False	True
N	Y
0	1
Black	White

$0 \cdot A = 0$
$1 \cdot A = 1$
$A + 1 = 1$
$A + 0 = A$
$A + A = A$
$A \cdot A = A$
$A \cdot \bar{A} = 0$
$A + \bar{A} = 1$

$A + B \cdot C = (A + B) \cdot (A + C)$

$\overline{AB} = \bar{A} + \bar{B}$

$\overline{A + B} = \bar{A} \cdot \bar{B}$

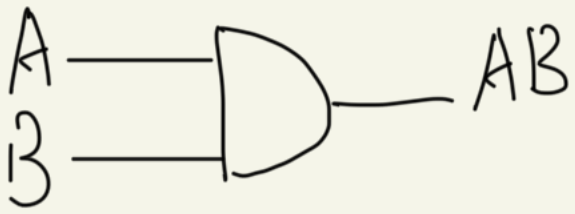
NOT	$\bar{A}$
AND	$A \cdot B$
OR	$A + B$
NAND	$\overline{A + B}$

if y then x

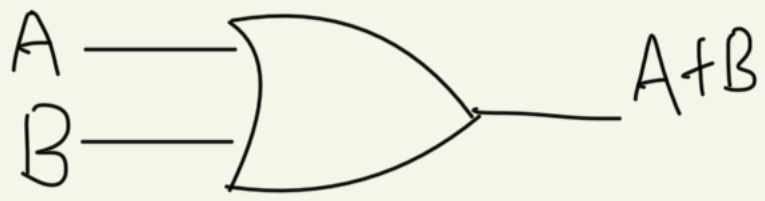
x	0	0	1	1
y	0	1	0	1
	1	0	1	1

if x then y 1 1 0 1

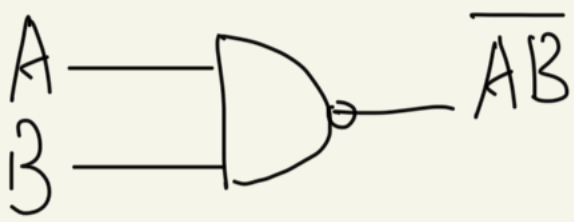
Logic Gates



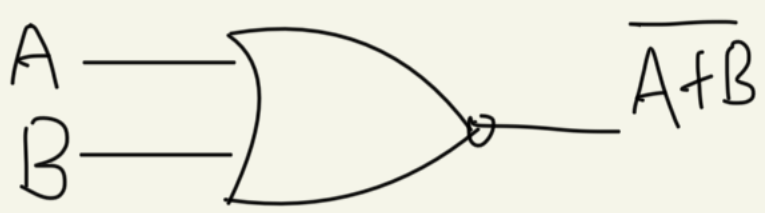
AND



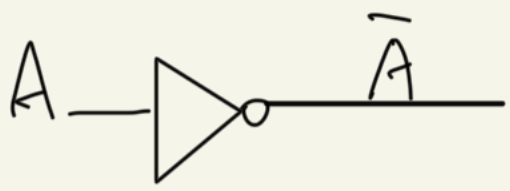
OR



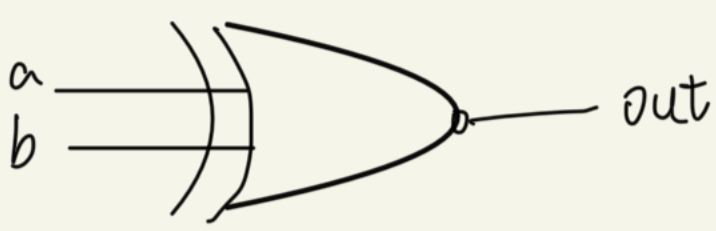
NAND



NOR



NOT

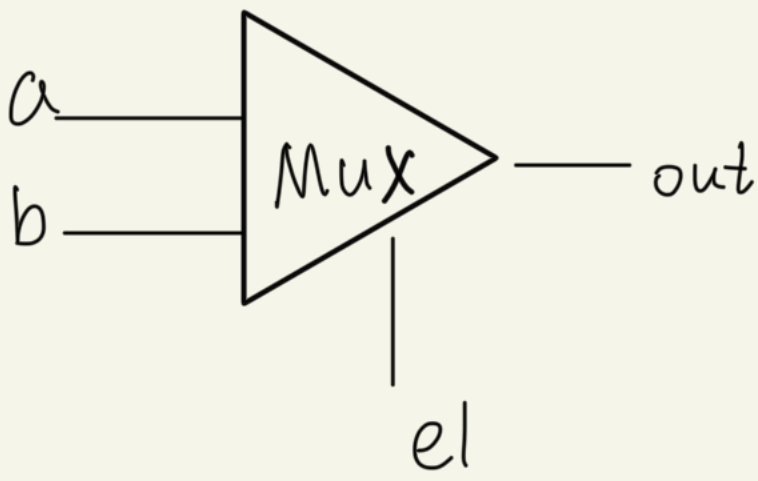


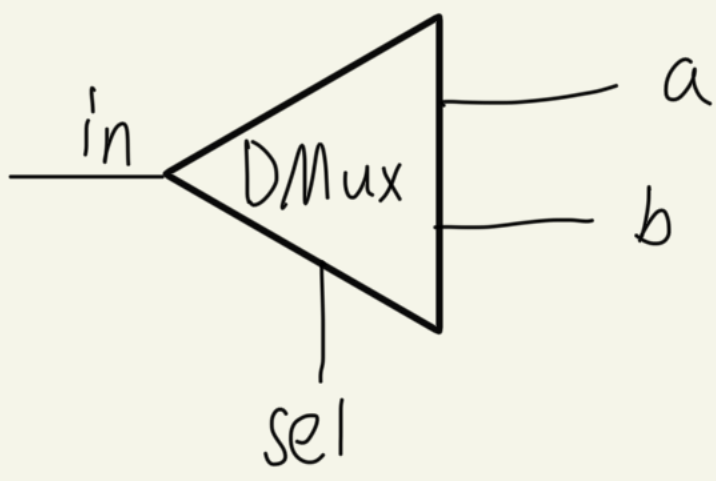
a	b	out
0	1	1
0	0	0
1	1	0

只有1个1返回1

XOR

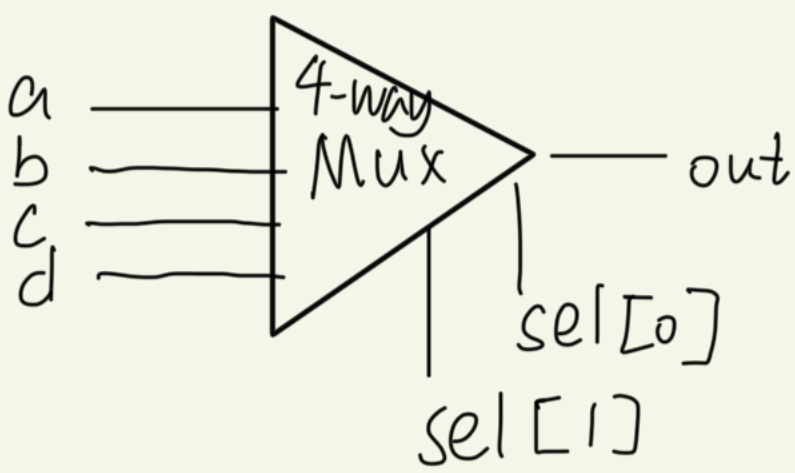
1	0	1





a	b	sel	out
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

sel	a	b
0	in	0
1	0	in



二进制

1110 0010

↑

第1位是1就是-128

-128+64+32+0+0+0+2=-30

如何把正数 invert 负数

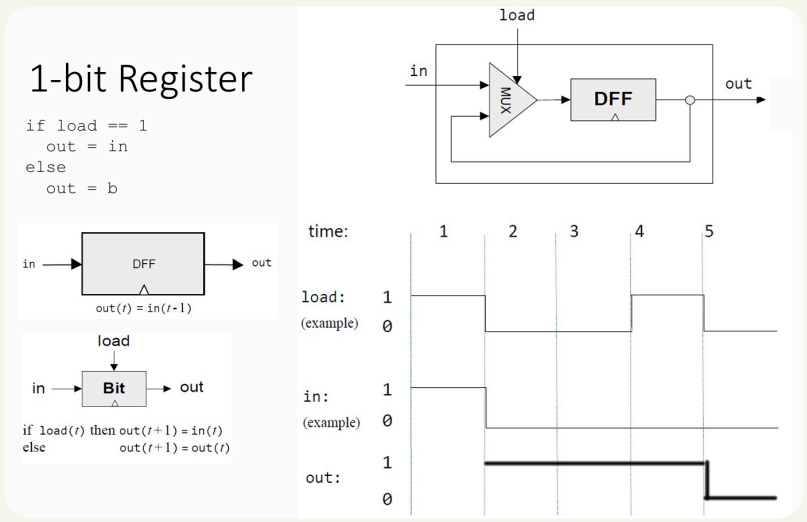
- invert
- +1

29 = 0001 1101

invert = 1110 0010

+1 = 1110 0011

触发器



RAM : Random-Access Memory

1. SRAM static RAM

2. DRAM Dynamic RAM

PC 存记数器 (Program Counter)

工作流程

复位 1. if reset=1 , PC=0

加载 2. if load=1 , PC = address

存数 3. if inc=1 , PC = PC+1

else : continue

HACK

1. A指令（address Instruction）

@value

将value存到A寄存器

value为0~32767

功能：

1. 设置个数，@5 → A=5
2. 设置内存地址：@sum → A指向变量sum的地址
3. 配合C指令实现跳转，@Loop后跟跳转路径

2. C指令（Compute Instruction）

dest=comp;jump

计算comp的值，将结果存储到dest，并根据comp与0的比较结果决定是否跳转到A寄存器指定的地址

comp（计算）：必选

算术：1 !A. D-1. A+1

1. 加法操作

D+A：D=5 A=3 →8

D+M：D=5 A=100 RAM[100]=10 → 15

2. 递增操作

A+1：A=7 → 8

M+1；A=200 M[200]=50 →RAM[200]=201

3. 减法操作

D-A

D-M

逻辑：&A，D|A !D

1. 位取反

!A : 对A的每一位取反

!M : 对RAM[A]每一位取反

2. 位二操作

D&A : D=0b1010 A=0b1100 → 0b1000

D&M : D=0b1010 A=200 RAM[A]=0b1100 → 0b1100

3. 位或操作

D|A : D=0b1010 A=0b1100 → 0b1110

D|M : D=0b1010 A=300 RAM[A]=0b1100 → 0b1110

dest (目标) : 可选 · 指定结果存储位置

M: RAM[A]

D: D寄存器 数据

A: A寄存器 地址

MD 表示同时存入M和D

jump (跳转) : 可选

JGT greater than >0

JEQ equal ==0

JMP jump

JGE greater equal >=0

JLT less than <0

JNE not equal !=0

JLE less equal <=0

```
// 设置初始值
@R0
D=M
```

将A设为0
D=RAM[0]

```
// 示例1：使用@R0读取内存
@R0      // A指令：设置A=0（因为R0对应地址0）
```

```
@n
M=D         // n = RAM[0]
@i
M=1         // i = 1
@sum
M=0         // sum = 0

(LOOP)      // 循环开始
@i
D=M         // D = i
@n
D=D-M       // D = i - n
@STOP
D;JGT       // 如果 i > n, 跳转到STOP
@i
D=M         // D = i
@sum
M=D+M       // sum = sum + i
@i
M=M+1       // i = i + 1
@LOOP
0;JMP       // 无条件跳回LOOP

(STOP)      // 循环结束
@sum
D=M
@R1
M=D         // RAM[1] = sum

(END)
@END
0;JMP       // 无限循环（程序终止）
```

因为此时 $n = i - n > 0$

```
D=M         // C指令：将RAM[0]的值加载到D寄存器
@5          // A指令：设置A=5
M=D         // C指令：将D的值存储到RAM[5]

// 示例2：使用@数字作为常数
@10         // A指令：设置A=10
D=A         // C指令：将常数10加载到D寄存器（注意：A是值，不是地址）
@R1         // A指令：设置A=1（R1对应地址1）
M=D         // C指令：将D的值（10）存储到RAM[1]

// 示例3：混合使用
@R0         // A=0
D=M         // D=RAM[0]
@100        // A=100
M=D         // RAM[100]=D
```

```
@SCREEN
D=A
@addr
M=D         // addr = 屏幕基地址
@R0
```

```
D=M
@n
M=D         // n = 高度（从RAM[0]读取）
@i
M=0         // i = 0
(LOOP)
@i
D=M
@n
D=D-M
@END
D;JEQ       // 如果 i == n, 结束
@addr
A=M
M=-1        // 设置16像素为黑色（全1）
@i
M=M+1       // i++
@32
D=A
@addr
M=D+M       // addr += 32（下一行）
@LOOP
0;JMP
(END)
```

VM CODE

1. 压栈

push constant n 把常数n压入栈

栈：

eg. push constant 30

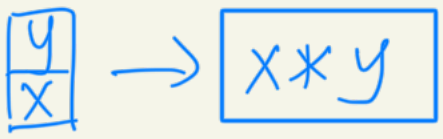
push constant 40

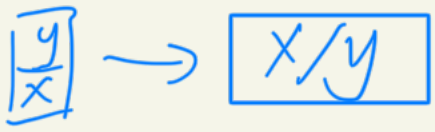
push argument n 把变量压入栈

用数字代替字母

2. 算术运算

add	+	弹出a,b	计算a+b	结果压栈
sub	−	弹出a,b	计算a-b	结果压栈
neg	取负	弹出栈顶x	计算-x	结果压栈
mult	*	弹出y,x	压入 x/y	
div	/	弹出y,x	压入 x/y	





先压入当分子

3. 逻辑运算

and 弹出a,b 计算a&b

57 and 28 = 24

57	0000000000111001
28	0000000000011100
and	0000000000011000 = 24

or 弹出ab 计算a\|b

24 or 82 = 90

24	0000000000011000
82	0000000001010010
or	0000000001011010 = 90

not 弹出栈顶x,计算~x 即按位取反,结果为-x-1

4. 比较

eq 弹出ab 若 a==b 压入-1 反之压入0

lt 弹出ab 若 a<b 压入-1 反之压入0

gt 弹出ab 若 a>b 压入-1 反之压入0

5. 弹出

pop xxx x 弹出xxx找x个

6. 跳转
if-goto 条件跳转
goto 无条件跳转

内存段	描述	示例
constant	常数（虚拟段，不占用内存）	`push constant 5`将5压栈
argument	函数参数段	`push argument 2`将第3个参数压栈
local	局部变量段	`push local 0`将第1个局部变量压栈
static	静态变量段（全局）	`push static 3`将静态变量3压栈
this	当前对象段	`push this 1`将this[1]压栈
that	第二个对象段	`push that 2`将that[2]压栈
pointer	指针段（this/that）	`push pointer 0`将this压栈
temp	临时变量段	`push temp 5`将temp[5]压栈

VM CODE 题型

1. 代码分析

自己一步一步看

2. 翻译表达式

eg. X = (a+b) * c 翻译成 VM 代码

```
push argument 1
push argument 2
add
push argument 3
mult
```

3. VM → HACK

- 内存地址计算正确
- 栈指针管理
- 条件跳转实现

eg. add

```
@SP
M=M-1
A=M
D=M
@SP
M=M-1
A=M
M=M+D
```

弹第二个

弹第一个

相加并压回

@SP　　栈指针调整
M=M+1

eg. push local i

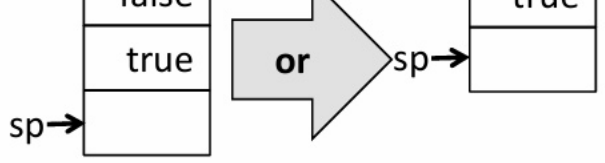
@i
D=A
@LCL
A=D+M
D=M

@SP
A=M
M=D

@SP
M=M+1

4、画图：

or



Note: Boolean values,
use **true** or **false**.
Do not use 1 or 0.