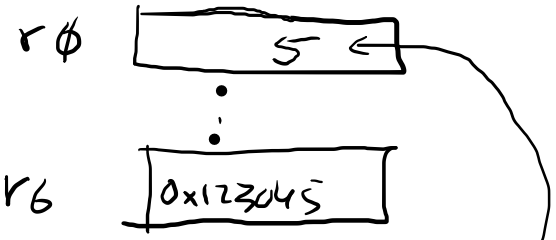
October 4, 2021

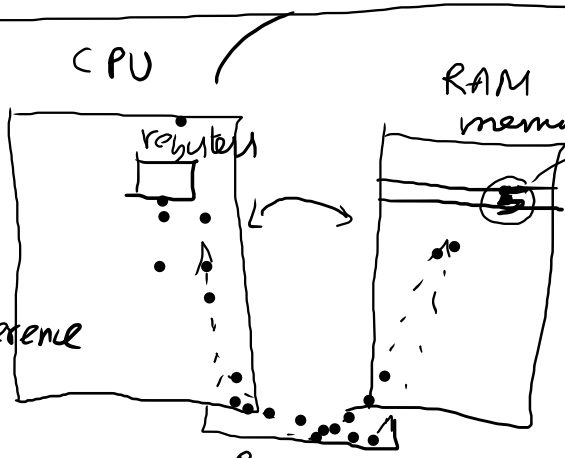
- scanf to input 2+ #'s.
- ldr, e.g. ldr r0, [r6]] (1)
- swap function
- 1-bit adder, ADDER (2)
- Arrays
- sort chars of strings] (3)

int x[10]; ↝ ARM



(reg) ← (memory)
Dest source

ldr r0, [r6]



ldr : memory ↝ CPU

str : CPU ↝ memory

mov r6, #0x123456 [ ] ↔ dereference

ldr r0, [r6] — in C: int *a;

a = 0x123456;

*a = 5;

ldr vs ldrb

str vs strb

memory

stack

$2^8 - 1$

#5

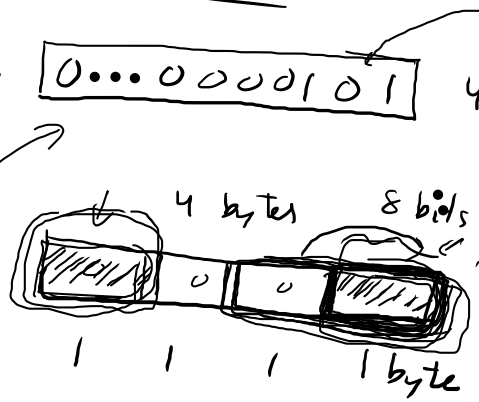
int => 4 bytes of memory

0 . 0000 0101

32 bits

mov r0, #5

str r0, [sp]



$0 \to 255$ ASCII

$0 \to 2^{16} - 1$

4 bytes = 32 bits

char → 1 byte

'a' → 97

sp 0x100000

10...0 (24'0)

printf("%c", c)

strb r0

str

ldrw

11111111 0...0 (24)

w = word → 2 bytes

sp

1 byte

ldr vs ldrb

Multiple inputs

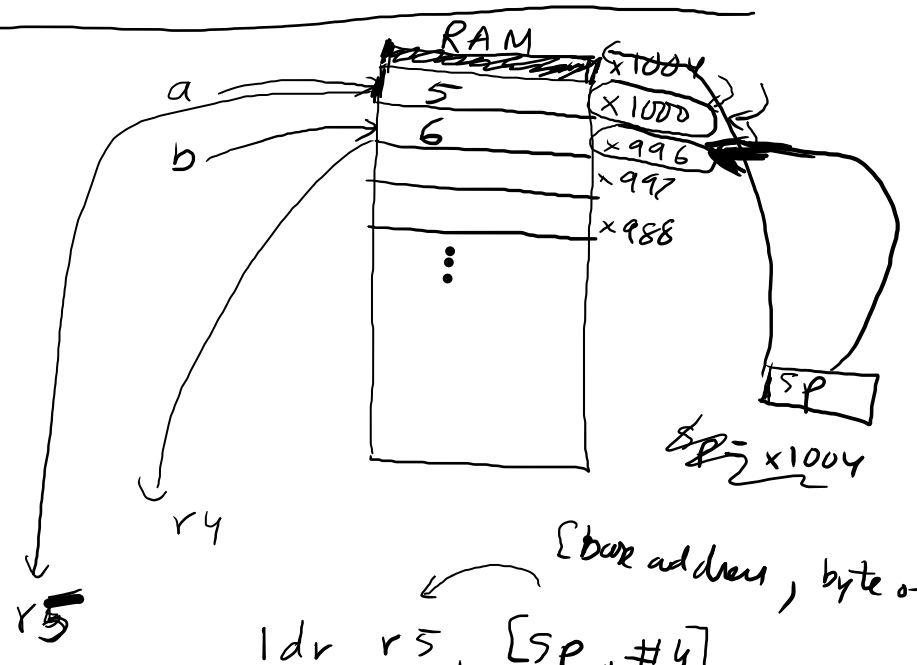
```
int a, b;
scanf("%d %d", &a, &b);
```

(×1000, ×996) — C / C++

r0 r1 r2

Emulate on assembler:

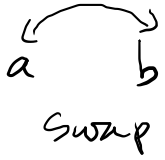
```
.data
inp: .asciz "%d %d"
     ⋮
ldr r0, =inp
sub sp, sp, #4   @ sp = x1000
mov r1, sp       @ r1 = x1000
sub sp, sp, #4   @ sp = x996
mov r2, sp       @ r2 = x996

@ r0 = "%d %d", r1 = &a, r2 = &b
bl scanf "%d %d", r1 = &a, r2 = &b

ldr r4, [sp]
```



[base address, byte offset]

```
ldr r5, [sp, #4]
@ ldr r5 [sp, #4]
```

Swapping



a = 3
b = 4

⇅ swap

a = 4
b = 3

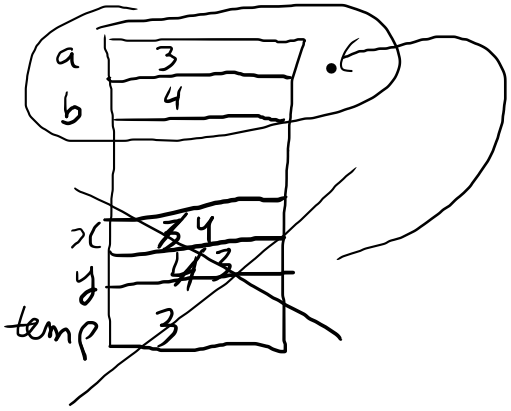
call by value

C

```
int main()
{ int a=3, b=4;

  swap(a, b);
  printf("%d %d\n", a, b);
}
```

↳ 3 4

```
int x = a    int y = b
void swap (int x, int y)
{ int temp;

   temp = x;
   x = y;
   y = temp;

}
```

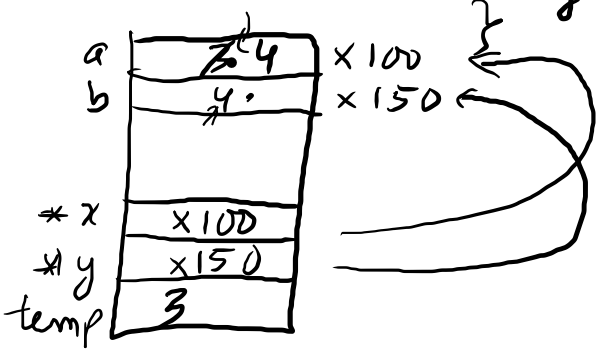


```
int main()
{ int a=3, b=4;
  swap(&a, &b);
```

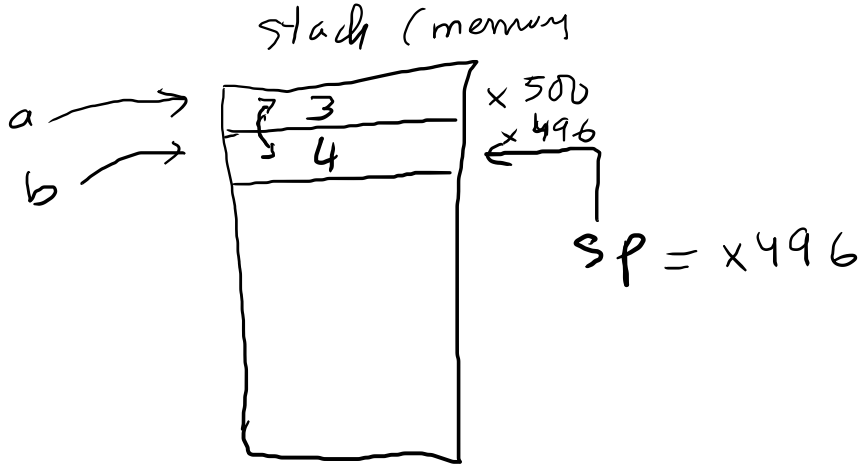
call by

fake reference

```
int *x = &a        int *y = &b
void swap (int *x, int *y)
{ int temp;
  // temp = x
  temp = *x;
  // x = y
  *x = *y;
  *y = temp;
}
```



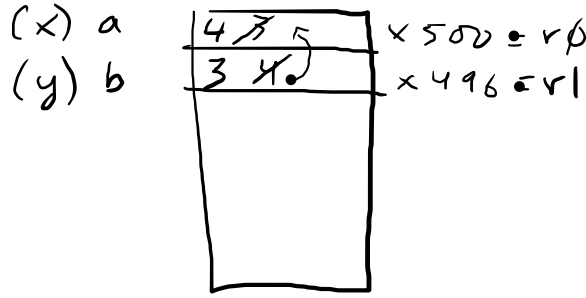
Assembler




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

    mov r0, #3
    push {r0}

    mov r0, #4

    push {r0}   r0 r1
    @ swap(&a, &b)
    @ r0 ← x500 , r1← x496

    mov r1, sp  @ r1 = sp = x496

    add r0, sp, #4   @ r0 = sp + 4 = 496 + 4 = x500
    bl swap
```

swap.s

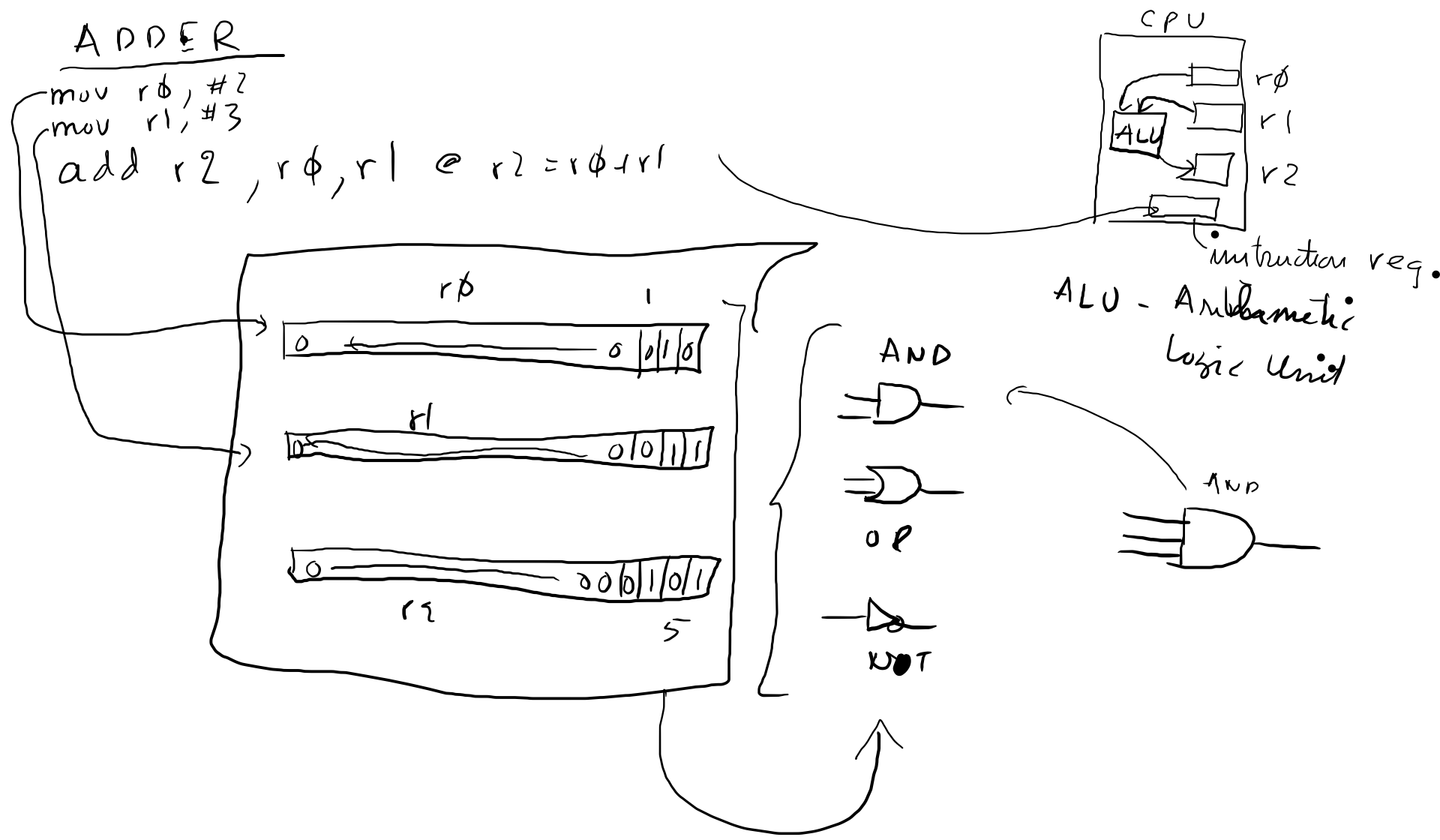
```
swap:
    push {fp, lr}
    add fp, sp, #4
@ r0 = address of a , r1 = address of b
@ temp = *a , r4 = temp
    ldr r4, [r0] @ r4 = *a = *r0
    ldr r2, [r1] @ r2 = *b = [r1] = 4
    str r2, [r0] @ *r0 = r2 = 4
@ *b = temp = r4 = *r1 = [r1]
    str r4, [r1] @ [r1] = temp
    sub sp, fp, #4
    pop {fp, pc}
```



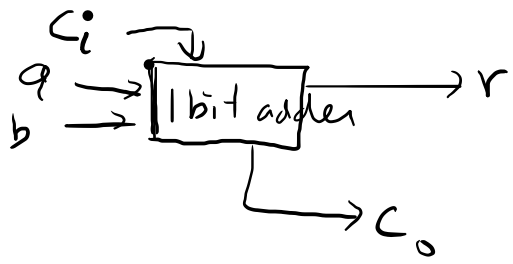

```
    mov r4, #3
    mov r5, #4
    push {r4} @ a
    push {r5} @ b
    mov r0, sp
    add r0, sp, #4
    bl swap
    pop {r5}
    pop {r4}
```

ADDER

```
mov r0, #2
mov r1, #3
add r2, r0, r1  @ r2 = r0 + r1
```



Build Logic Table



Swap

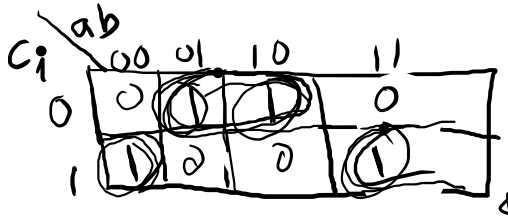
$2^3 = 8$

a	b	c_i	r	c_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

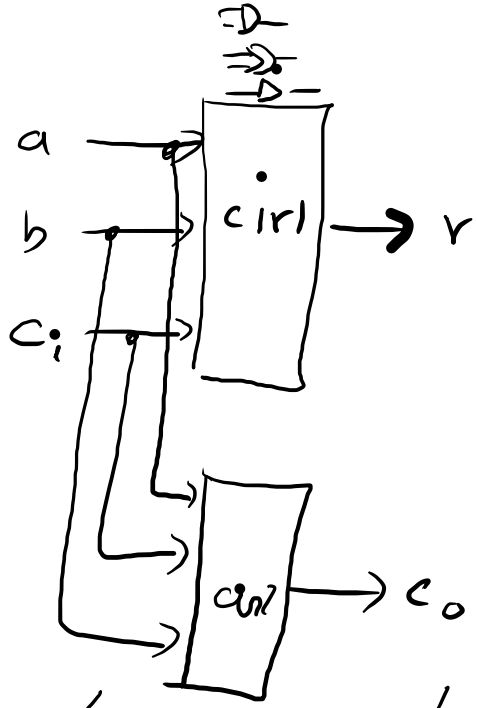
cirl circuit

cir2 circuit

K-map

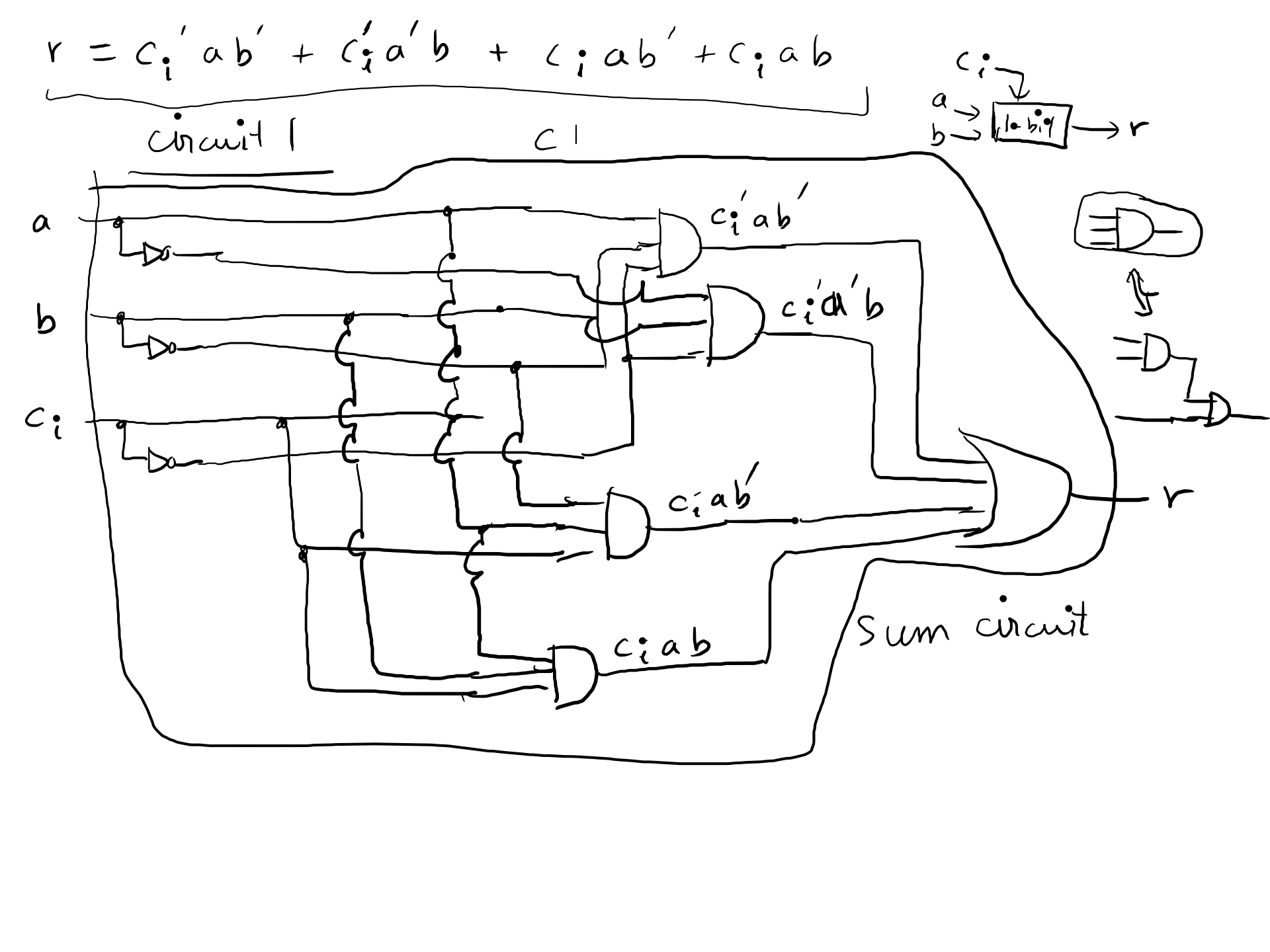


circuit for r



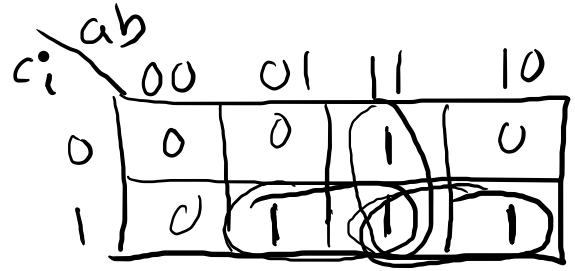
$r = c_i' a'b + ab' + c_i a b' + c_i ab$

$$r = c_i' a b' + c_i' a' b + c_i a b' + c_i a b$$



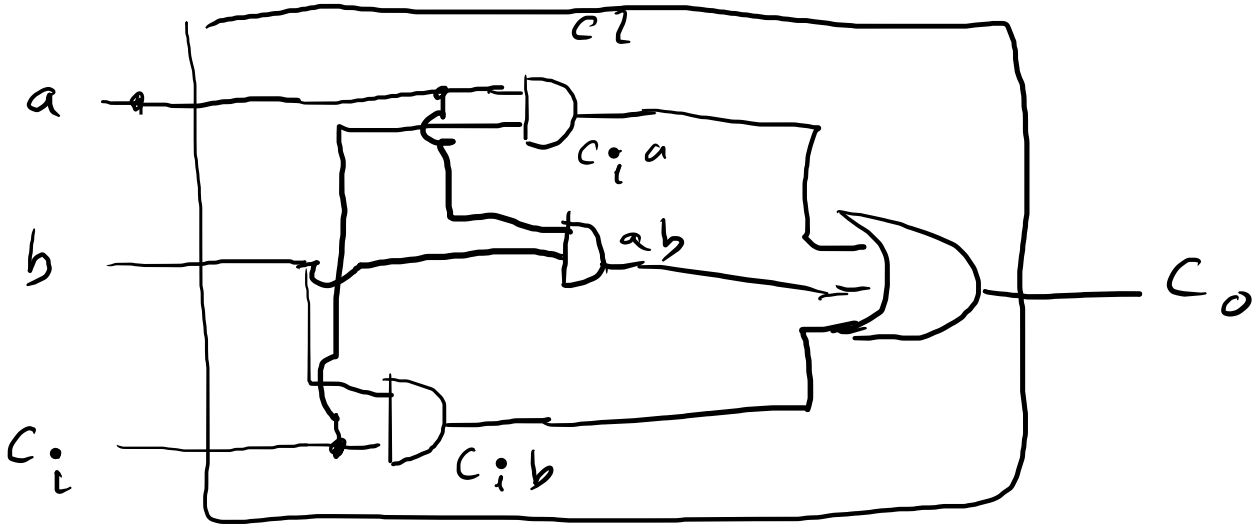
Carry

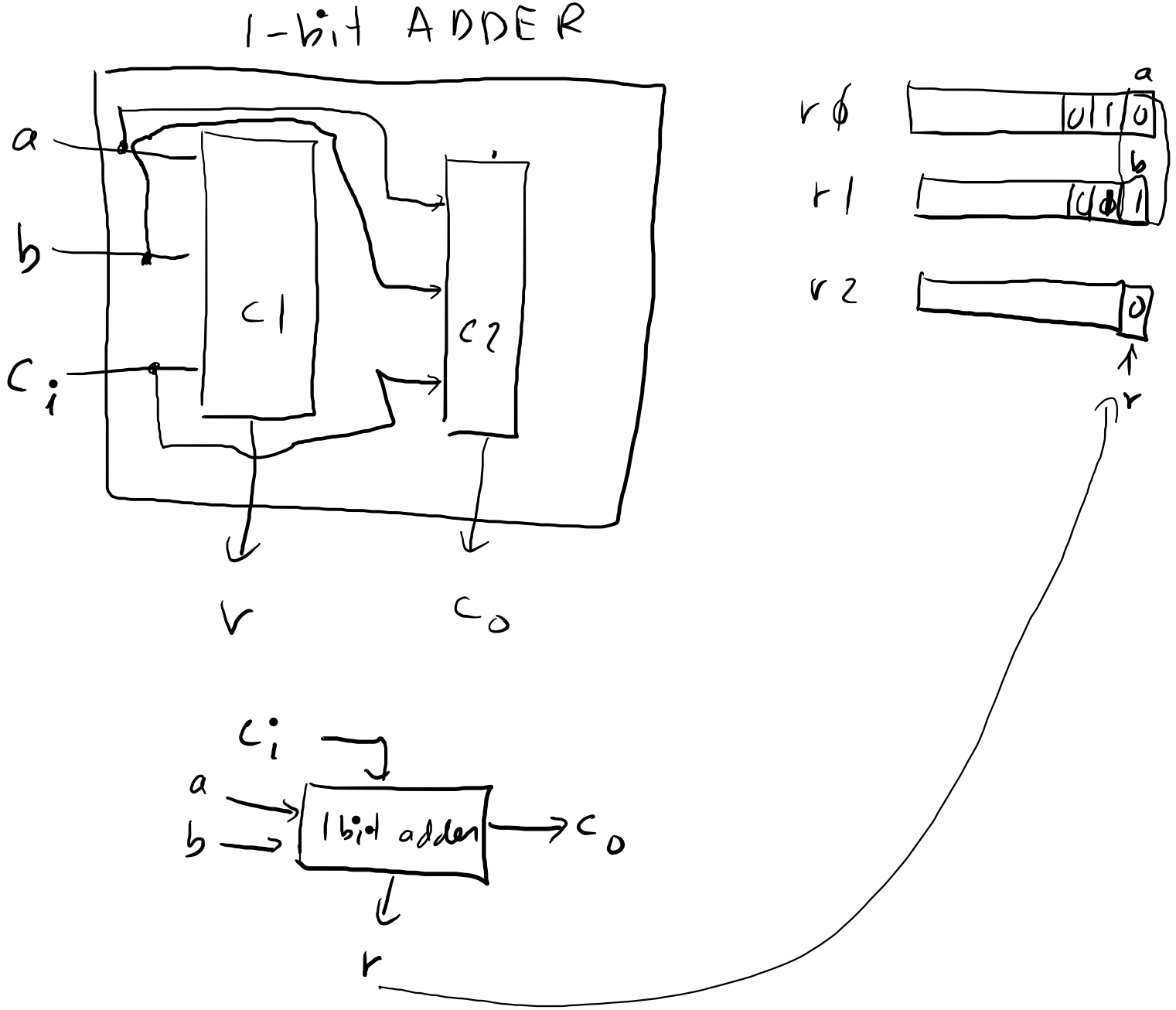
K-map



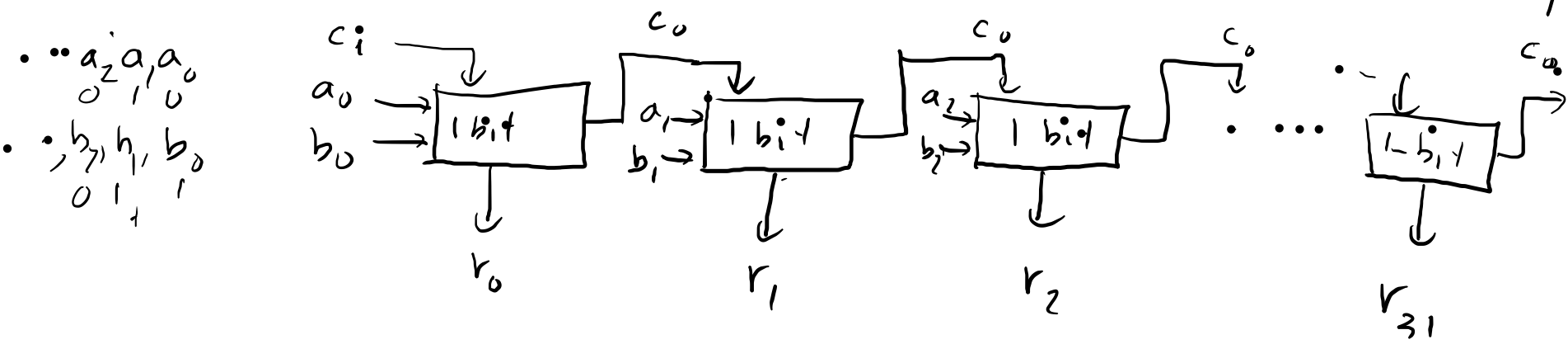
circuit 2

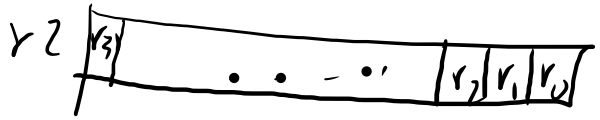
$$c_o = c_i b + c_i a + ab$$




1-bit ADDER
a
b
c_i
c1
c2
r
c_o
r0
r1
r2
a
b
r
c_i
a
b
1bit adder
c_o
r

Full adder


c flag
in CPSR
set
c_i
a_0
b_0
1 bit
c_0
a_1
b_1
1 bit
c_0
a_2
b_2
1 bit
c_0
1 bit
c_0
r_0
r_1
r_2
r_{31}


r2
r_{31}
r_2
r_1
r_0