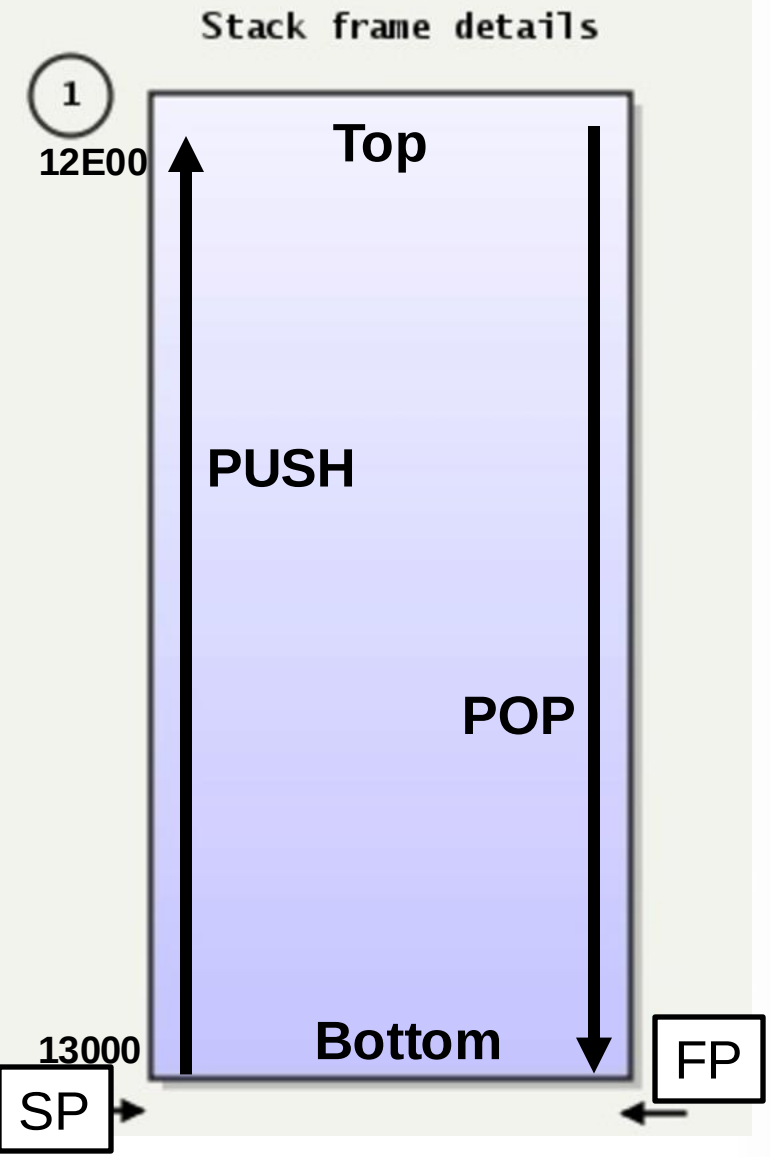
Class 5

CSC 590 ARM Architecture and Security
ARM Instruction Set (4):
Stack & Stack Frame & System Call

Dr. Si Chen (schen@wcupa.edu)

The Stack



Stack:

- A special region of your computer's memory that **stores temporary variables** created by each functions
- The stack is a "**LIFO**" (last in, first out) data structure
- Once a stack variable is freed, that region of memory becomes available for other stack variables.

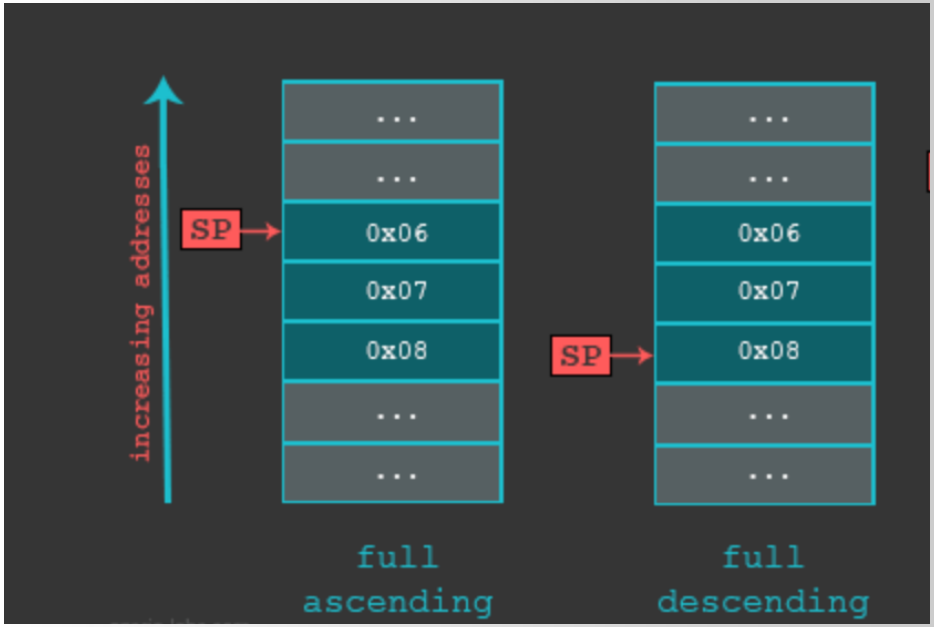
Properties:

- the stack grows and shrinks as functions **push and pop** local **variables**
- there is no need to manage the memory yourself, variables are allocated and freed **automatically**
- the **stack has size limits**
- stack variables only exist while the function that created them, is running

ARM	Description	x86
R0	General Purpose	EAX
R1-R5	General Purpose	EBX, ECX, EDX, ESI, EDI
R6-R10	General Purpose	–
R11 (FP)	Frame Pointer	EBP
R12	Intra Procedural Call	–
R13 (SP)	Stack Pointer	ESP
R14 (LR)	Link Register	–
R15 (PC)	<- Program Counter / Instruction Pointer ->	EIP
CPSR	Current Program State Register/Flags	EFLAGS

Stack and StackFrame

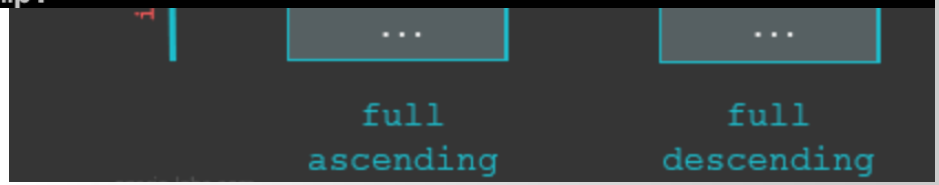
- The Stack is a memory region within the program/process. This part of the memory gets allocated when a process is created.
- We use Stack for storing temporary data such as local variables of some function, environment variables which helps us to transition between the functions, etc.
- We interact with the stack using **PUSH** and **POP** instructions.



Stack and StackFrame

- The Stack is a memory region within the program/process. This part of the memory gets allocated when a process is created.
- We use Stack for storing temporary data such as local variables of some function, en…en the functions, e…
- We interact…

```
gef➤  disas main
Dump of assembler code for function main:
   0x00010588 <+0>:     push    {r11, lr}
=> 0x0001058c <+4>:     add     r11, sp, #4
   0x00010590 <+8>:     sub     sp, sp, #8
   0x00010594 <+12>:    mov     r1, #4
   0x00010598 <+16>:    mov     r0, #3
   0x0001059c <+20>:    bl      0x10558 <add>
   0x000105a0 <+24>:    str     r0, [r11, #-8]
   0x000105a4 <+28>:    ldr     r1, [r11, #-8]
   0x000105a8 <+32>:    ldr     r0, [pc, #20]   @ 0x105c4 <main+60>
   0x000105ac <+36>:    bl      0x1147c <printf>
   0x000105b0 <+40>:    mov     r3, #0
   0x000105b4 <+44>:    mov     r0, r3
   0x000105b8 <+48>:    sub     sp, r11, #4
   0x000105bc <+52>:    pop     {r11, lr}
   0x000105c0 <+56>:    bx      lr
   0x000105c4 <+60>:    andeq   r5, r6, r0, asr #17
End of assembler dump.
```



Stack Example

```
.global main

main:
    mov   r0, #2  /* set up r0 */
    push  {r0}    /* save r0 onto the stack */
    mov   r0, #3  /* overwrite r0 */
    pop   {r0}    /* restore r0 to it's initial state */
    bx    lr      /* finish the program */
```

bx lr: This instruction will branch to the address contained in the link register (lr). This is commonly used to return from a function.

```
.global _start

.text

_start:
    @ Set up the stack frame for the main program
    push {fp, lr}         @ Push the frame pointer and link register onto the stack
    add fp, sp, #4        @ Set the frame pointer to the stack pointer + 4

    @ Prepare values for summation
    mov r0, #5            @ First number to sum
    mov r1, #7            @ Second number to sum

    @ Call the function
    bl my_function        @ Call function, result will be in r0

    @ The sum of 5 and 7 (12) is now in r0

    @ Restore the stack frame
    sub sp, fp, #4        @ Set the stack pointer to the frame pointer - 4
    pop {fp, lr}          @ Pop the frame pointer and link register from the stack

    @ Exit the program
    mov r7, #1            @ Exit syscall number
    swi 0                 @ Make the syscall to exit the program

my_function:
    @ Set up the stack frame for the function
    push {fp, lr}
    add fp, sp, #4

    @ Function body to sum two numbers
    add r0, r0, r1        @ Add r0 and r1, store result in r0

    @ Restore the stack frame
    sub sp, fp, #4
    pop {fp, lr}

    @ Return from the function
    bx lr
```

Stack Frame Example

Stack and Stack Frame

```c
#include <stdio.h>
#include <stdlib.h>

int my_function(int a, int b) {
    return a + b;
}

int main() {
    int x = 5, y = 7, sum;

    sum = my_function(x, y);

    printf("The sum is: %d\n", sum);

    return 0;
}
```

Stack and Stack Frame

```
        .type   my_function, %function
my_function:
        @ args = 0, pretend = 0, frame = 8
        @ frame_needed = 1, uses_anonymous_args = 0
        @ link register save eliminated.
        str     fp, [sp, #-4]!
        add     fp, sp, #0
        sub     sp, sp, #12
        str     r0, [fp, #-8]
        str     r1, [fp, #-12]
        ldr     r2, [fp, #-8]
        ldr     r3, [fp, #-12]
        add     r3, r2, r3
        mov     r0, r3
        add     sp, fp, #0
        @ sp needed
        ldr     fp, [sp], #4
        bx      lr
        .size   my_function, .-my_function
        .section        .rodata
        .align  2
.LC0:
        .ascii  "The sum is: %d\012\000"
        .text
        .align  2
        .global main
        .syntax unified
        .arm
        .type   main, %function
main:
        @ args = 0, pretend = 0, frame = 16
        @ frame_needed = 1, uses_anonymous_args = 0
        push    {fp, lr}
        add     fp, sp, #4
        sub     sp, sp, #16
        mov     r3, #5
        str     r3, [fp, #-8]
        mov     r3, #7
        str     r3, [fp, #-12]
        ldr     r1, [fp, #-12]
        ldr     r0, [fp, #-8]
        bl      my_function(PLT)
        str     r0, [fp, #-16]
        ldr     r1, [fp, #-16]
        ldr     r3, .L5
```


arm-linux-gnueabi-gcc -S -march=armv7-a your_program.c -o your_program.s

hello_world.s

```
1   .data                   /* section declaration */
2   hello_string:
3       .asciz "Hello, World!\n"  /* null-terminated string */
4
5   .text                   /* section declaration */
6   .global _start          /* entry point for the program */
7
8   _start:                 /* _start label */
9       /* Write syscall */
10      mov r0, #1          /* file descriptor (stdout) */
11      ldr r1, =hello_string /* pointer to the string */
12      mov r2, #14         /* string length */
13      mov r7, #4          /* syscall number for sys_write */
14      swi 0               /* invoke syscall */
15
16      /* Exit syscall */
17      mov r0, #0          /* exit status */
18      mov r7, #1          /* syscall number for sys_exit */
19      swi 0               /* invoke syscall */
```

hello_world.s

```
1   .data                   /* section declaration */
2   hello_string:
3       .asciz "Hello, World!\n"  /* null-terminated string */
4
5   .text                   /* section declaration */
6   .global _start          /* entry point for the program */
7
8   _start:                 /* _start label */
9       /* Write syscall */
10      mov r0, #1          /* file descriptor (stdout) */
11      ldr r1, =hello_string /* pointer to the string */
12      mov r2, #14         /* string length */
13      mov r7, #4          /* syscall number for sys_write */
14      swi 0               /* invoke syscall */
15
16      /* Exit syscall */
17      mov r0, #0          /* exit status */
18      mov r7, #1          /* syscall number for sys_exit */
19      swi 0               /* invoke syscall */
```

```
schen@molly:~$ arm-linux-gnueabi-as -o hello_world.o hello_world.s
schen@molly:~$ arm-linux-gnueabi-ld -o hello_world hello_world.o
schen@molly:~$ qemu-arm hello_world
Hello, World!
```

https://arm.syscall.sh/

Stdin, stdout, and stderr

Name	File descriptor	Abbreviation
Standard input	0	stdin
Standard output	1	stdout
Standard error	2	stderr

Q & A