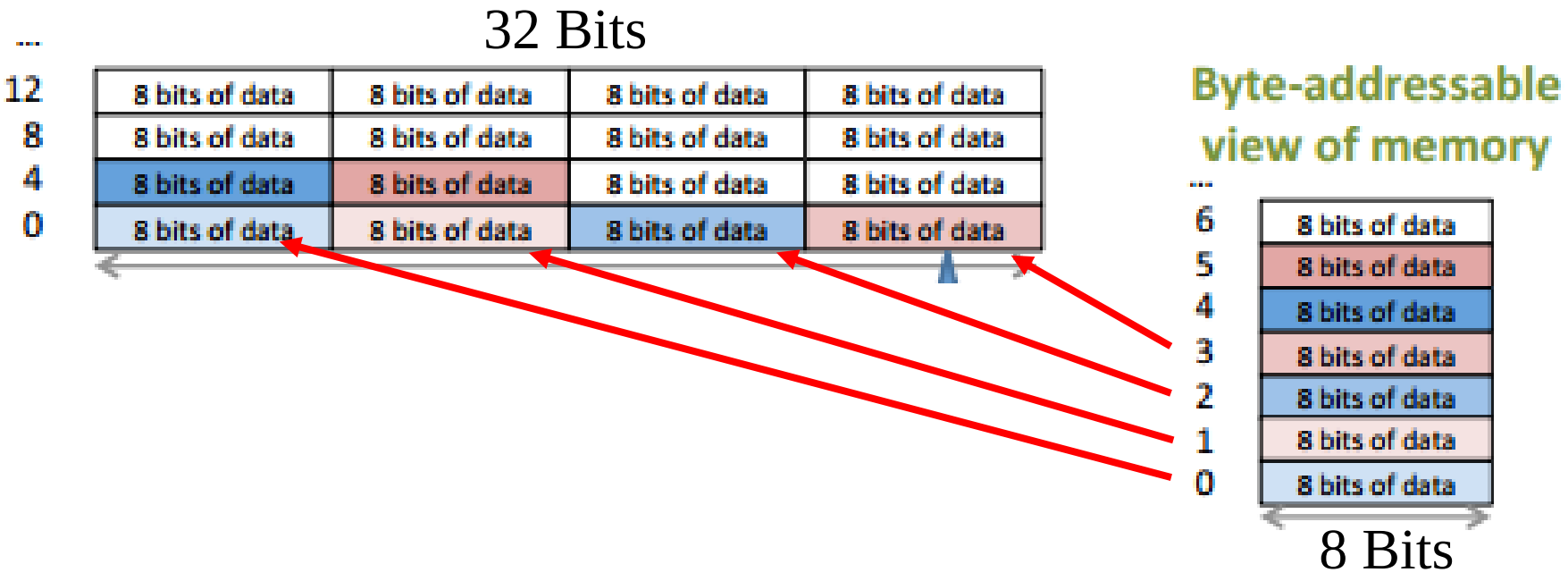
2020

Memória


32 Bits
8 Bits
Registrador de Propósito Geral
(Pequena e Rápida)
R0
R1
R2
R3
R4
Controle
Contador de Programa
Registrador de Instrução
Memória
(Grande e Lenta)
0
1
2
3
4
5
6
7
8
9
Operando 1
Operando 2
ALU
Operação
Endereço da Memória
Registrador de Dados

Memória no ARM Cortex M0+





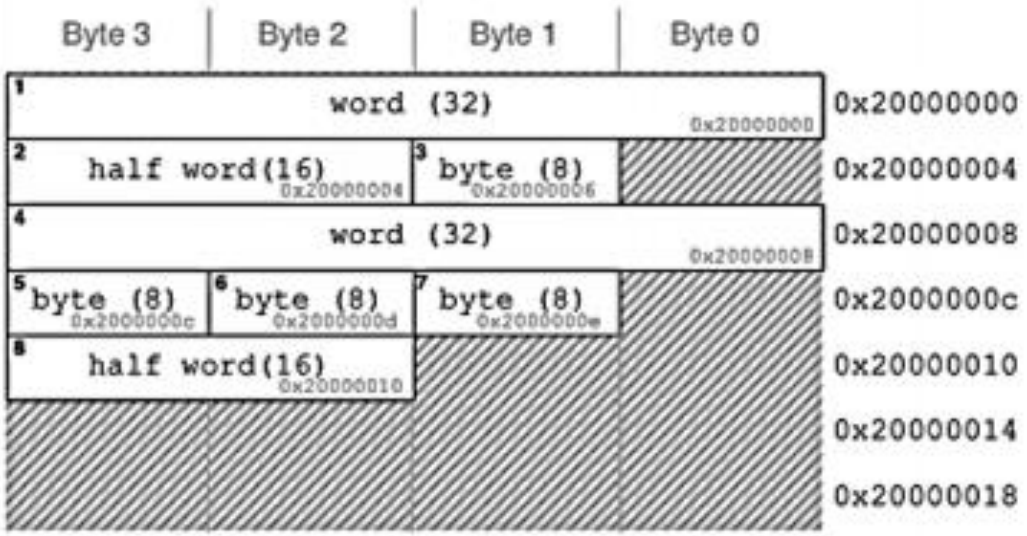
Tipo de dado e alocação na memória

Data type	Size	Range
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
short int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295
long long	8 bytes	-9,223,372,036,854,775,808 9,223,372,036,854,775,807
unsigned long long	8 bytes	0 to 18,446,744,073,709,551

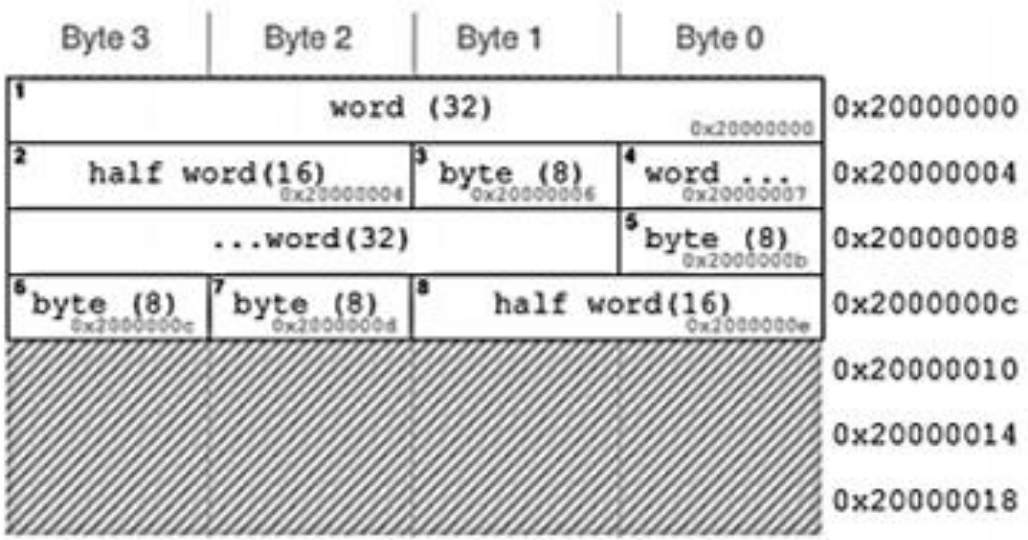
ANSI C (ISO C89)

ISO C99

Data type	Size	Range
int8_t	1 byte	-128 to 127
uint8_t	1 byte	0 to 255
int16_t	2 bytes	-32,768 to 32,767
uint16_t	2 bytes	0 to 65,535
int32_t	4 bytes	-2,147,483,648 to 2,147,483,647
uint32_t	4 bytes	0 to 4,294,967,295
int64_t	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
uint64_t	8 bytes	0 to 18,446,744,073,709,551,615



Aligned access

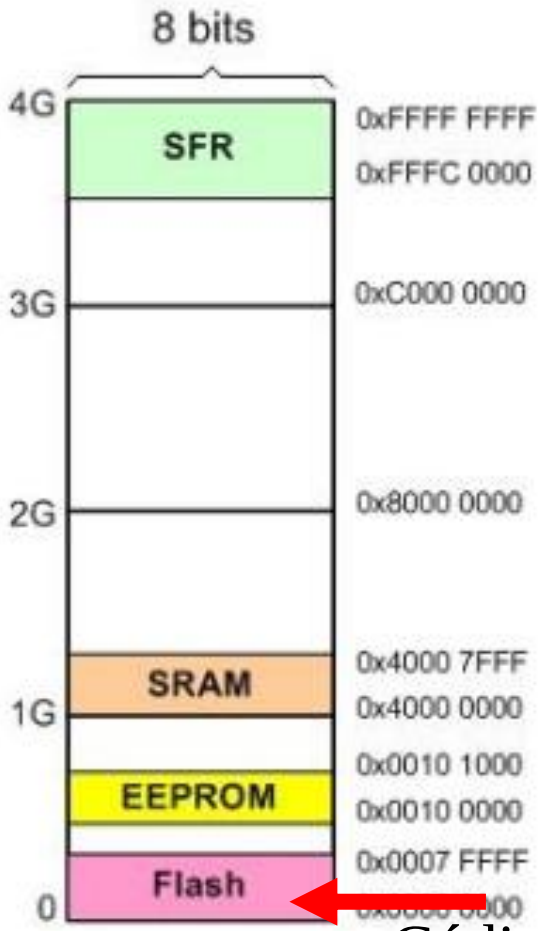


Unaligned access

Onde os dados são armazenados?

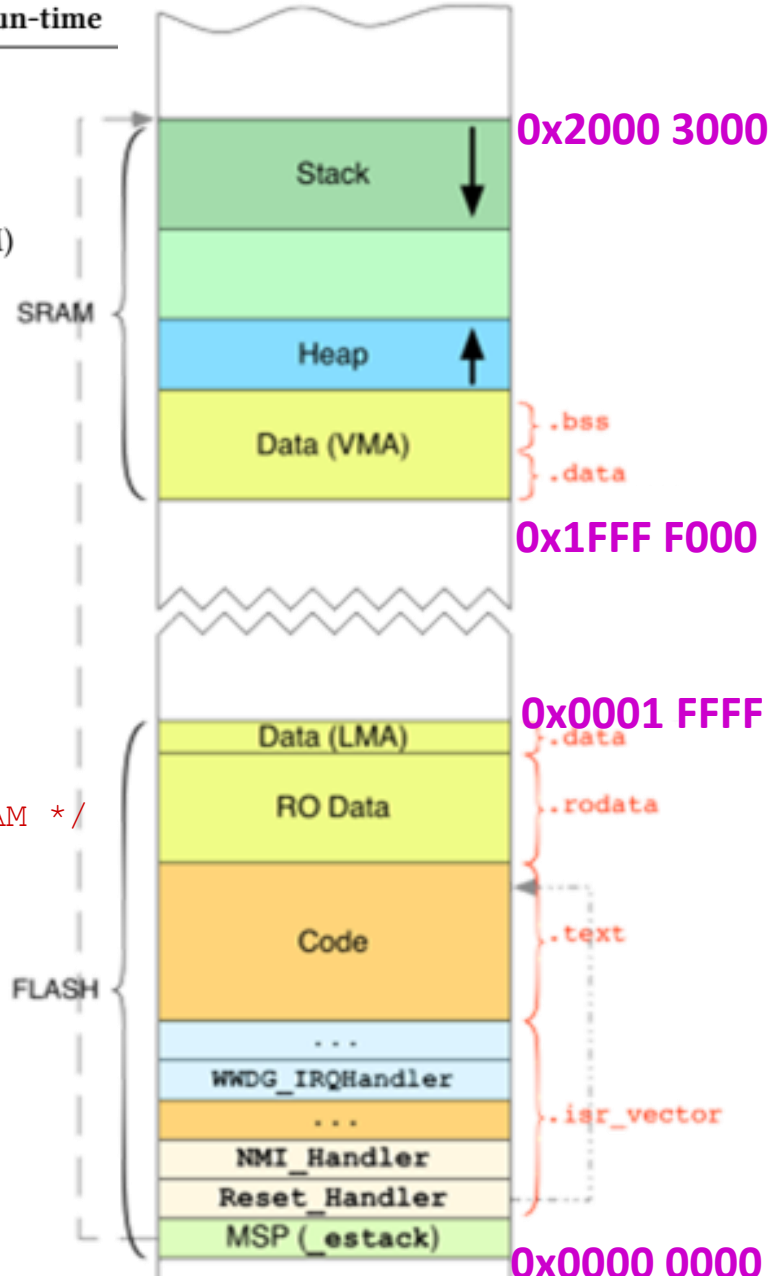
	Allocated size	Allocated address
Flash	128KB	0x00000000 to 0x0001FFFF
SRAM	16KB	0x1FFFF000 to 0x20002FFF
I/O	All the peripherals	0x400FF000 to 0x400FFFFF

Type of memory	Number of possible write/erase cycles	Write time per memory cell	Typical cell size with 0.8-μm technology
Volatile memory			
RAM	unlimited	≈ 70 ns	≅ 1700 μm²
Non-volatile memory			
EEPROM	100,000–1,000,000	3–10 ms	≈400 μm²
EPROM	1 (not UV-erasable)	≈50 ms	≈200 μm²
Flash EEPROM	≈10, 000	≈10 μs	≈200 μm²
FRAM	$\approx 10^{10}$	≈100 ns	≈200 μm²
PROM	1	≈100 ms	—
ROM	0	—	≈100 μm²



Onde os dados são armazenados?

Language structure	Binary file section	Memory region at run-time
Global un-initialized variables	`.common`	Data (SRAM)
Global initialized variables	`.data`	Data (SRAM+Flash)
Global `static` un-initialized variables	`.bss`	Data (SRAM)
Global `static` initialized variables	`.data`	Data (SRAM+Flash)
Local variables	<no specific section>	Stack or Heap (SRAM)
Local `static` un-initialized variables	`.bss`	Data (SRAM)
Local `static` initialized variables	`.data`	Data (SRAM+Flash)
Const data types	`.rodata`	Code (Flash)
Const strings	`.rodata.1`	Code (Flash)
Routines	`.text`	Code (Flash)



Definição do Stack e Heap no CodeWarrior

```
/* Highest address of the user mode stack */
_estack = 0x20003000; /* end of m_data */
__SP_INIT = _estack;

/* Generate a link error if heap and stack don't fit into RAM */
__heap_size = 0x00;   /* required amount of heap  */
__stack_size = 0x0400;    /* required amount of stack */
```

Qual o valor do registrador SP ao ligar o µControlador?

Exemplo do ARM Cortex M0+ no CodeWarrior

main.c | *main.c

```
#include "derivative.h"
#include <stdlib.h>  // deve-se incluir esta biblioteca para usar malloc()

int a=5; // Variável Global - alocada em .data - SRAM
const int table[8] = {5,0,1,5,6,7,9,10};  //Array de variáveis alocada em .rodata - Flash

int main(void){
    int g=0; //Variável local - alocada no Stack - SRAM
    static int i[10]={0}; //Array de variáveis locais alocadas em .data - SRAM
    int *p;
        p = malloc(3 * sizeof (int)); // Array de variáveis alocada no Heap - SRAM
```

Variables | Breakpoints | Registers | Memory | Modules

Name	Value	Location
(x)= g	2	0x20002ff4
> i	0x1ffff00c	0x1ffff00c
v p	0x1ffff074	0x20002ff0
(x)= *p	1	0x1ffff074
a	6	0x1ffff000
> table	0x00001bbc	0x00001bbc

Obs.: Mais informação sobre como é feita a divisão de memória no arquivo .map da pasta FLASH no CW

O que é o Stack?

- LIFO – Last-in First-out
- Armazenagem temporária
  - Dados locais
  - Endereços de retorno de funções
  - Passar parâmetros entre funções

Dados Existem só enquanto a função está sendo executada

- Tamanho dos dados é pré-determinado
- Acesso rápido
- Gerenciamento de memória automático
- Instruções Push e Pop

O que acontece?

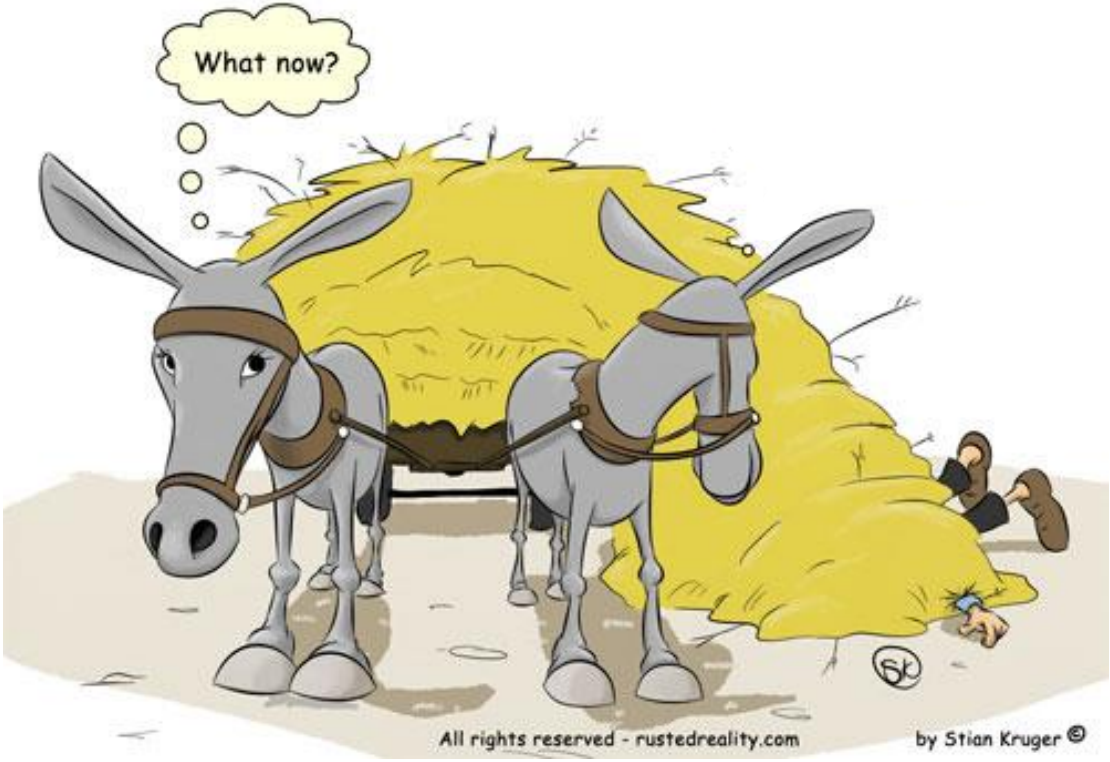
Declaração de vetores ou matrizes muito grandes

```
char hugeArray[10000000000000000];
```

Ou

Operações Recursivas

```
int foo(int x)
{
   if (x <= 0) return x;
      return foo(x - 1);
}
```



Para que usar o Heap?

- Armazenagem de dados
  - Variáveis globais
- Não é gerenciada automaticamente
- Alocação de memória é feita utilizando malloc() ou calloc()
- Memória alocada pode ser redimensionada utilizando realloc()
- É necessário desalocar a memória usando free()
- Memória pode se tornar fragmentada