

ARM Architecture Memory / Data Type Alignments

Complete the Stack memory layout and translate the following C code into ARM assembly

```
#include <stdio.h>
int main() {
    short a[3];
    short *ptr1;
    short tmp;
    short *ptr2;

    a[0] = -4;
    a[1] = 333;
    a[2] = 0;

    ptr1 = &a[1];
    ptr2 = &a[2];

    tmp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = tmp;

    tmp = a[0];

    printf("a[0]=%hd, a[1]=%hd, a[2]=%hd\n", a[0], a[1], a[2]);
}
```


high
mem

```
.cpu    cortex-a53
.syntax unified

.section .rodata
fmt:
.asciz "a[0]=%hd, a[1]=%hd, a[2]=%hd\n"

.global main
.text
.align 2

main:
.equ    FP_OFF, 4
.equ    A, _____
.equ    PTR1, _____
.equ    TMP, _____
.equ    PTR2, _____
.equ    PAD, _____
.equ    FRMADD, _____
push    {fp, lr}
add     fp, sp, _____
_____ , _____ , _____

@ a[0] = -4, a[1] = 333, a[2] = 0
mov     r3, _____
_____ r3, [fp, _____] @ a[0]
_____ r3, _____
_____ r3, [fp, _____] @ a[1]
mov     r3, _____
_____ r3, [fp, _____] @ a[2]
```

```
@ ptr1 = &a[1]
add     r3, _____, _____
_____ r3, [fp, _____]

@ ptr2 = &a[2]
add     r3, _____, _____
_____ r3, [fp, _____]

@ tmp = *ptr1
ldr     r3, [_____]
_____ r3, [_____]
strh    r3, [_____]

@ *ptr1 = *ptr2
ldr     r3, [_____] @ ptr2
_____ r3, [_____]
ldr     r2, [_____] @ ptr1
_____ r3, [_____]

@ *ptr2 = tmp
ldrsh   r3, [_____] @ tmp
ldr     r2, [_____] @ ptr2
strh    _____, [_____]

@ tmp = a[0]
ldrsh   r3, [_____]
strh    r3, [fp, _____]

@ print values
_____ , _____
ldrsh   _____, [_____]
```

```

ldrsh    _____, [_____]          bx lr
ldrsh    _____, [_____]          .end
bl        printf
sub      sp, fp, FP_OFFSET
pop      {fp, lr}

```

=====

Translate the following C code into Assembly. Assume all variables and parameters are stored on the stack.

```

#include <stdio.h>
#include <stdlib.h>
#define SIZE 4
int a = 20;
unsigned short b = 10;
short c;

int dosth(int a0, int a1, int a2, int a3, short a4, unsigned short a5){
    return a0 + a1 + a2 + a3 + a4 + a5;
}

int
main(int argc, char* argv[]){
    int arr[SIZE];
    FILE *f = fopen(argv[1], "r");
    if (f==NULL){
        perror("error open file");
        return -1;
    }
    for(int i = 0; i < SIZE; ++i){
        fscanf(f, "%d", &arr[i]);
        printf("%d", arr[i]);
    }
    a = dosth(arr[0], arr[1], arr[2], arr[3], c, b);
    printf("%d", a);
    return 0;
}

```

```

.cpu cortex-a53
.syntax unified
.arch armv6
.extern fopen
.extern perror
.extern fscanf
.extern printf
//create a and b

```

```

//create c

```

```

.section .rodata
.Lfmt1:    .string "r"
.Lfmt2:    .string "error open file"
.Lfmt3:    .string "%d"

```

```

.section .text
.align 2
.global main
.type main, %function
.equ SIZE, _____
.equ FP_OFF, _____ //need r4
.equ ARR, _____
.equ F, _____
.equ I, _____
.equ PAD, _____
.equ OARG6, _____
.equ OARG5, _____
.equ FRMADD, _____

```

```

main: #create stack frame
push {_____ fp, lr}
add fp, sp, FP_OFF
_____

```

```

# FILE *f = fopen(argv[1], "r");

```

```

//if (f == NULL){.....}

```

```

//start for loop

```

```

//use r4 as a backup for &arr[i]
add r4, fp, -ARR

```

```

.Lfor:

```

```

.Lendfor:

```

```

//call a = dosth(arr[0], arr[1], arr[2],
arr[3], c, b)

```

```

.Lendif:

```

```

//call printf("%d", a);
mov r1, r0
ldr r0, =.Lfmt3
bl printf
b .LreturnS

```

```

.LreturnF:

```

```

    mov r0, -1
    b .Ldone

```

```

.LreturnS:

```

```

    mov r0, 0

```

```

.Ldone:

```

```

    sub sp, fp, FP_OFF
    pop {r4, r5, fp, lr}
    #go back to the caller
    bx lr

```

```

.size main, (. - main)

```

```

.section .text

```

```

.align 2

```

```

.global dosth

```

```

.type dosth, %function

```

```

.equ FP_OFF, _____

```

```

//no preserved is used

```

```

.equ ARG5, _____

```

```

.equ ARG6, _____

```

```

dosth:

```

```

//return

```

```

sub sp, fp, FP_OFF

```

```

pop {fp, lr}

```

```

bx lr

```

```

.size dosth, (. - dosth)

```

```

.section .note.GNU-stack, "", %progbits

```

```

.end

```