2: Cortex-M Overview Stack & Exception Handling

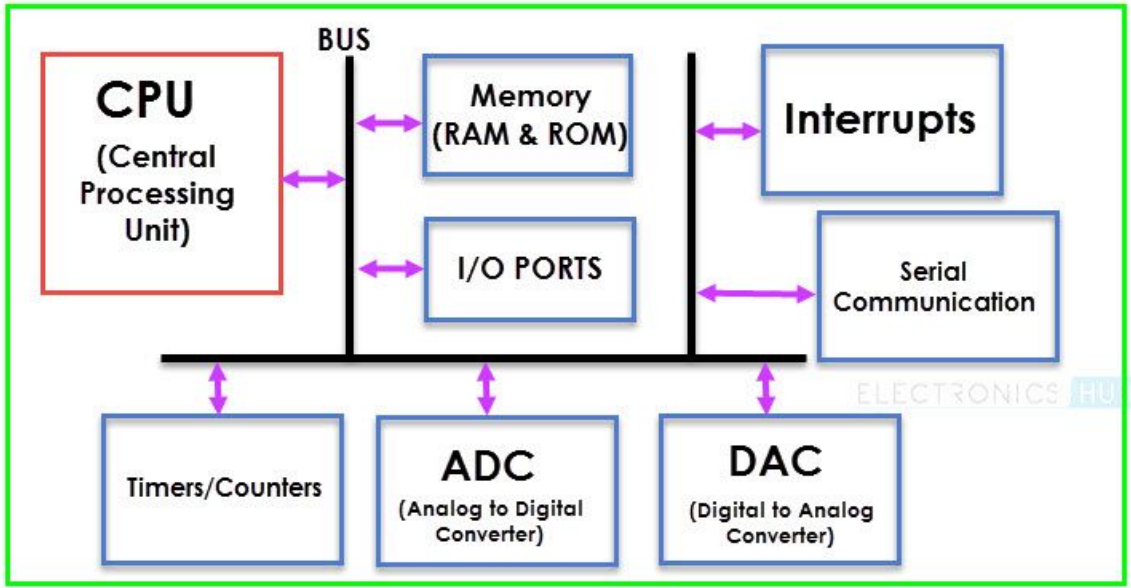
Phan Minh Thong

Cortex-M Overview

Cấu trúc vi điều khiển STM32 cũng như các vi điều khiển cơ bản gồm:

- 1 bộ xử lý trung tâm- CPU
- bộ nhớ(RAM, ROM)
- các ngoại vi (Uart, SPI, …)
- các I/O Ports.
- Một số bộ điều khiển ngắt.

Tất cả được kết nối qua các Bus.

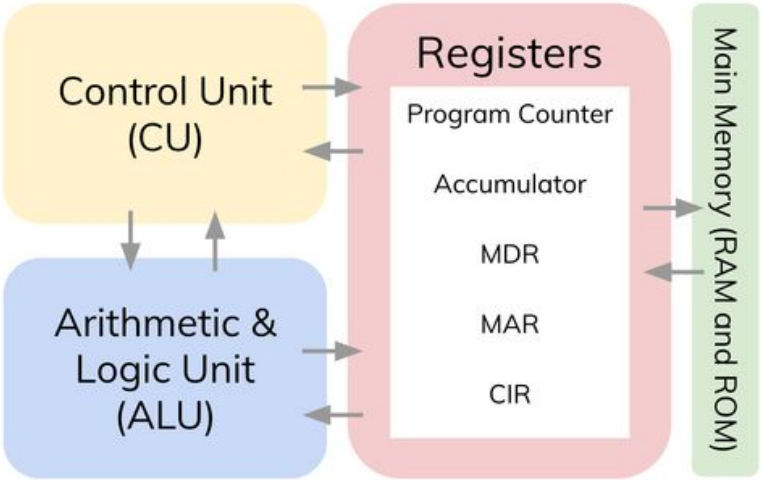


CPU

CPU là trái tim của vi điều khiển, nhiệm vụ là tìm và nạp lệnh, giải mã và thực hiện lệnh.

Cấu trúc chung của 1 CPU gồm:

- CU: Điều khiển hoạt động của bộ xử lý.
- ALU: Thực hiện tính toán số học và Logic theo lệnh của CU.
- ALU thực hiện tính toán trên bộ thanh ghi(Register Bank)

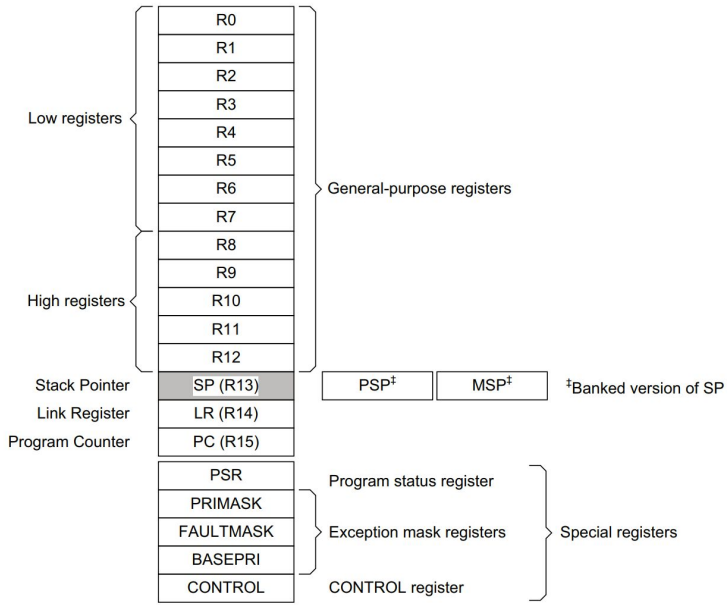


Đối với Cortex-M, bộ thanh ghi gồm 16 thanh ghi, R0-R15. Ngoài ra còn có các thanh đặc biệt

- R0-R12 là 13 thanh ghi đa dụng, dùng để tính toán.
- R14: Link Register, lưu vị trí trả về của 1 hàm.
- R15: PC, chứa địa chỉ lệnh tiếp theo sẽ được thực thi.
- R13: Stack Pointer, trỏ đến đỉnh Stack. Có 2 Stack Pointer. MSP và PSP.

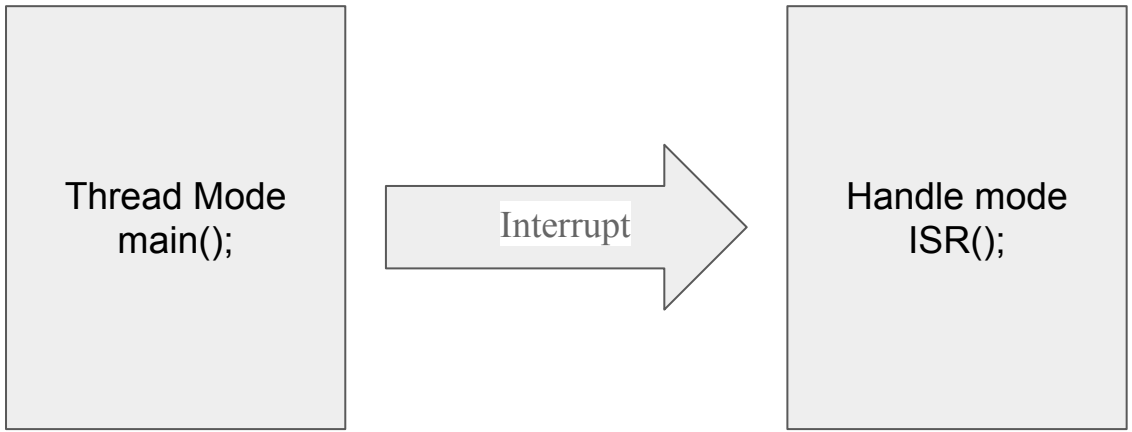
2.1.3 Core registers

The processor core registers are:



Vi xử lý ARM Cortex-Mx hoạt động ở 2 chế độ:

1. *Thread mode:* Tất cả các code chương trình viết trong main() sẽ được thực thi dưới chế độ Thread mode của vi xử lý. Mặc định, vi xử lý luôn bắt đầu ở chế độ Thread mode.
2. *Handler mode:* Tất cả các xử lý ngoại lệ (System Exception) và ngắt (Interrupt) sẽ được thực thi trong Handler mode của vi xử lý. Khi vi xử lý gặp System Exception hoặc Interrupt, chuyển từ Thread mode thành Handler mode và trình phục vụ tương ứng sẽ được thực thi ở Handle mode.



MSP & PSP

Cortex-M có 2 SP.

Main Stack Pointer (MSP). Là SP mặc định, được sử dụng ở cả Thread mode và Handle mode. **MSP** được sử dụng trong chương trình chính và cả xử lý ngắt.

Process Stack Pointer (PSP). Là 1 SP thay thế, nó **chỉ hoạt động** ở Thread mode. PSP thường được sử dụng cho hoạt động của các Task trong hệ thống.

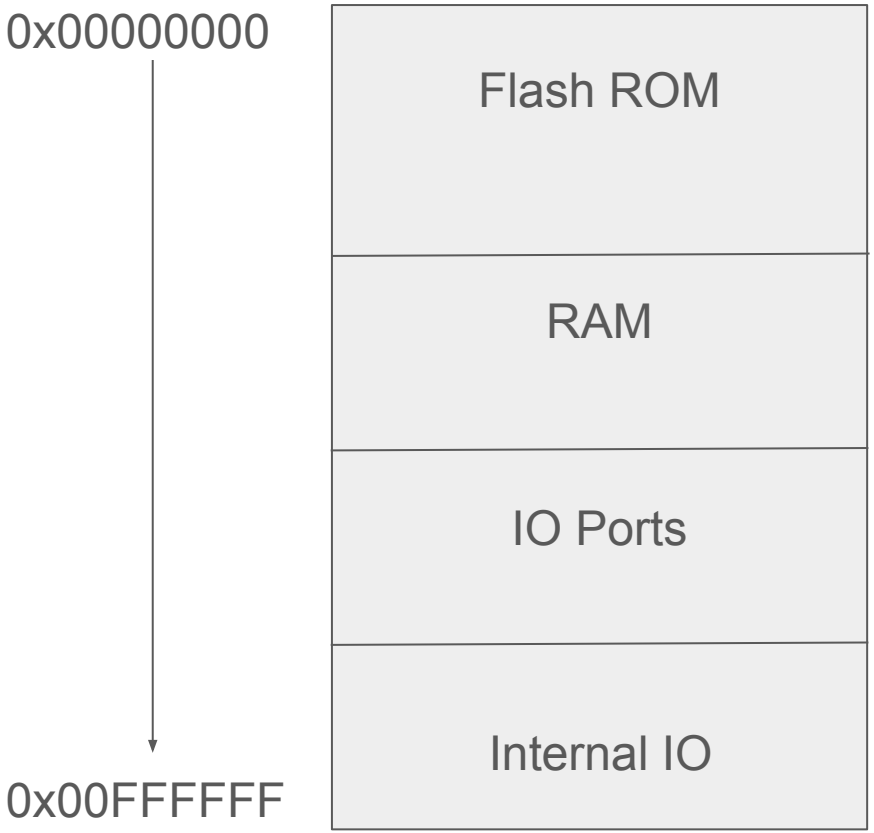
Trong Thread mode, bit[1] của thanh ghi `CONTROL` quy định SP nào sẽ được sử dụng:

Stack

Cortex-M Memory Map

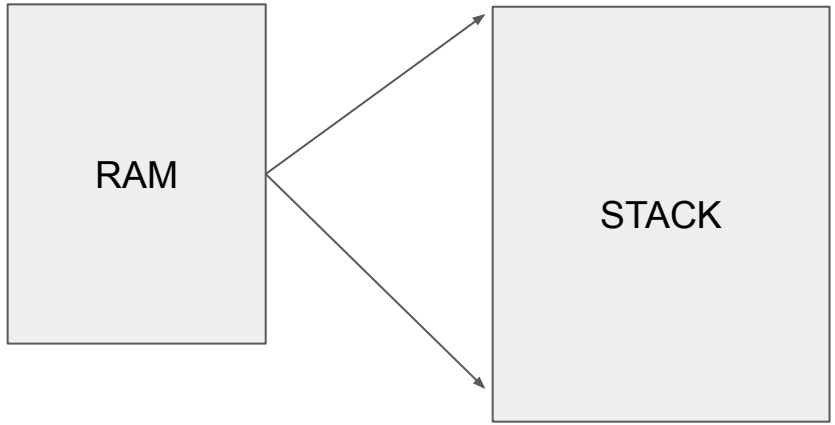
Các vi điều khiển Cortex-M đều được tổ chức bộ nhớ giống nhau, chỉ khác ở kích thước từng vùng.

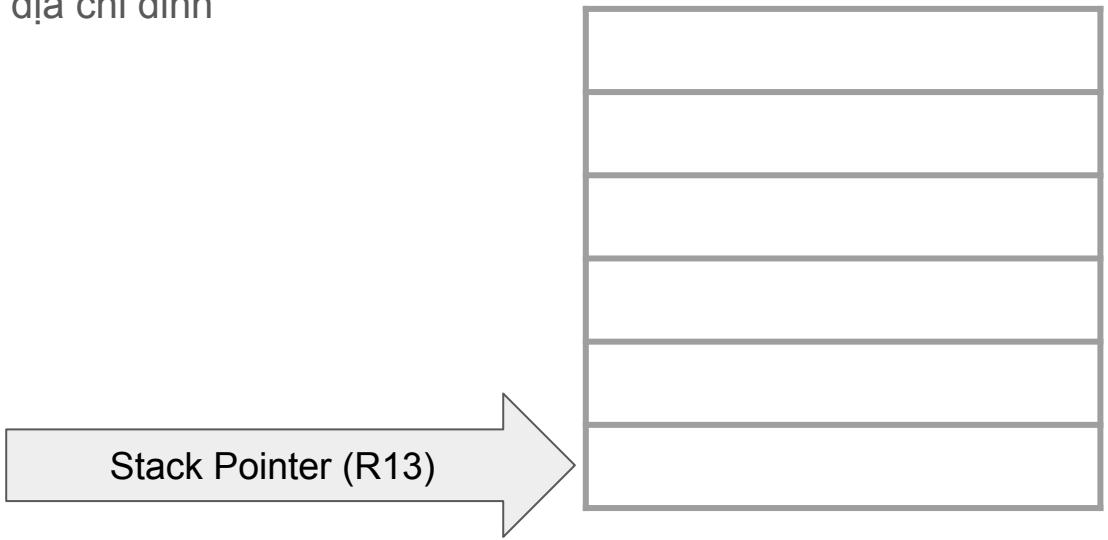
Vùng nhớ bắt đầu ở địa chỉ 0x00000000. địa chỉ sẽ tăng theo chiều hướng xuống.



Stack

Stack được cấp phát trong RAM, là 1 vùng nhớ hoạt động theo kiểu LIFO.
Vì kiến trúc là 32bit, nên Stack sẽ hoạt động theo kiểu 32bit, mỗi ô nhớ cách nhau 4byte địa chỉ.
R13, Stack Pointer luôn trỏ tới địa chỉ đỉnh Stack.





Stack Manipulation

2 thao tác chính trên Stack là PUSH & POP.

PUSH: Đưa dữ liệu vào Stack.
POP: Lấy dữ liệu ra khỏi Stack.

Stack Manipulation

Cortex_M sử dụng **full descending stack.**
Tức là đáy của Stack sẽ là địa chỉ lớn nhất.

Khi push data vào Stack, SP **sẽ giảm 4** và data sẽ được thêm vào vị trí đó.

Ví dụ: Push(R2); (R2 = 0x09)

SP = SP -4 ;

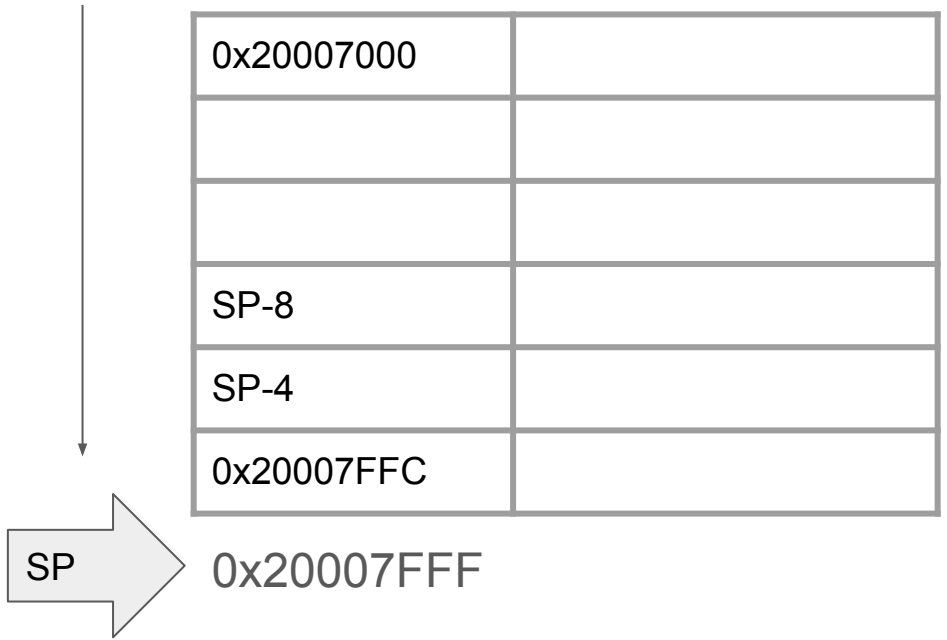
*(SP) = R2.

Khi pop dữ liệu, data tại SP sẽ được lấy ra khỏi Stack, sau đó SP sẽ **tăng lên 4** để trỏ đến đỉnh Stack (Mới).

Ví dụ: Pop(R0): (*(SP) = 0x08)

*(R0) = *(SP) = 0x08;

SP = SP +4;

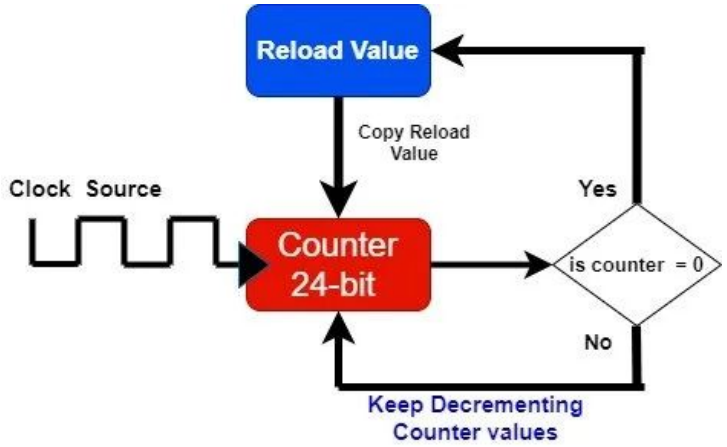


SysTick

SysTick là 1 timer của hệ thống, không giống như các ngoại vi Timer, Systick Timer là một Timer độc lập nằm trong lõi Cortex.

SysTick là Timer **24-bit**, **đếm ngược**, xảy ra ngắt khi giá trị đếm bằng 0, và **tự nạp lại** giá trị đếm ban đầu.

Trong RTOS, SysTick thường dùng để tạo tín hiệu đồng bộ các Task.



Một số thanh ghi sau để cấu hình và điều khiển Systick Timer.

SYST_CSR - Systick Control & Status Register: cấu hình điều khiển và trạng thái.

- **Bit[16] - COUNTFLAG**: Cờ này được tự động set lên 1 khi bộ đếm tràn (đếm về 0).
- **Bit[2] - CLKSOURCE**: Chọn nguồn cấp xung clock cho timer.

  0 = External Clock = AHB / 8.

  1 = Processor Clock = AHB.
- **Bit[1] - TICKINT**: Cho phép ngắt Systick nếu bit này bằng 1.
- **Bit[0] - ENABLE**: Cho phép bộ đếm hoạt động.

  0 = DISABLE.

  1 = ENABLE.

  Khi được ENABLE, bộ đệm sẽ lấy giá trị đã cài đặt sẵn trong thanh ghi Reload và bắt đầu đếm ngược.

Một số thanh ghi sau để cấu hình và điều khiển Systick Timer.

SYST_RVR - Systick Reload Value Register :

- Thanh ghi này chưa giá trị nạp lại để bắt đầu đếm.
- Vì SysTick là timer 24 bit, nên chỉ sử dụng 24 bit thấp
- Giá trị từ 0x0000.0001 - 0x00FF.FFFF.

SYST_CVR - Systick Current Value Register

- Đây là thanh ghi chứa giá trị đếm của SysTick.
- Thanh ghi này sẽ bắt đầu với giá trị lấy từ thanh ghi SYST_RVR, và đếm ngược về 0.

Cấu hình SysTick

- Đặt giá trị đếm ban đầu - Reload Value bằng thanh ghi **SYST_RVR**.
- Xóa giá trị đếm hiện tại trên thanh ghi **SYST_CVR**.
- Cấu hình hoạt động cho Systick Timer bằng thanh ghi Control **SYST_CSR**.
    - Chọn nguồn cấp xung Clock cho Systick bằng **Bit[2] - CLKSOURCE**.
    - Cho phép ngắt Systick (nếu sử dụng ngắt) bằng **Bit[1] - TICKINT**.
    - Cho phép bộ đếm Systick hoạt động bằng **Bit[0] - ENABLE**.

Hàm phục vụ ngắt SysTick_Handler() sẽ tự động được gọi khi SysTick đếm về 0.

```
/*************Steps to reload SysTick:*/
/*1/Reload timer with number of cycles per millisecond.*/
SysTick->LOAD = 72000-1;
/*2/Clear Current Value Reg.*/
SysTick->VAL = 0;
/*3/Select Internal Clock Source.*/
SysTick->CTRL |= 1<<2;
/*4/Enable Interupt.*/
SysTick->CTRL |= 1<<1;
/*5/Enable Systick.*/
SysTick->CTRL |= 1<<0;
```

Delay()

Ứng dụng cơ bản nhất của SysTick là tạo delay.

```
#define MAX_DELAY
0xFFFFFFFF
__IO uint32_t g_curr_tick;
__IO uint32_t g_curr_tick_p;
__IO uint32_t tick_freq =1;

void Tick_Increment(void){
        g_curr_tick += tick_freq;
}

void SysTick_Handler(){
        Tick_Increment();
}
```

```
uint32_t get_tick(void){
        __disable_irq();
        g_curr_tick_p = g_curr_tick;
        __enable_irq();
        return g_curr_tick_p;
}


void delay(uint32_t delay){
        uint32_t tick_start = get_tick();
        uint32_t wait = delay*1000;
        if(wait <MAX_DELAY){
                wait += (uint32_t)(tick_freq);
        }

        while((get_tick()- tick_start) < wait){}
}
```

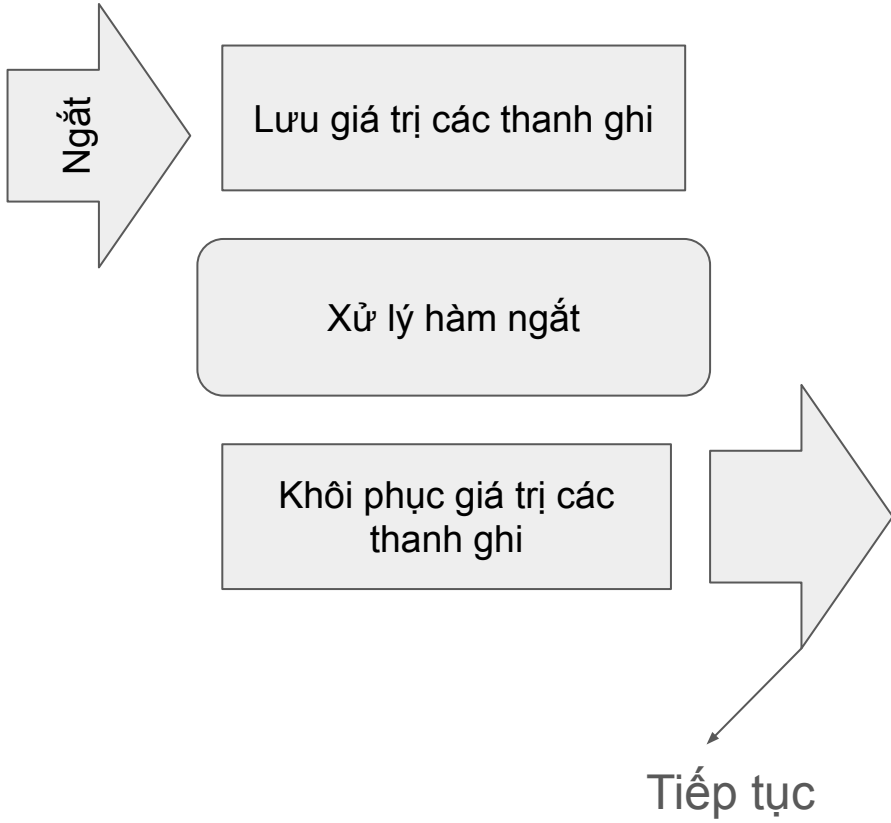
Exception Handling

Với Cortex-M, khi 1 exception xảy ra (khi có sự kiện Ngắt):

- Giá trị trạng thái chương trình được lưu lại.
- Bộ xử lý chuyển đến chương trình phục vụ xử lý exception.
- Sau khi xử lý exception, các giá trị trạng thái sẽ được khôi phục.
- Chương trình tiếp tục tại nơi mà nó đã tạm dừng để xử lý exception.

➢ Việc lưu lại giá trị trạng thái chương trình là việc lưu các giá trị thanh ghi trong CPU.

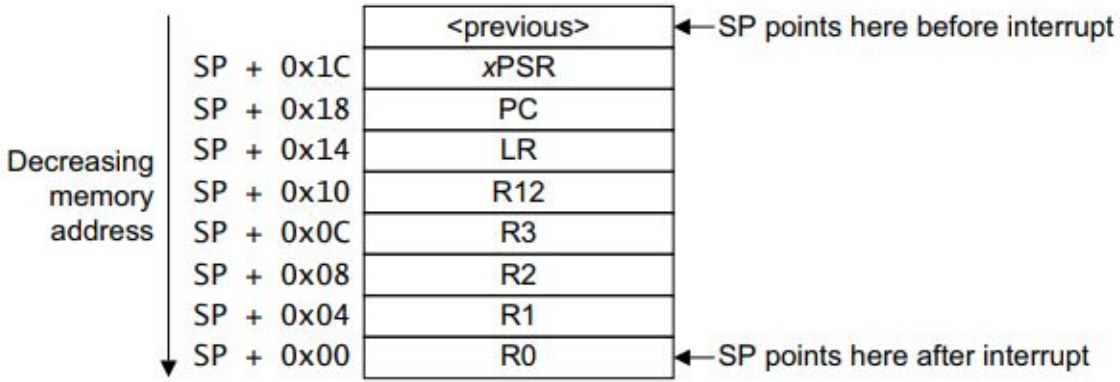
➢ Khôi phục giá trị tương ứng với việc khôi phục lại giá trị cho từng thanh ghi tương ứng.



Stack frame

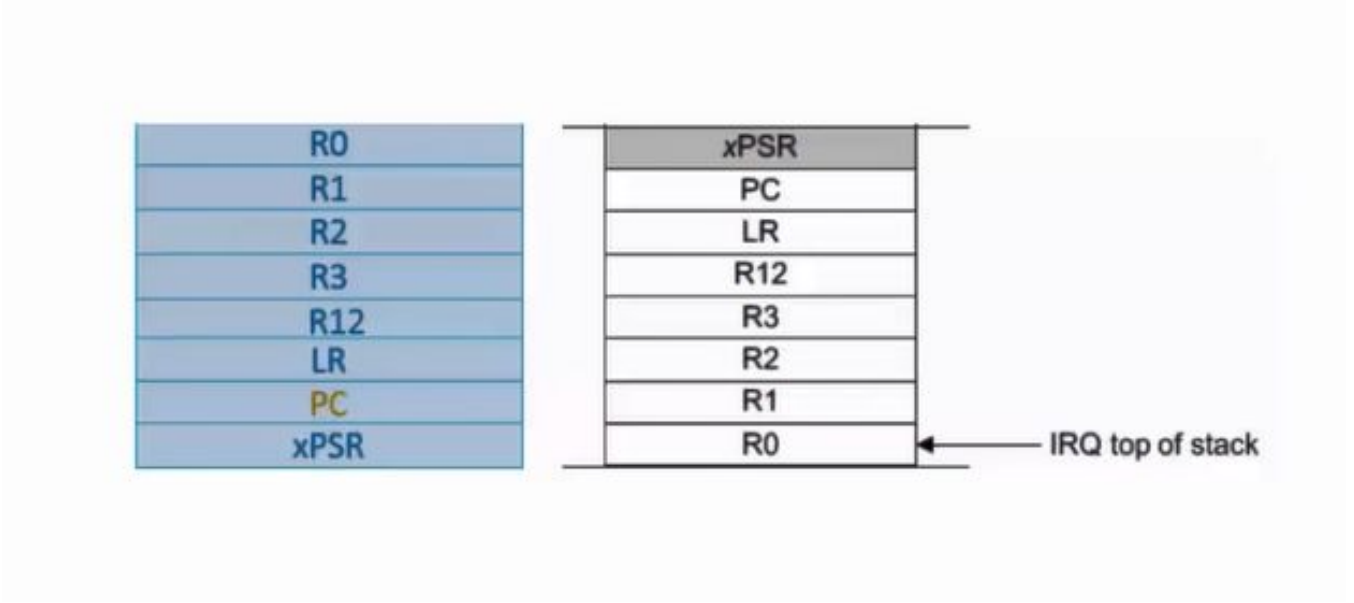
Các thanh ghi sẽ được bộ xử lý push vào Stack khi exception xảy ra. Bao gồm 8 thanh ghi sắp xếp theo cấu trúc, gọi là Stack Frame, chứa các thông tin:

- xPSR: Chứa thông tin hiện tại của chương trình.
- R0, R1, R2, R3, R12.
- LR: Chứa địa chỉ lệnh tiếp theo khi hàm hiện tại trả về.
- PC: Chứa địa chỉ lệnh tiếp theo sau khi hoàn tất exception. Có nghĩa là lệnh đáng lẽ sẽ được thực hiện nếu không có exception.



Khi Stack Frame lưu vào bộ nhớ và khôi phục hoàn toàn tự động, việc sắp xếp trong Stack như hình sau:

Điều này dẫn đến một khả năng: Nếu ta thay đổi thông tin của Stack Frame trong khi xử lý exception, bộ xử lý sẽ cập nhật dữ liệu mới này vào các thanh ghi sau khí thoát exception.



Manual Context Switching

Quá trình lưu trữ, thay đổi và khôi phục lại Stack Frame mới gọi là Context Switching.

Ứng dụng quá trình này, thay đổi giá trị PC, ta có thể thay đổi việc chạy giữa các task với nhau trong RTOS.

Từ đó có thể phát triển ứng dụng giúp chạy các task đồng bộ với nhau.

- Sử dụng SysTick để tạo ngắt định kì->bên trong hàm ngắt xử lý để chọn task thực thi->sau đó cập nhật.