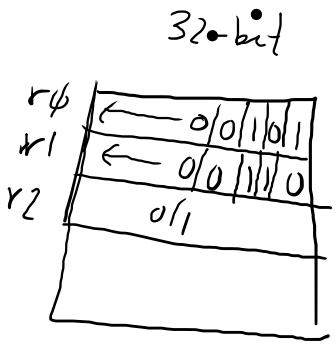
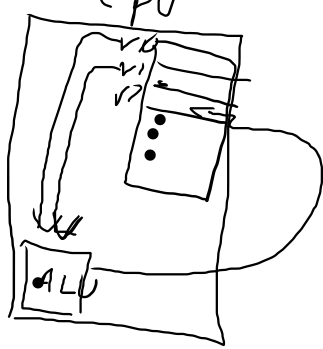
September 20, 2021

- Loops ✓ ✓
- if-else ✓ ✓
- computer arch ~ 30 min ✓
- More practice w/ stacks ✓
- codeblocks
- comparing values ✓ if (r1 < r2) ✓

Computer Architecture

```
mov r0, #5 → ..0101
mov r1, #6
add r2, r0, r1
```





r0: 00...00101
r1: 00...01010

ALU Hardware — ADDER

00...00101111

int a vs unsigned int b;

Hardware

- NOT, AND, OR gate
  - set
  - alg. abstract
  - graph

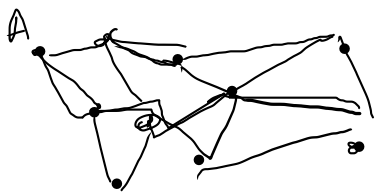
stack, queue, linked list tree.

x	NOT (!, ~)
0	1
1	0

A	B	A && B (AND)
0	0	0
1	0	0
0	1	0
1	1	1

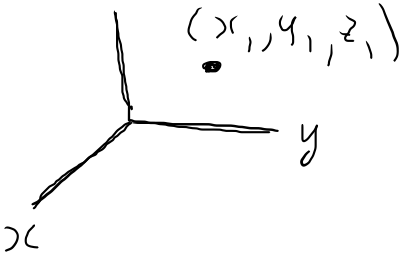
A	B	A || B
0	0	0
1	0	1
0	1	1
1	1	1

Graph



Page rank

Basis function



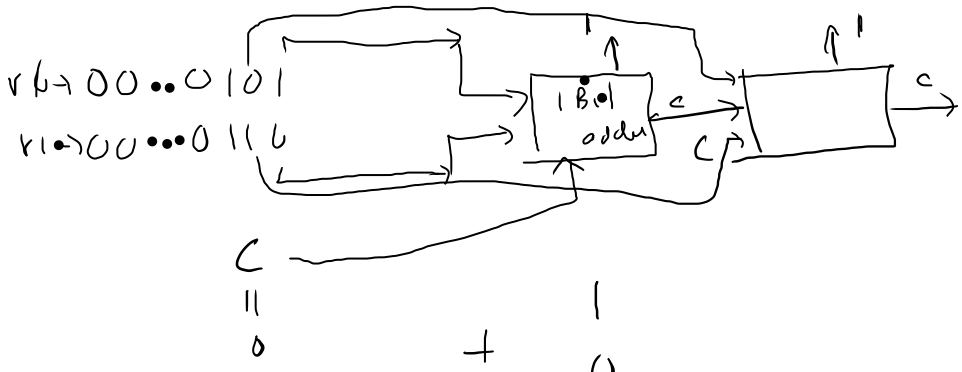
B_1: (1, 0, 0)

B_2: (0, 1, 0)

B_3: (0, 0, 1)

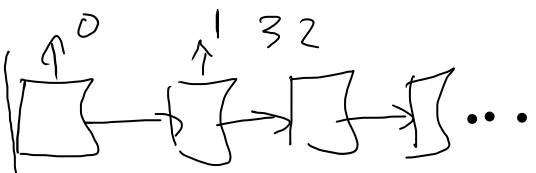
Basis function

$(x_1, y_1, z_1) = x_1(1,0,0) + y_1(0,1,0) + z_1(0,0,1)$

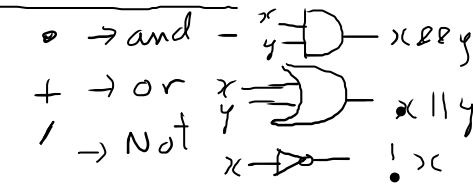


c	A	B	+	C
0	0	0	0	0
0	1	0	1	0
0	0	1	1	0
0	1	1	0	1

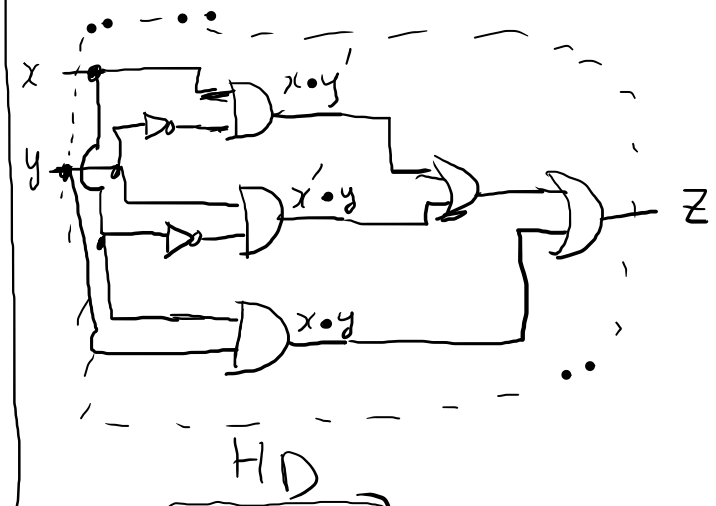
A (~~&&~~, ||, !.) B → +, C



Gate



$Z = x \cdot y' + x' \cdot y + x \cdot y$



7 gates

$$Z = x \cdot y' + x' \cdot y + \overbrace{x \cdot y}^{a} + \overbrace{x \cdot y}^{a}$$

$$\underbrace{x \cdot y' + x \cdot y} + x' \cdot y + x \cdot y$$

$$x(y' + y) + (x' + x)y$$

$$= x \cdot 1 + 1 \cdot y$$

$$= x + y$$

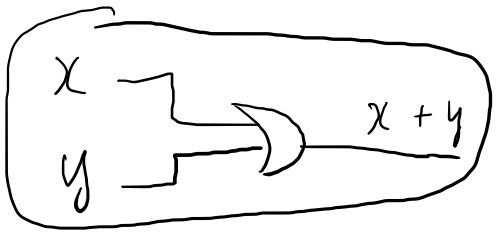
$a + a = a$

a	$a + a$
0	0
1	1

$a + a' = 1$

a	$a + a$
0	1
1	1

word



1 gate vs 7 gates

Key

- algebraically
- → = Graphically (K-map)

• if . else / compare values

cmp arg1, arg2 → arg1 − arg2

cmp #2, #3 (if (2 < 3)

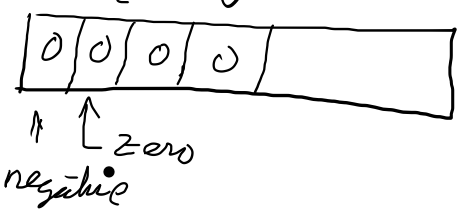
2 − 3 = −1

blt opl (2 < 3)

else:
    false
    b skip
opl:
    true
skip:

if (2 < 3)
{
opl:
    ...
}
else
{
    ...
}
↓

cpsr register



1 0 0 0

a1 < a2

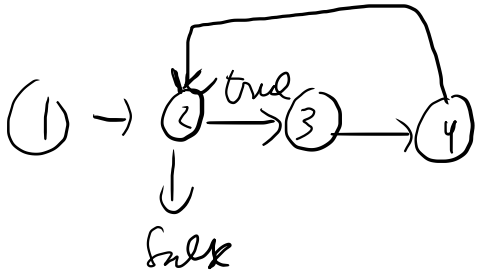
- lt - less than
- gt - greater than
  a1 > a2
- eq - a1 == a2
- ne - a1 ≠ a2
- ge - a1 >= a2
- le - a1 <= a2

Loops

(for, while, do-while) → same

```
for (init ; logical check ; update)
{
        operations
}
```

(1) init (2) logical check (3) operations (4) update



while ⇔ for

```
init
while (logical check)
{
        operations
        update
}
```

Assembler

There are no loops

pseudo

ARM loop syntax

```
init (a=0)   @ (1)
loop:
    cmp a, #100   @ (2)
    bge quit      @ (2)
    ≡ @(3)
    add a, a, #1  @ (4) update
    b loop
quit:
```

```
(required) int b;
for (int a = p1 ...
```

```
for (int a = 0; a < 100; a++)
{
    ≡
}
```

```
mov r4, #0   @ r4 ≡ a, counter variable
loop:
    cmp r4, #100   @ a < 100
    bge out_loop
    @ ≡
    add r4, r4, #1   @ a++
    b loop
out_loop:
```

```
for(int a = 100 ; a >= 0 ; a--)
{
    // opcodes
}
```

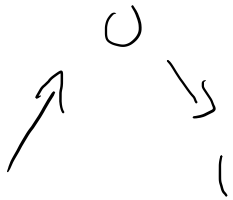
↓

```
@ use r4 = a
    mov r4, #100   @ a = 100
loop:
    cmp r4, #0   @  a > 0
    ble quit_loop
    @ opcodes
    sub r4, r4, #1  @ a--
    b loop
quit_loop:
```

Example: write a program that plays rock / paper / scissors

paper / scissor / rock
0 1 2

$0 < 1 < 2 < 0$



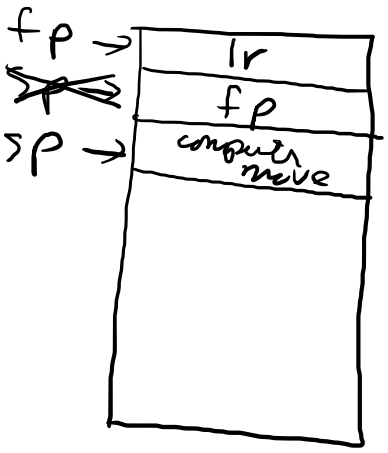
- random # generator
- rand() → 0 and LARGE ← <stdlib.h>
  - rand() % 2 → 0, 1, or 2 — mod(n, 2) / n % 2
  - srand(seed) — SEED
  - time(0); — time.h → current-time in millisec

srand(time(0)) ← set seed to current time

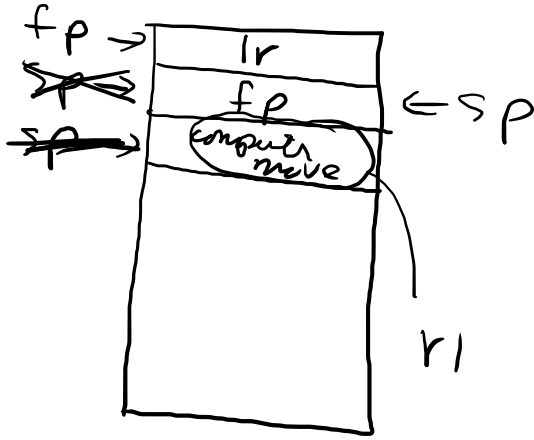
main.s | computer.s | mod.s | who wins.s | user_move

```
.cpu cortex-a53 @ pi 3
.fpu neon-fp-armv8

.data
cw: .asciz "computer wins\n"
iw: .asciz "I win\n"
.text
.align 2
.global main
.type main, %function
main:
    push {fp,lr} @ save lr, then fp
    add fp, sp, #4
```

```
@ reset the seed
@ time(0) -> t
mov r0, #0
bl time  @ returns time into r0 = t
@ srand(t) because t is already in r0
bl srand  @ reset rand seed

@ get computer move
@  computer() -> 0,1,2
bl computer
push {r0}  @ save computer move onto stack
```



```
@ reset the seed
@ time(0) -> t
mov r0, #0
bl time  @ returns time into r0 = t
@ srand(t) because t is already in r0
bl srand  @ reset rand seed

@ get computer move
@ computer() -> 0,1,2
bl computer
push {r0}  @ save computer move onto stack

@ user_move() -> 0,1,2
bl user_move  @ r0
pop {r1}
```



r0 ← user
r1 ← computer

@ whowins(r0, r1) → whowins(user, computer)

@ whowins → 0 if I win, 1 if computer wins → r0

```
cmp r0, #0          ←
bne comp_win
ldr r0, =iw
bl printf
b done
comp_win:
ldr r0, =cw
bl printf

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

computer.s

```
computer:
    push {fp, lr}
    add fp, sp, #4
    @ rand() % 2
    bl rand() @ r0 = rand number
    @ mod(r0, 2)
    mov r1, #2
    bl mod @ 0, 1, 2 -> r0
    sub sp, fp, #4
    pop {fp, pc}
```

mod.s

```
mod:
    push {fp, lr}
    add fp, sp, #4
    @ r0 = n, r1 = m
    @ mod(n,m) = n % m
    udiv r2, r0, r1
    mul r2, r1, r2
    sub r2, r0, r2
    mov r0, r2
    sub sp, fp, #4
    pop {fp, pc}
```

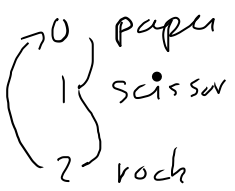
user_move.s

```
.data:
user_inp: .asciz "Enter 0-paper, 1-scissor, 2-rock:\n"
inp: .asciz "%d"
.text
:

user_move:
    push {fp, lr}
    add fp, sp, #4
    @ r4 <- user input
    mov r4, #3
```

```
user_loop:    @ if r4 >= 0 and r4 <= 2
    cmp r4, #0
    blt get_again
    cmp r4, #2
    bgt get_again

    b guid_user_loop

get_again:
    ldr r0, =user_inp
    bl printf
    ldr r0, =inp    @ scanf("%d", sp)
    sub sp, sp, #4
    mov r1, sp
    bl scanf
```

```
    ldr r4, [sp]
    b user_loop
quit_user_loop:
    mov r0, r4
    sub sp, fp, #4
    pop {fp, pc}
```

whowins.s




```
whowins:
    push {fp, lr}
    add fp, sp, #4
    @ r0 <- user, r1 <- computer
    @ 0 <- user wins, 1 computer wins
    @ 2 <- tie
    sub r4, r0, r1
```

r0 - r1 → -1 computer wins
→ -2 I win
→ 0 tie
→ 1 I win
→ 2 computer wins

```
cmp r4, #-1
beq comp_wins
cmp r4, #2
beq comp_wins
cmp r4, #0
beq tie
cmp r4, #1
beq iwin
cmp r4, #-2
beq iwin
b quit_whowins
```

```
comp_wins:
    mov r0, #1
    b quit_whowins
iwin:
    mov r0, #0
    b quit_whowins
tie:
    mov r0, #2
quit_whowins:
    sub sp, fp, #4
    pop {fp, pc}
```