.zip , .rar , .tar

apk

November 8, 2021

Lab 3

→ main.c checkprime.c

main.s checkprime.s

I/O check (n)

rφ ←

(1) main.c & checkprime.s

(2) main.s & checkprime.c

(3) main.s & checkprime.s

test.c → .... → test.s $\xrightarrow{\text{compiler}}$ test.o

Lab5 → end of nov

Lab6 ~ Dec. 18 → 2 days before grades are due

Lab 4 & 5

→ r0 → r3 ← function args passing (volatile)

r4 → r10 ← loops ~ at least 2 reg. for loop counter

r4 → r8

variables → stack

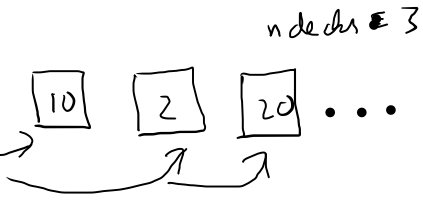
function (r0, r1, r2, r3, stack)

mov r3, #7

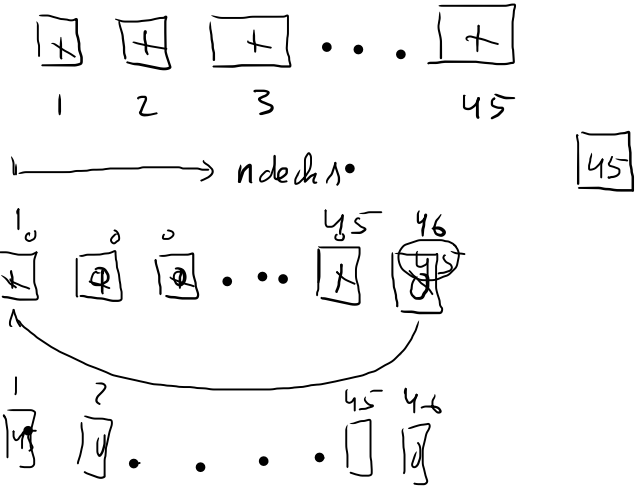
bl printf

Lab #4

(0) Init array 45



(1) Init decks



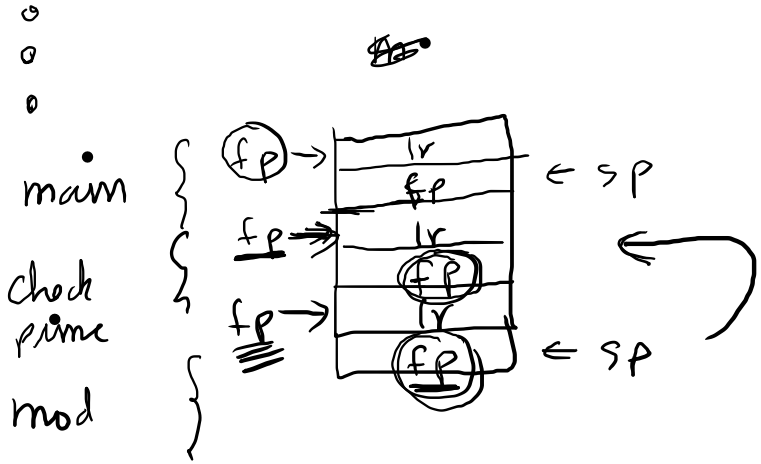
→ allocate 1-D array of 46 elements

init each elt → ϕ

push {fp, lr}

add fp, sp, #4

```
int decks[46] = {0};
int ndeck = ϕ;
init_array (decks, 46);
ndecks ← init_decks (decks, 46, &ndecks);
do
{
    print_decks (decks, ndecks);
    make_move (decks, ndecks);
    remove_zeros (decks, &ndecks);
} while (!endgame (decks, ndecks));
```



push {fp, lr}

sub sp, fp, #4

pop {fp, pc}

Lab 4

⋮

main:

```
push {fp, lr}
add fp, sp, #4
@ allocate memory for 46 elts array
@ calculate # of bytes equivalent to 46 elts
@ #bytes = 46 * 4
mov r1, #4
mul r0, r0, r1
mov r0, #48   @ immediate value <= 255
mov r0, r0, LSL #2
→ sub sp, sp, r0
str sp, [fp, #-8]   @ *(fp-8) = sp
mov r0, #0
str r0, [fp, #-12]   @ ndecks = 0
```

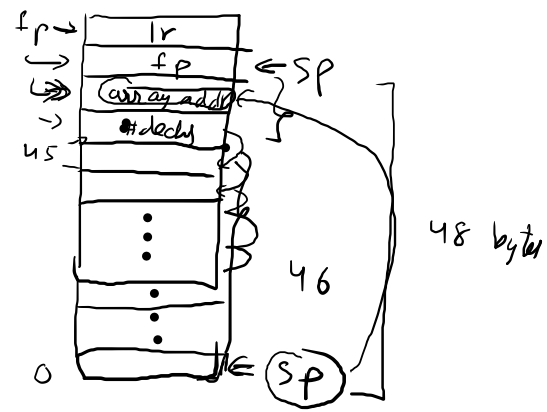
system

```
swi
mov r0, #0
bl time
bl srand
```

C

```
int main()
{ int decks[46];
  // int *decks;
  // decks = (int *)malloc(46 * sizeof(int));
```

stack



@ init_array (array, 46)
r0

```
ldr r0, [fp, #-8]
```
↑ offset

```
mov r1, #46
bl init_array
```

@ init_deck

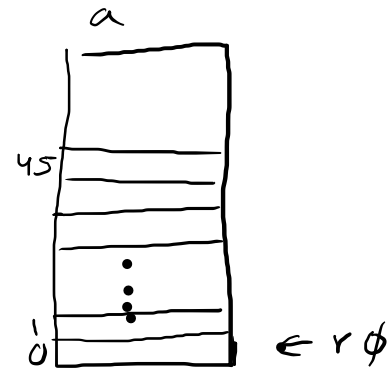
@ init_deck (array, 46, &ndecks)
r0 r1 r2

```
ldr r0, [fp, #-8]
mov r1, #45
sub r2, fp, #12
bl init_deck
```

i * 4

C

```
void init_array(int *a,
                int n)
{
}
```



for (int i = 0;

a[0] → ldr r4, [r0, #0]

a[1] → ldr r4, [r0, #4]

a[2] → ldr r4, [r0, #8]

r2 ← counter * 4

a[i] ldr r4, [r0, r2]

init_array.s

```
.
.
.

init_array:
    push {fp, lr}
    add fp, sp, #4
    @ r0 = address of array , r1 = # of elts
    @ r10 ← i
    mov r10, #0      @ i = 0
init_array_loop:  @ goto point
    cmp r10, r1      @ i < r1
    bge end_init_array_loop
    mov r2, #0
    mov r3, r10, LSL #2
    str r2, [r0, r3]
```

$[r0, \overbrace{i*4}] = 0$

```
for (i = 0; i < r1; i++)
{
    *(r0 + i) = 0;
    // r0[i] = 0
}
```

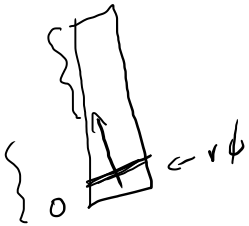
```
    add r10, r10, #1    @ i++
    b init_array_loop
end_init_array_loop:
    sub sp, fp, #4
    pop {fp, pc}
```

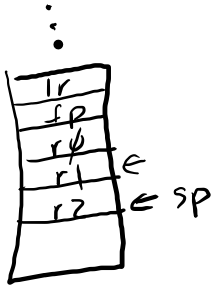
init_deck.s

⋮

```
init_deck:
    push {fp, lr}
    add fp, sp, #4
    @ r0 ← address of array, r1 ← 45, r2 = & ndecks
    push {r2, r1, r0}
    mov r10, #0     @ r10 = 0
init_deck_loop:
    cmp r1, #0
    ble init_deck_loop_end     @ r1 > 0

    bl rand     @ r0 = rand()
    @ mod(r0, r1)
    ldr r1, [sp, #4]
    bl mod      @ r0 = rand() % r1
    add r4, r0, #1     @ rand() % r1 + 1 = r4
```



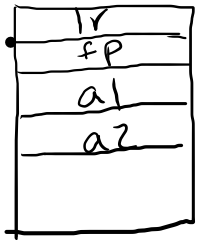


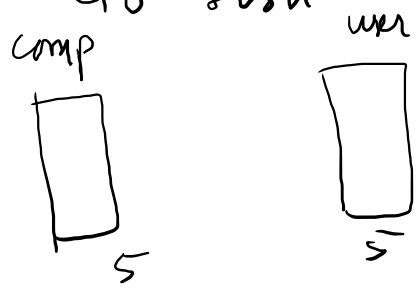

```
r1  = 45;
r10 = 0;
while (r1 > 0)
{ // rand # 1 -> r1
  r4 = rand() % r1 + 1;
  *(r0 + r10) = r4;
  r10 = r10 + 1;
  r1 = r1 - r4;
}
```

```
@ *(r0 + r10) = r4
ldr r0, [sp, #8]
mov r1, r10, LSL #2   @ r10 * 4 => r1
str r4, [r0, r1]      @ *(r0 + r10) = r4
add r10, r10, #1      @ r10 = r10 + 1
ldr r1, [sp, #4]
sub r1, r1, r4        @ r1 = r1 - r4
str r1, [sp, #4]
b init_dech_loop
init_dech_loop_end:
    sub sp, fp, #4
    pop {fp, pc}
```

Lab 5

Go-fish



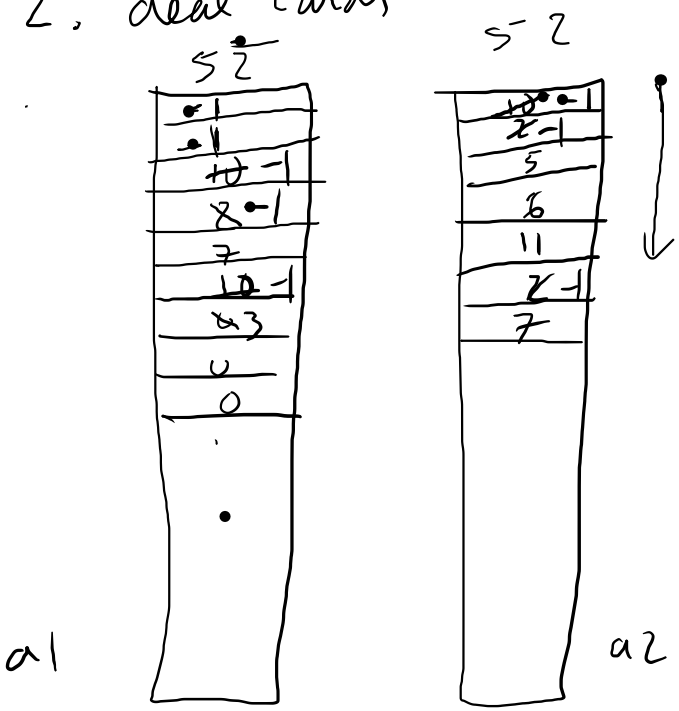


52 -cards

1 - A
2 - 2

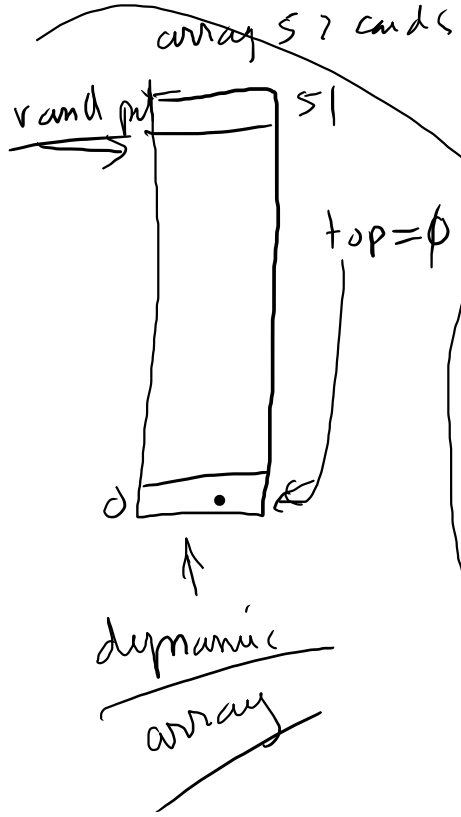
10 → 10
11 → J
12 - Q
13 - K

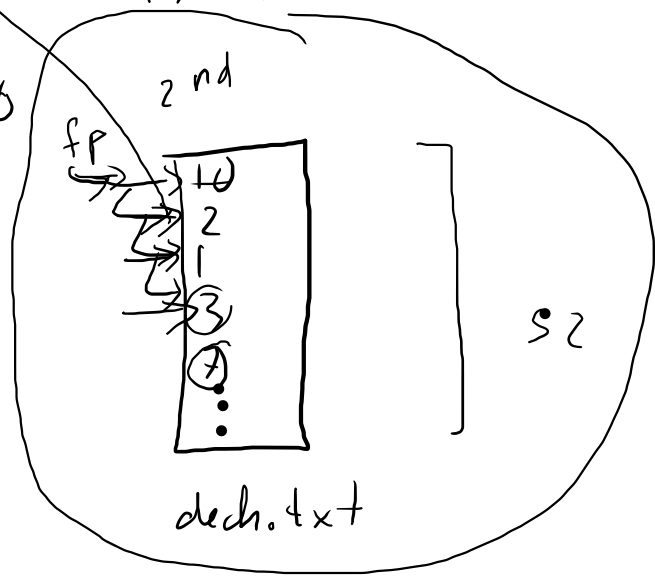
1. shuffle deck (file)
2. deal cards



np1 = 0 K 2
dyn array

np2 = 0 1
array





3. check_pairs for both player, update np1, np2

while (!players have no cards left)

{

4. p1 ask for card

5. p2 ask for card

}

1. if p2 has a card, x for over and update np1

2. if p2 does not have card, then go-fish

4. ask_for_cards (deck, p1, p2, & np1)

r0 r1 r2 r3

fp

5. ask_for_cards (deck, p2, p1, & np2)

6. comp np1 vs np2