October 11, 2021

- Arrays - 1D
- passing more than 4 args to a function

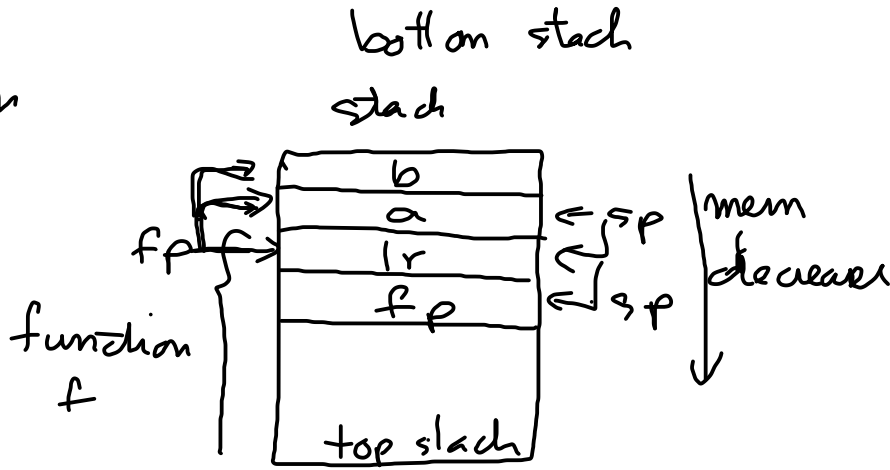
f(r0, r1, r2, r3, stack)

volatile

printf

f(r0, r1, r2, r3 | a, b) ←

1. push {b}
2. push {a}



push {r0, r1}

|||

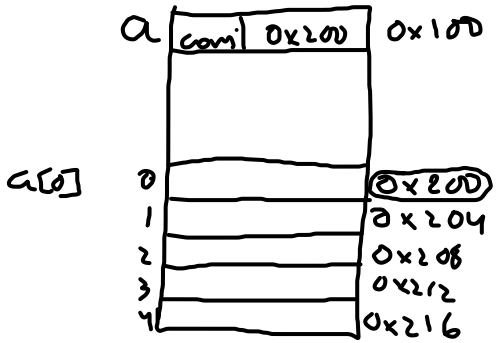
push {r1, r0}

r0
r1
r2

Arrays (1-D)

```
int a[5];      //int [] a = new int[5];
```

↓ stack



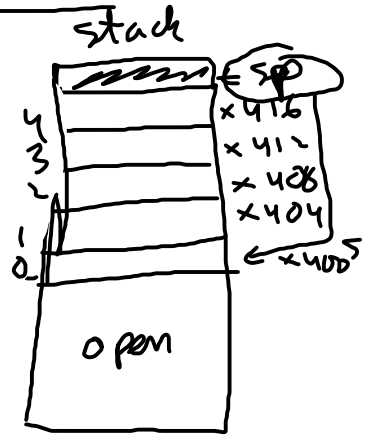

```
int b;
a = b;
```

```
int a[5];   ⟹   int *a;
```

sizeof(int) → 4 bytes

```
a = (int *)malloc(5 * 4);
```

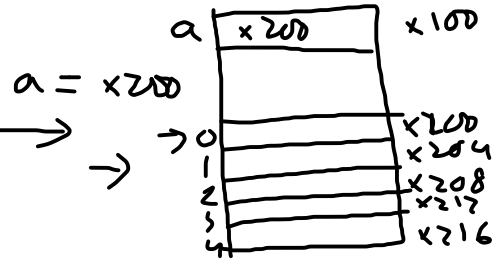
Arrays in ARM



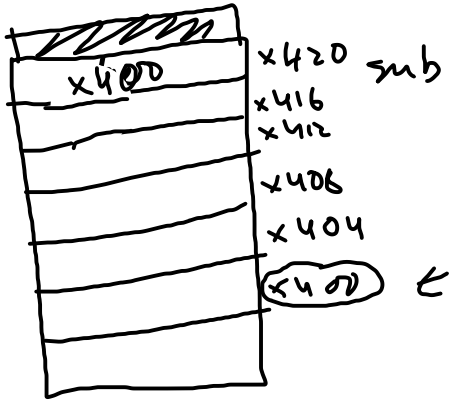
int a[5]; 5 * 4 = 20 bytes

⇒ sub sp, sp, #20

a ≡ sp



sp = x400

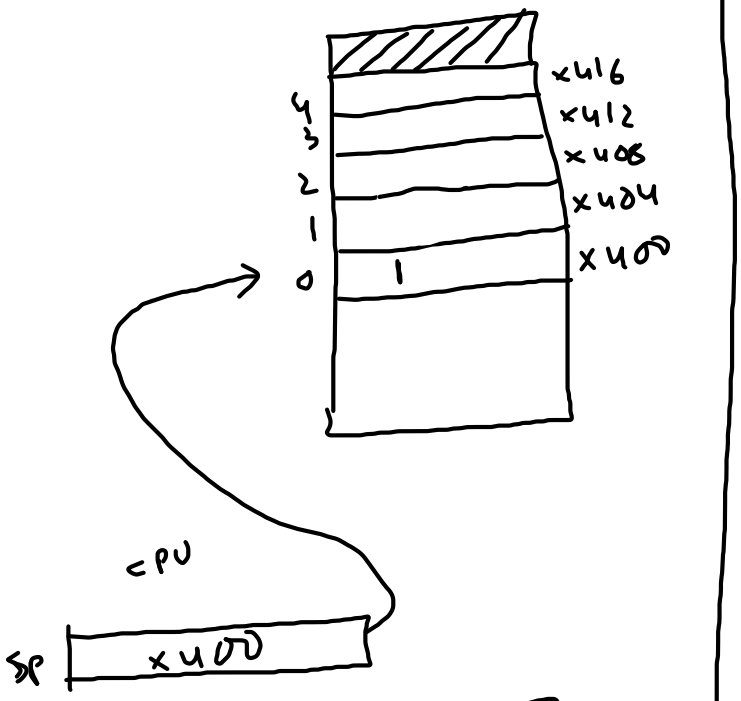


sub sp, sp, #24

```
void test()
{ int x[11]; → stack
```

malloc / new → heap

x = (int *) malloc (40);



SP ≡ a

a[0] = 1 , *a = 1

↑

[ ] ← dereference in assembler

↓

→ [SP] = 1

mov r0, #1 ←

str r0, [sp]] ⇒ a[0] = 1, *a = 1

a[1] = 2 , *(a+1) = 2

↓ ↘

*(x200 + 1 * 4)

*(*x204) = 1

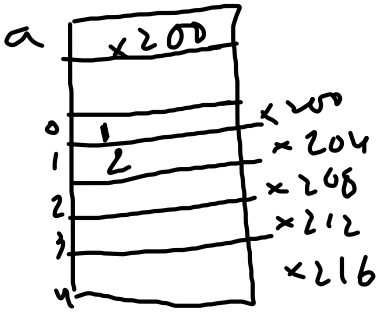
{mov r0, #2

str r0, [sp, #4] ↓

↓

//str r0, *(sp+4) [sp + 4] = 2

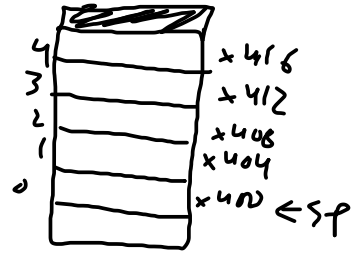
C



Ex: write a simple ARM code to store random #'s, 1 → 10 into the 5-elt array

main.s

```
    .
    .
    .
    mov r0, #0
    bl time
    bl srand    @ set the seed to current time
    @ r10 → i
    mov r10, #0 @ i=0
loop:
    cmp r10, #5 @ i<5
    bge quit_loop
    bl rand @ rand() → r0
    mov r1, #10
    bl mod  @ mod(r0,r1) → r0
    add r0, r0, #1  @ mod(r0,r1)+1
```

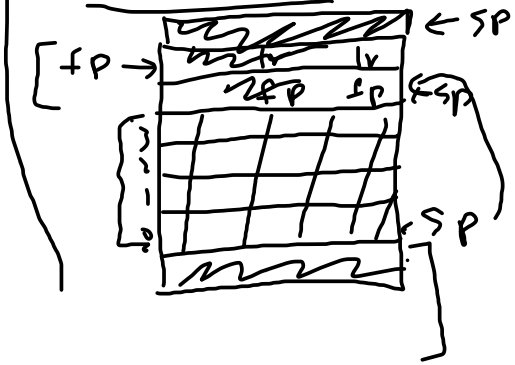


loop

```
for (i=0; i<5; i++)
{
    a[i] = rand() % 10 + 1;
}
```

(r0 → a, r1 → 1)

function




```
    .
    .
    .
    sub sp, fp, #4
```

@ Need to store r0 into a[i] → a[r10]

@ str r0, [sp, 4*r10]

① @ mov r1, #4
@ umul r1, r1, r10 @ r1 = r1 * i = 4*i
@ str r0, [sp, r1]

② @ mov r1, r10, LSL #2 @ left shift 2 bits
@ str r0, [sp, r1]

mov r1, #4
umul r1, r1, r10
str r0, [sp, r1] @ a[i] = rand()%10 + 1

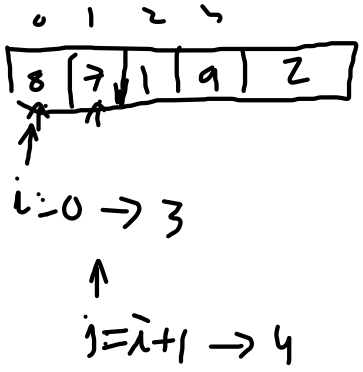
add r10, r10, #1 @ i = i+1

b loop

quit_loop:

a[0] ↓ 0*4 = i*4
str r0, [sp]
a[1] ↓ i*4 = i*4
str r0, [sp, #4]
a[4] byte offset
str r0, [sp, #16]
= i*4

Ex: sort array of random #'s



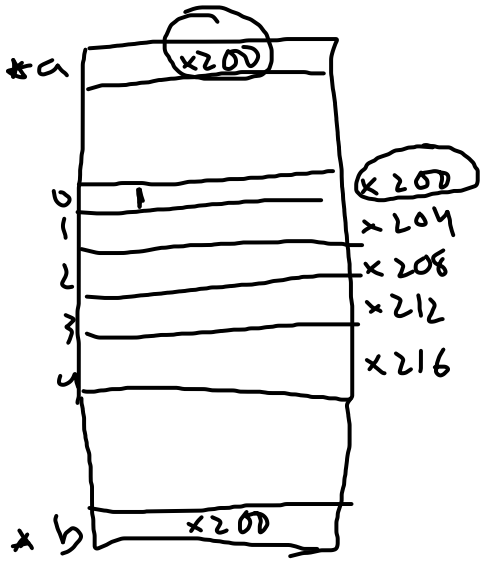

```
if (a[i] > a[j])
{
   temp = a[i];
   a[i] = a[j];
   a[j] = temp
}
```

```
for (i = 0; i < 4; i++)
{
   for (j = i+1; j < 5; j++)
   {  if (a[i] > a[j])
      {  int temp = a[i];
         a[i] = a[j];
      }  a[j] = temp;
   }
}
```

```
int main()
{ int a[5];

   test (a, 5);

}
```




```
void test (int *a)
   or
void test (int a[])
void test (int a[5])
```

```
int b[] = a;

void test(int b[], int n)
{
    b[0] = 1; // *(x200 + 0*4) = *(200) = 1

}
```

sort.s

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

.data

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

sort:
  push {fp, lr}
  add fp, sp, #4
@ sort(a,5) -> bl sort(r0=sp, r1=5)
@ r0 = address of first elt
@ r1 = 5
@ r10 -> i
@ r9 -> j

  mov r10, #0

loop_outer:
    sub r2, r1, #1   @ 5-1
    cmp r10, r2      @ i<4
    bge quit_outer
    add r9, r10, #1  @ j=i+1
loop_inner:
    cmp r9, r1       @ j<5
    bge quit_inner
    @ a[i] -> temp
    mov r3, r10, LSL #2
    ldr r3, [r0, r3] @ r3=a[i]
    ldr r4, r9, LSL #2
    ldr r4, [r0, r4] @ r4=a[j]
                                  cmp r3, r4
                                  ble else
```

```
@ a[i] = a[j]
@ str r4, [r0, i*4]
mov r5, r10, LSL #2
str r4, [r0, r5] @ a[i] = a[j]
@ a[j] = temp = r3
mov r4, r9, LSL #2
str r3, [r0, r4]
else:
add r9, r9, #1    @ j = j+1

quit_inner:

add r10, r10, #1   @ i = i+1
quit_outer:

sub sp, fp, #4
pop {fp, pc}
```

```
main:                                  int a[10];
    push {fp, lr}                          #40
    add fp, sp, #4
    @ allocate mem for array
                     24
    sub sp, sp, #20
                     ↑
    mov r0, sp       # els * 4
    mov r1, #5
                     ↑
    bl sort          mov r0, r0, LSL #2
    sub sp, fp, #4   sub sp, sp, r0
    pop {fp, pc}
```