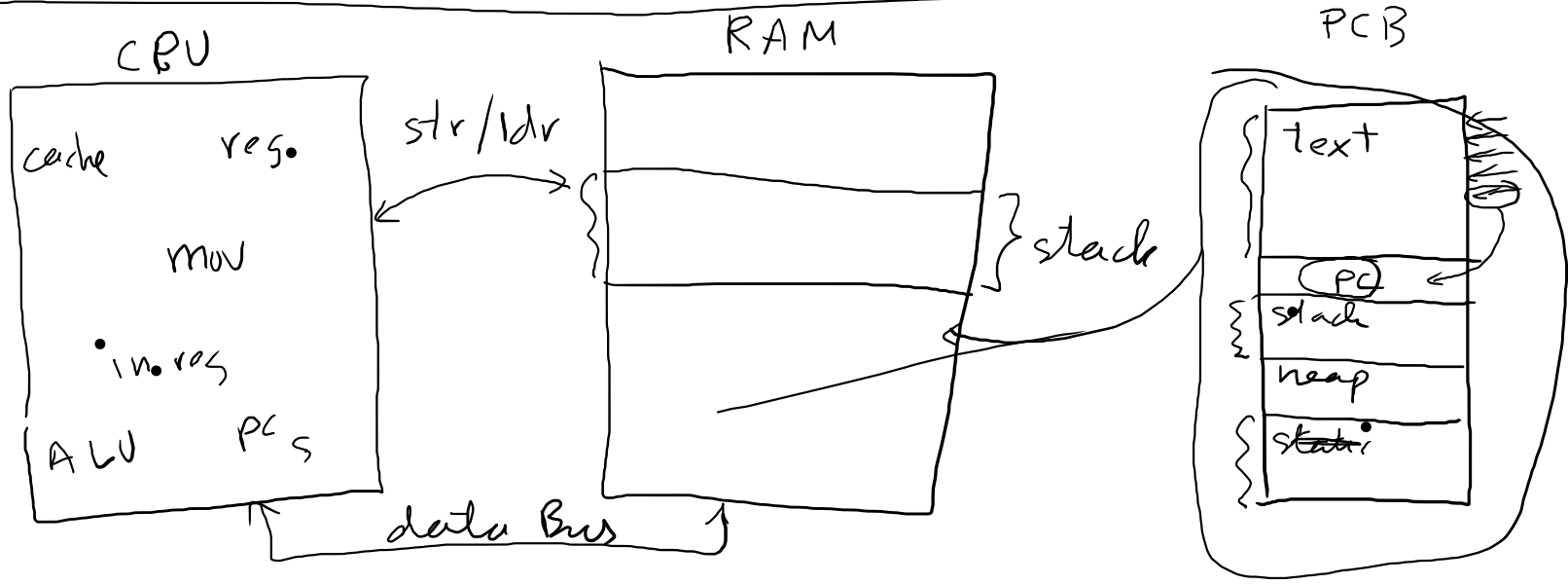
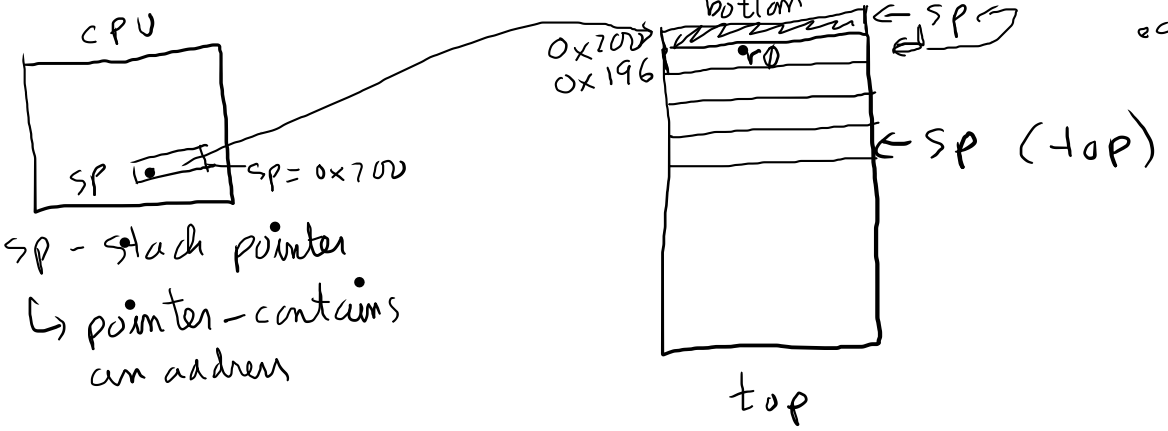
September 13, 2021

- user input
- loops
- if-then
- stacks
- functions

(2)

(1)





align (2)

$2^{(2)} = 4$

ldr dest, source

- push
- pop

push {r0} ≡

① sub sp, sp, #4 @ sp = sp - 4

② str r0, [sp]

[ ] ≡ de-reference op

de-reference op

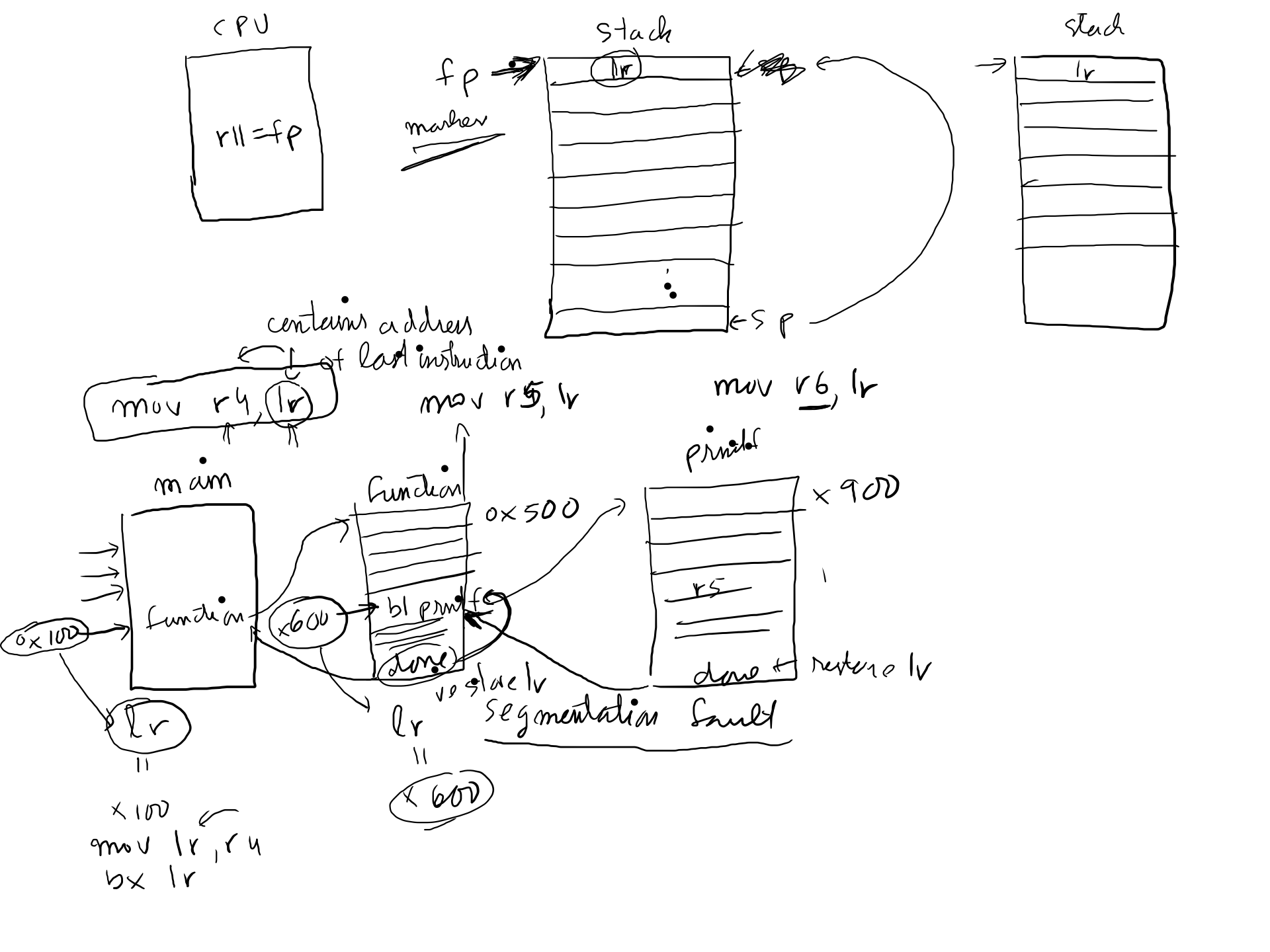
pop {r1}

[0x196] = r0

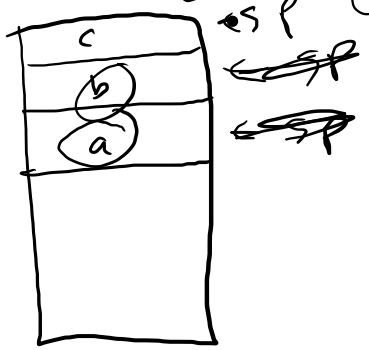
in C.

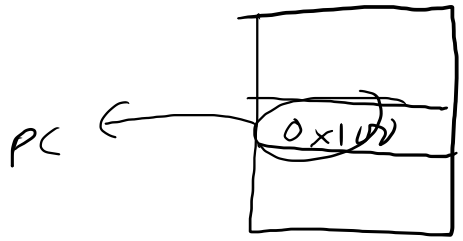
① ldr r1, [sp] *s;

≡ ② add sp, sp, #4 *s = 1


CPU
r11=fp
Stack
fp
lr
marker
sp
Stack
lr
contains address of last instruction
mov r4, lr
mov r5, lr
mov r6, lr
main
function
0x100
x100
mov lr, r4
bx lr
lr
function
0x500
x600
bl printf
done
lr
x600
printf
x900
r5
done
return lr
Segmentation fault

pop {fp, pc}





fp ← a

pc ← b

pop {pc} ≡→ { pop {lr}
bx lr

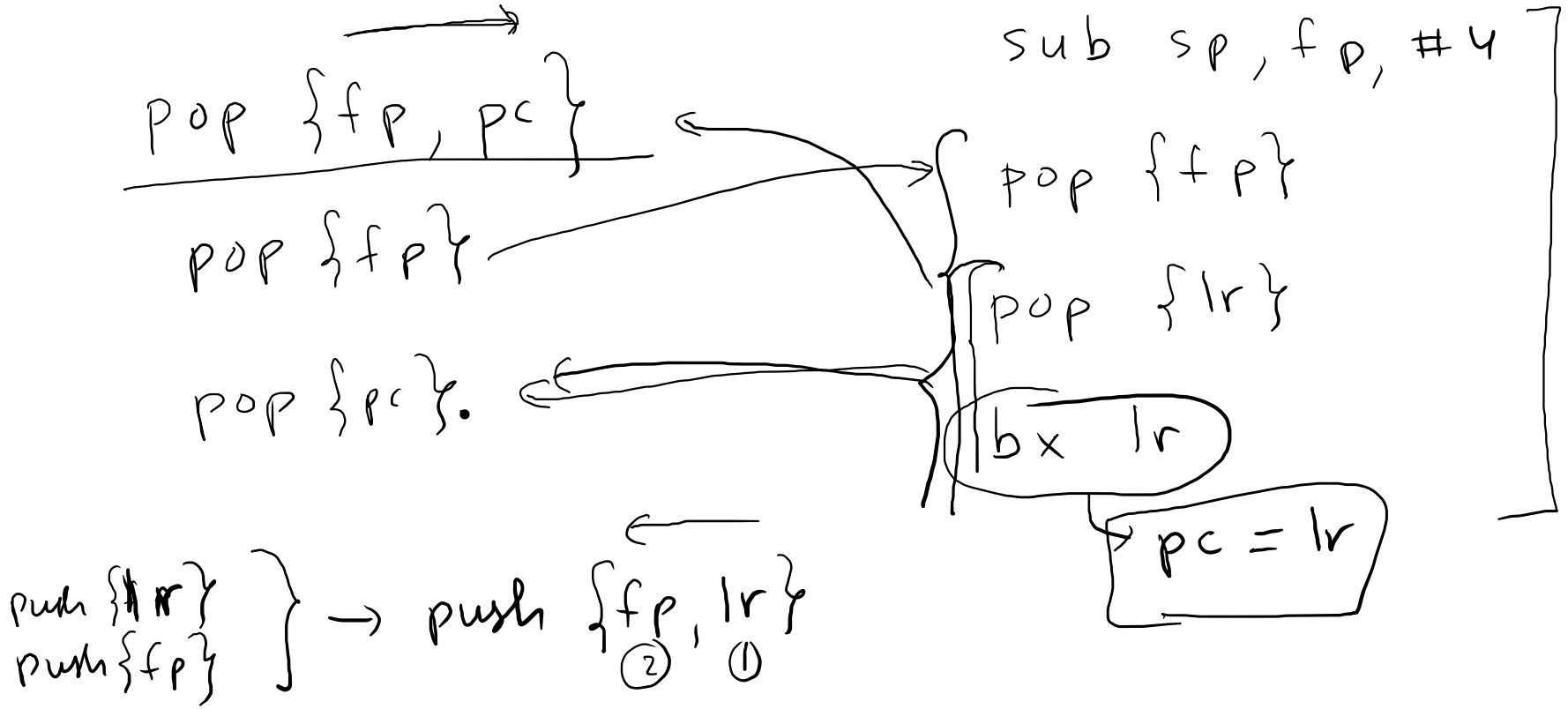
r11

```
main:
    push {lr}
    push {fp}
    add fp, sp, #4 @ fp = sp+4
0x700 → bl test
```

lr = 0x200

```
    sub sp, fp, #4
    pop {fp}
    pop {lr}
    bx lr
```


```
test:
    push {lr}
    push {fp}
    add fp, sp, #4
    ⋮
    sub sp, fp, #4 @ sp = fp-4
    pop {fp}
    pop {lr}
    bx lr
```



---

test:

→ push {fp, lr}

add fp, sp, #4

⋮

sub sp, fp, #4

→ pop {fp, pc}

begin

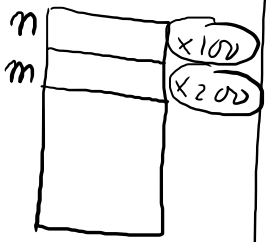
end

Ex: write a prog. that asks user for two integers
print out sum

```
#include <stdio.h>
void get_input(int *, int *);

int main()
{  int n, m;
   int sum;
   get_input(&n, &m);

   sum = n + m;
   printf("sum = %d\n", sum);
   return 0;
}
```




```
void get_input(int *n, int *m)
{
   scanf("%d %d", n, m);
}
```

main.s

```
.cpu    cortex-a53
.fpu    neon-fp-armv8

.data
outp:   .asciz  "sum = %d \n"  @ printf("sum = %d\n", sum)

.text
.align 2
.global main
.type main, %function

main:
    push {fp, lr}
    add fp, sp, #4
```

@ int n, m ; ← allocate memory on stack
r0 r1

@ get_inp (&n, &m);
× 100

sub sp, sp, #4 @ sp = sp - 4 → store n

mov r0, sp @ r0 = 0x100

sub sp, sp, #4 @ sp = sp - 4 → store m

mov r1, sp @ r1 = 0x96

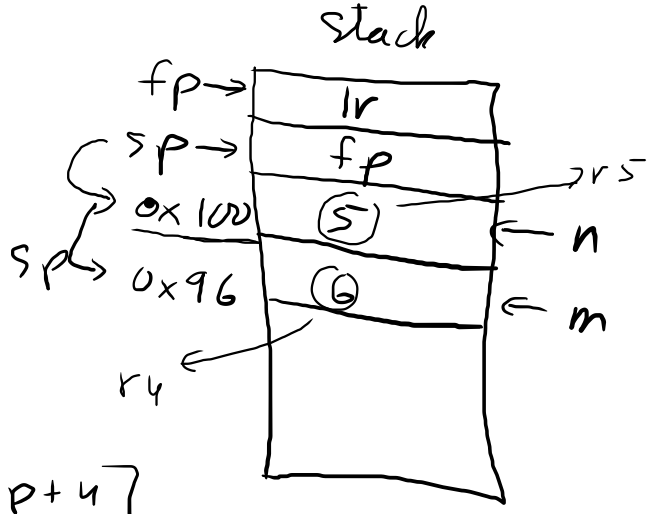
bl get_inp

@ [sp] → 6

@ [sp+4] → 5

ldr r4, [sp] @ r4 = *sp ≡ [sp]

ldr r5, [sp, #4] @ r5 = *(sp+4) ≡ [sp+4]



```
add r4, r4, r5   @ r4 = r4 + r5   @ r4 = sum
@ printf ("Sum = %d \n", sum)
                  r0          r1
mov r1, r4   @ r1 = r4 = sum
ldr r0, =outp
bl printf
@ restore fp, lr, return
sub sp, fp, #4
pop {fp, pc}
```

] @ last 2 steps of each function

get_inp.s

```
.cpu cortex-a53
.fpu neon-fp-armv8
.data
@ scanf("%d %d", &n, &m)
inp: .asciz "%d %d"
.text
.align 2
.global get_inp
.type get_inp, %function
get_inp:
    push {fp, lr}
    add fp, sp, #4
```

@ r0 = &n , r1 = &m

@ scanf ("%d %d", &n, &m)

r0 r1 r2

mov r2, r1 @ r2 = &m

mov r1, r0 @ r1 = &n

ldr r0, =inp

bl scanf

sub sp, fp, #4

pop {fp, pc}