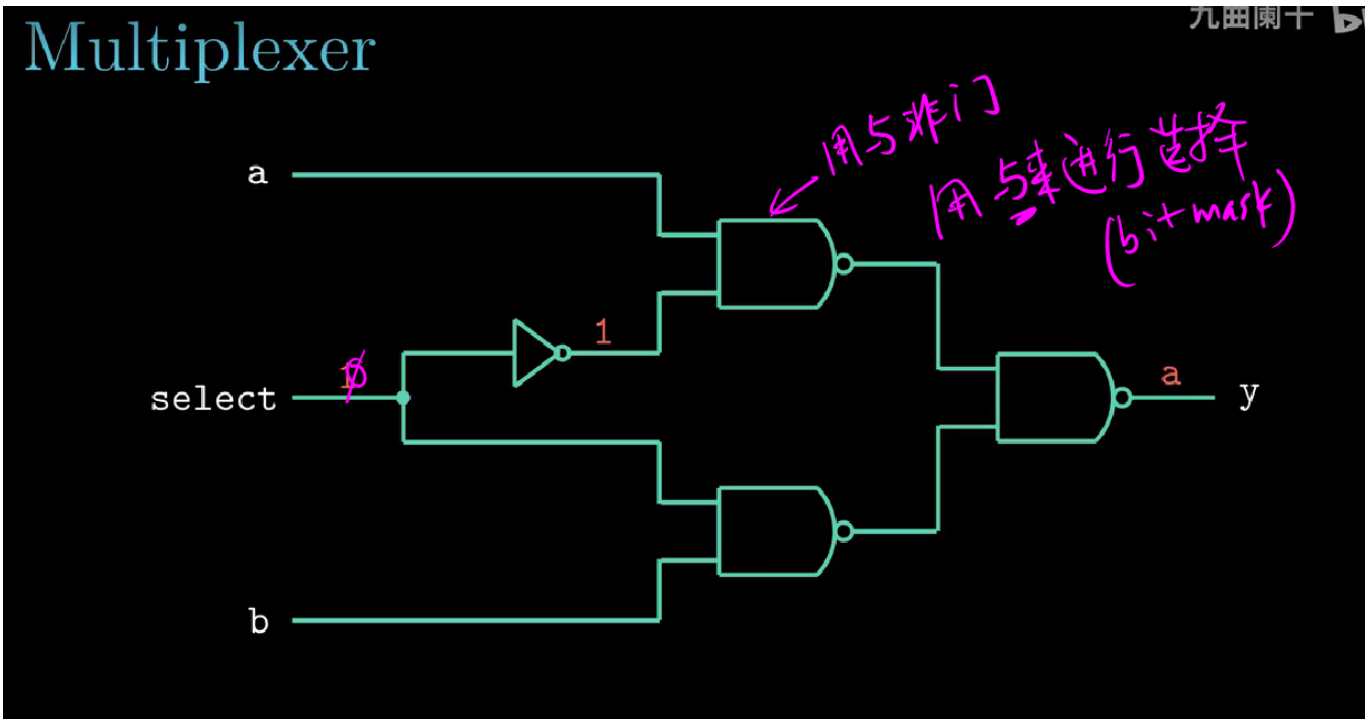
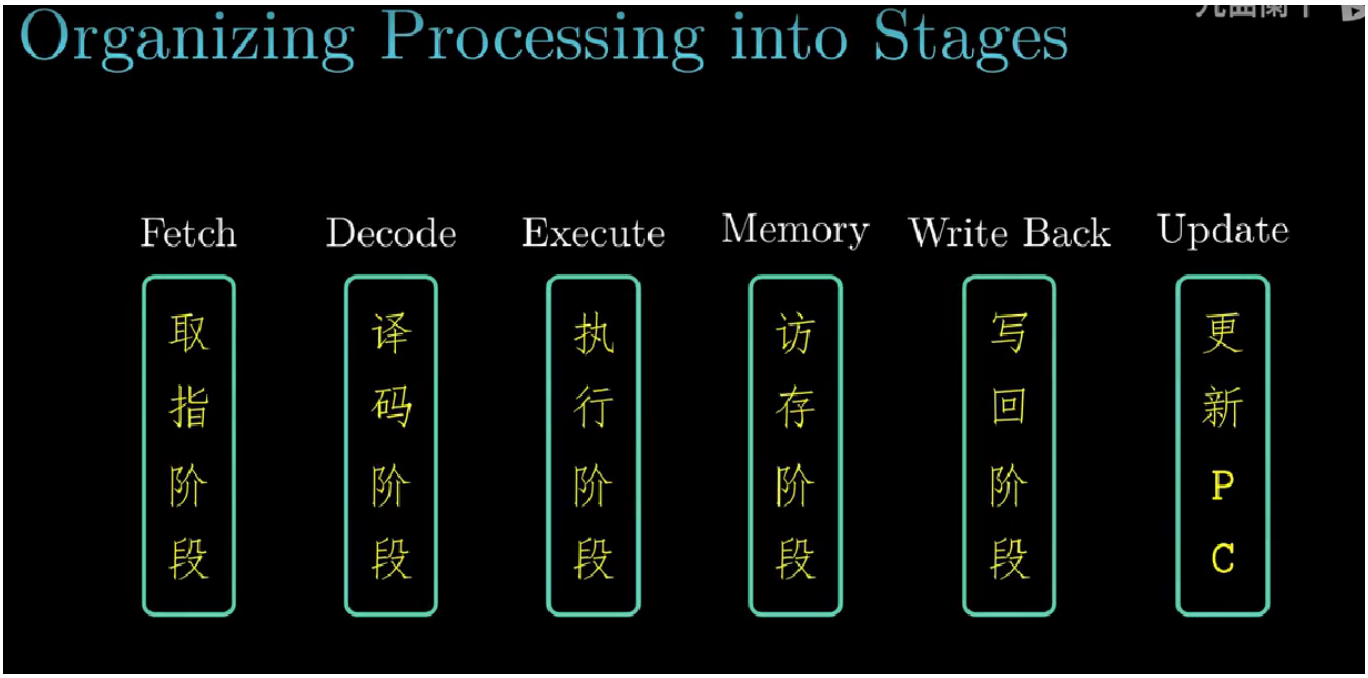
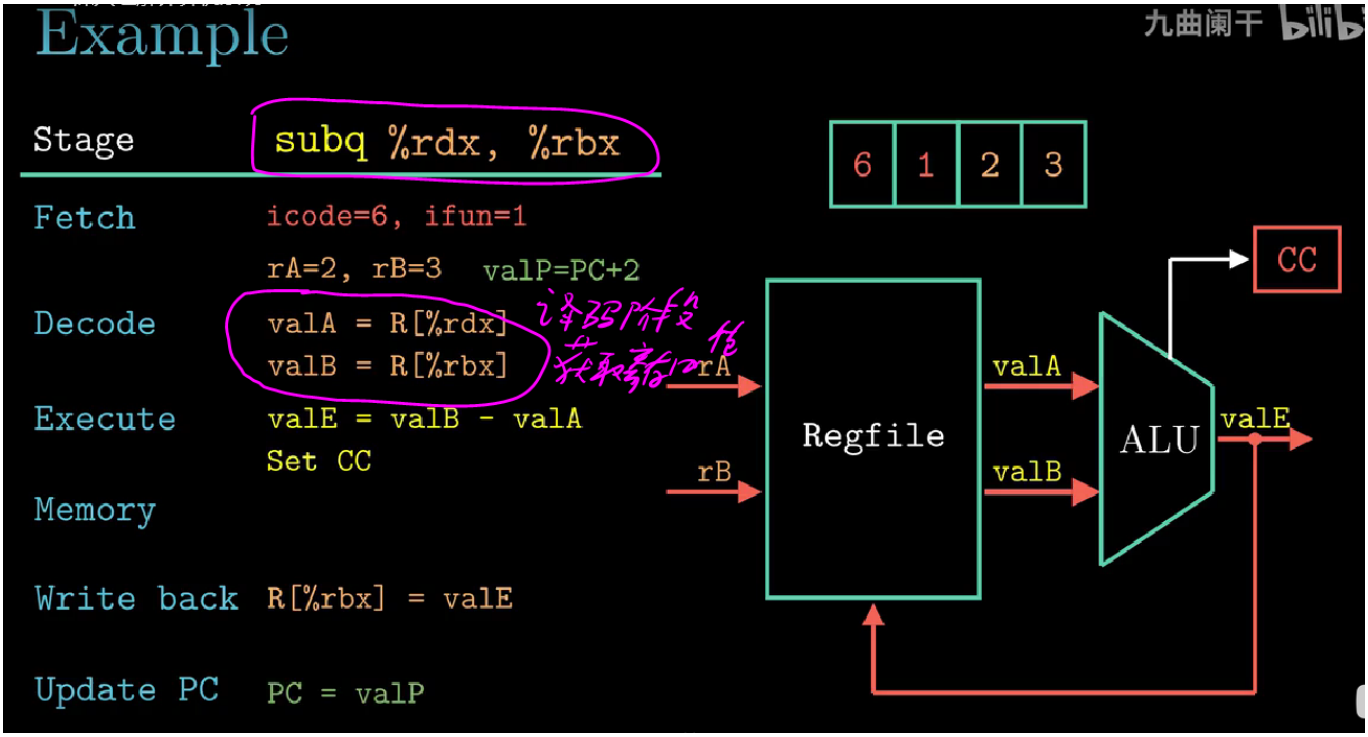
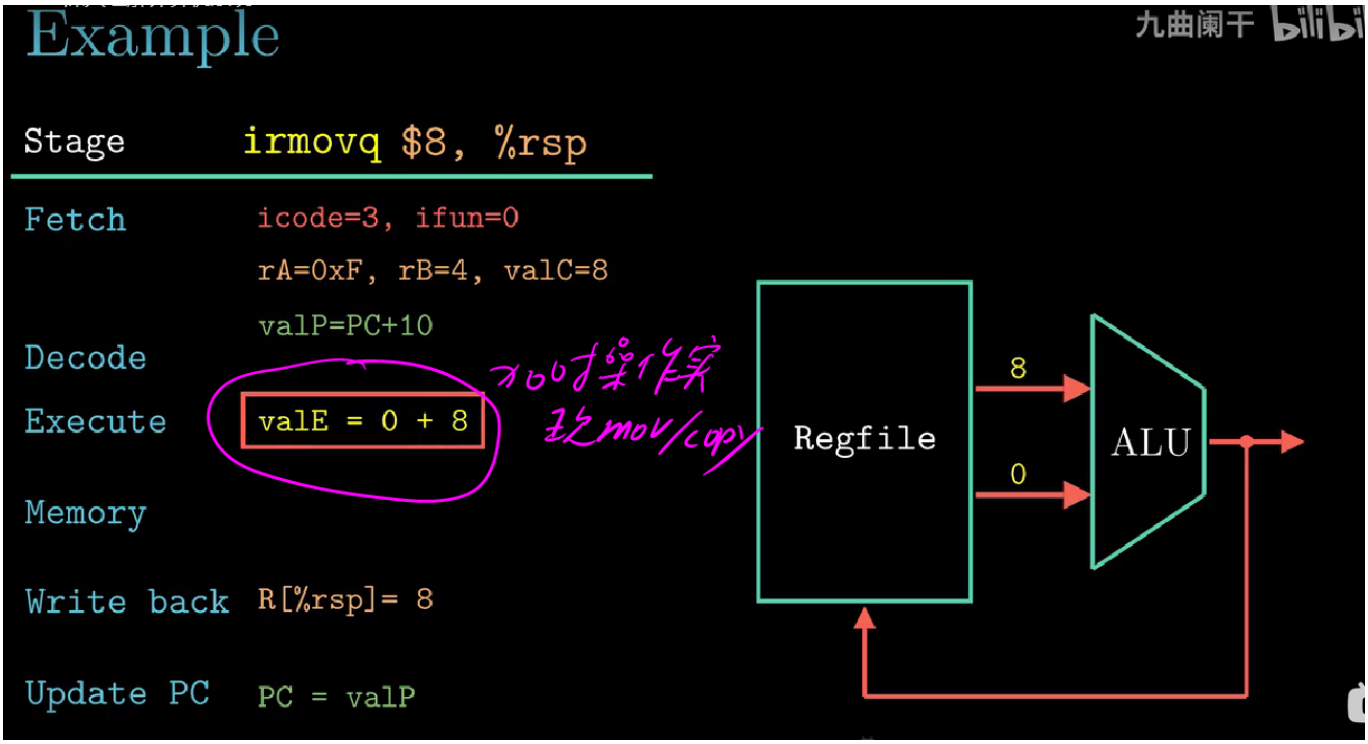
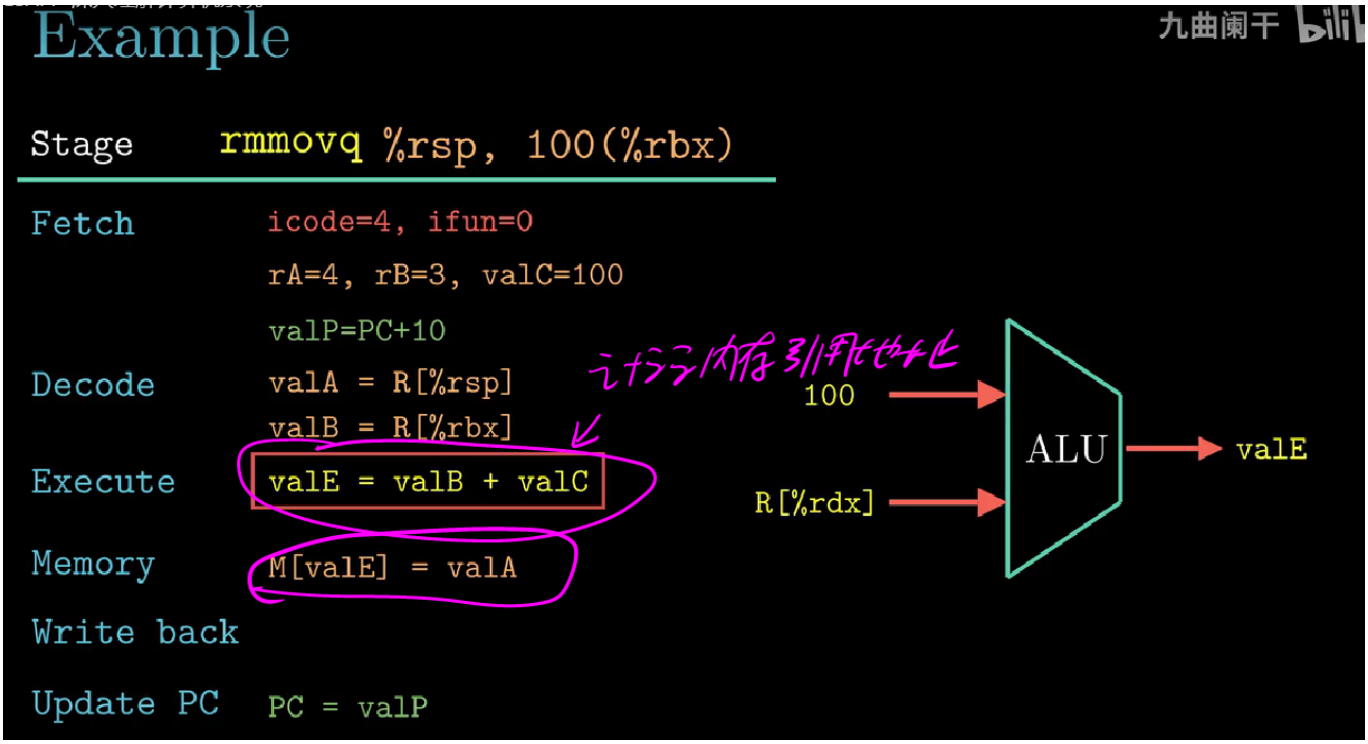

Multiplexer
a
select
b
1
a
y
用5非门
用5来进行选择
(bit mask)


Organizing Processing into Stages
Fetch
Decode
Execute
Memory
Write Back
Update
取指阶段
译码阶段
执行阶段
访存阶段
写回阶段
更新PC


Example
九曲阑干
Stage
subq %rdx, %rbx
Fetch
icode=6, ifun=1
rA=2, rB=3 valP=PC+2
Decode
valA = R[%rdx]
valB = R[%rbx]
Execute
valE = valB - valA
Set CC
Memory
Write back R[%rbx] = valE
Update PC PC = valP
6 1 2 3
rA
rB
Regfile
valA
valB
ALU
valE
CC

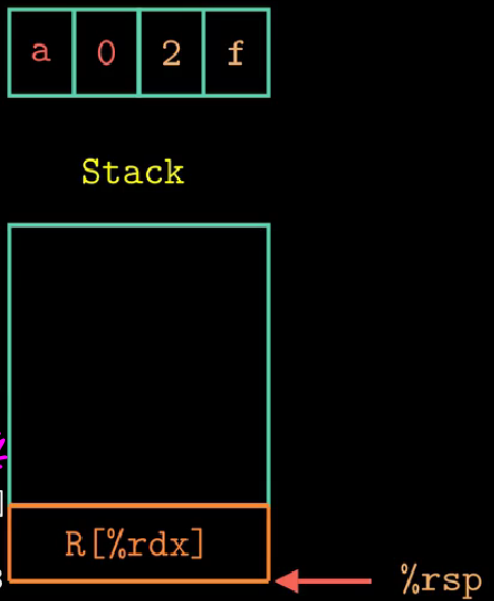

Example
九曲阑干
Stage irmovq $8, %rsp
Fetch icode=3, ifun=0
rA=0xF, rB=4, valC=8
valP=PC+10
Decode
Execute valE = 0 + 8
Memory
Write back R[%rsp]= 8
Update PC PC = valP
Regfile
8
0
ALU


Example
九曲阑干
Stage rmmovq %rsp, 100(%rbx)
Fetch icode=4, ifun=0
rA=4, rB=3, valC=100
valP=PC+10
Decode valA = R[%rsp]
valB = R[%rbx]
Execute valE = valB + valC
Memory M[valE] = valA
Write back
Update PC PC = valP
100
R[%rdx]
ALU
valE

Example

九曲阑干

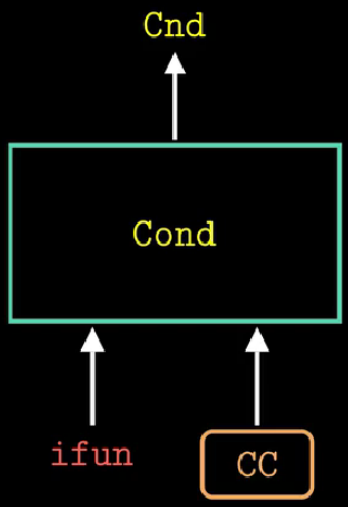
Stage	pushq %rdx
Fetch	icode=a, ifun=0 rA=2, rB=0xF valP=PC+2
Decode	valA = R[%rdx] valB = R[%rsp]
Execute	valE = valB +(-8)
Memory	M[valE] = valA
Write back	R[%rsp] = valE
Update PC	PC = valP

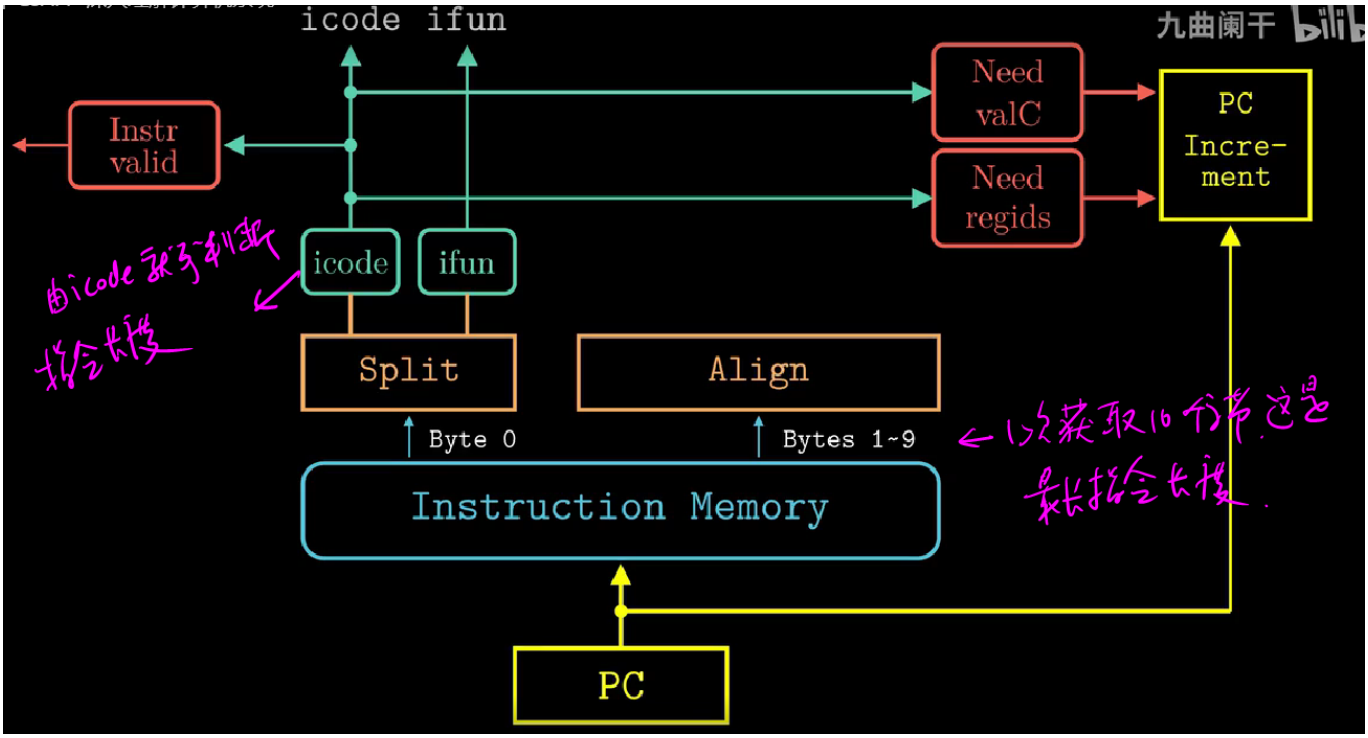


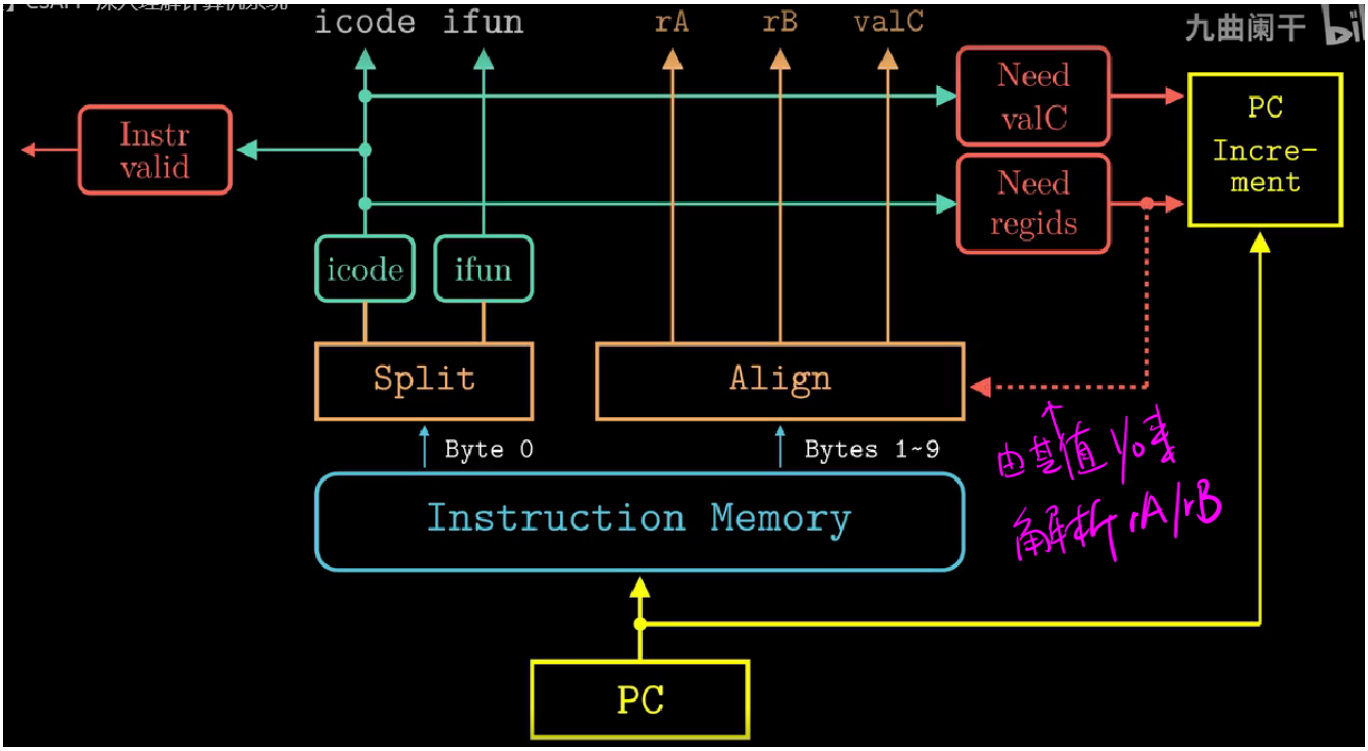
Example

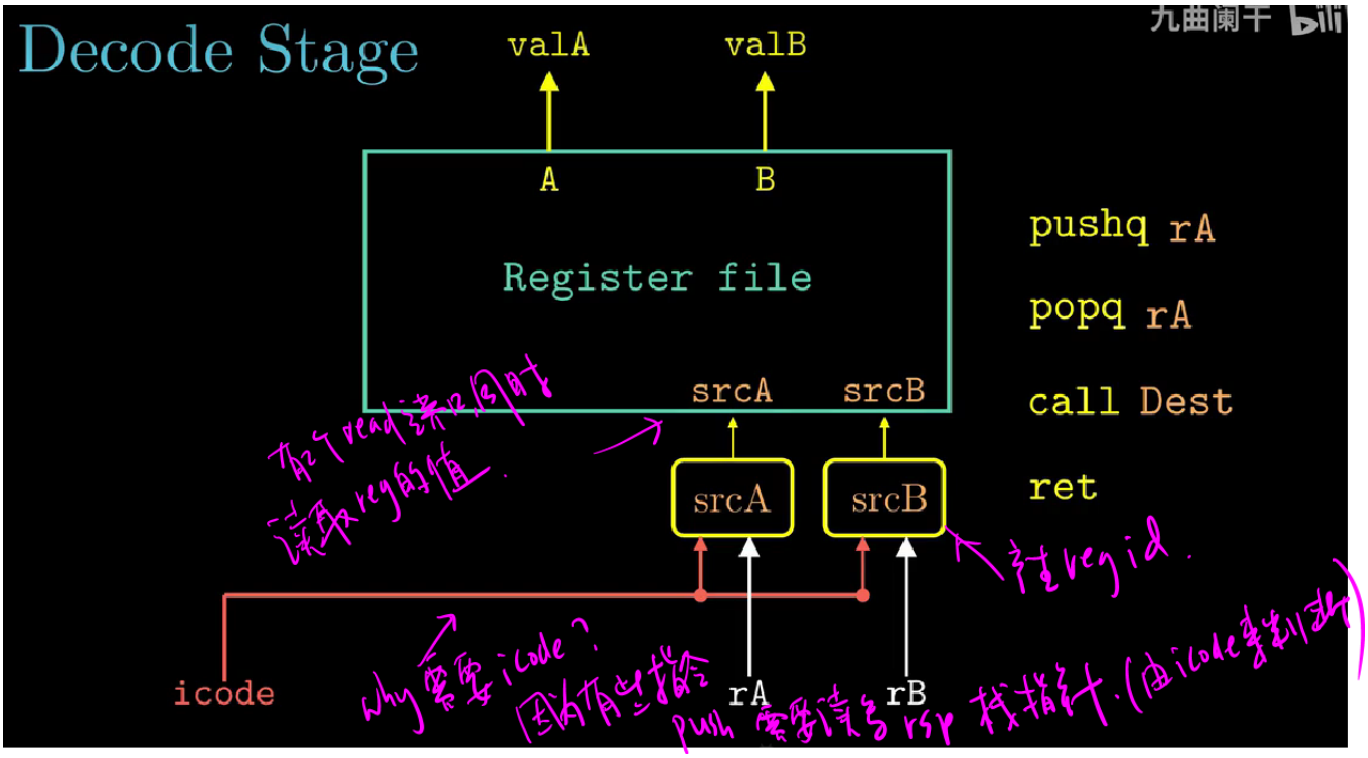
九曲阑干

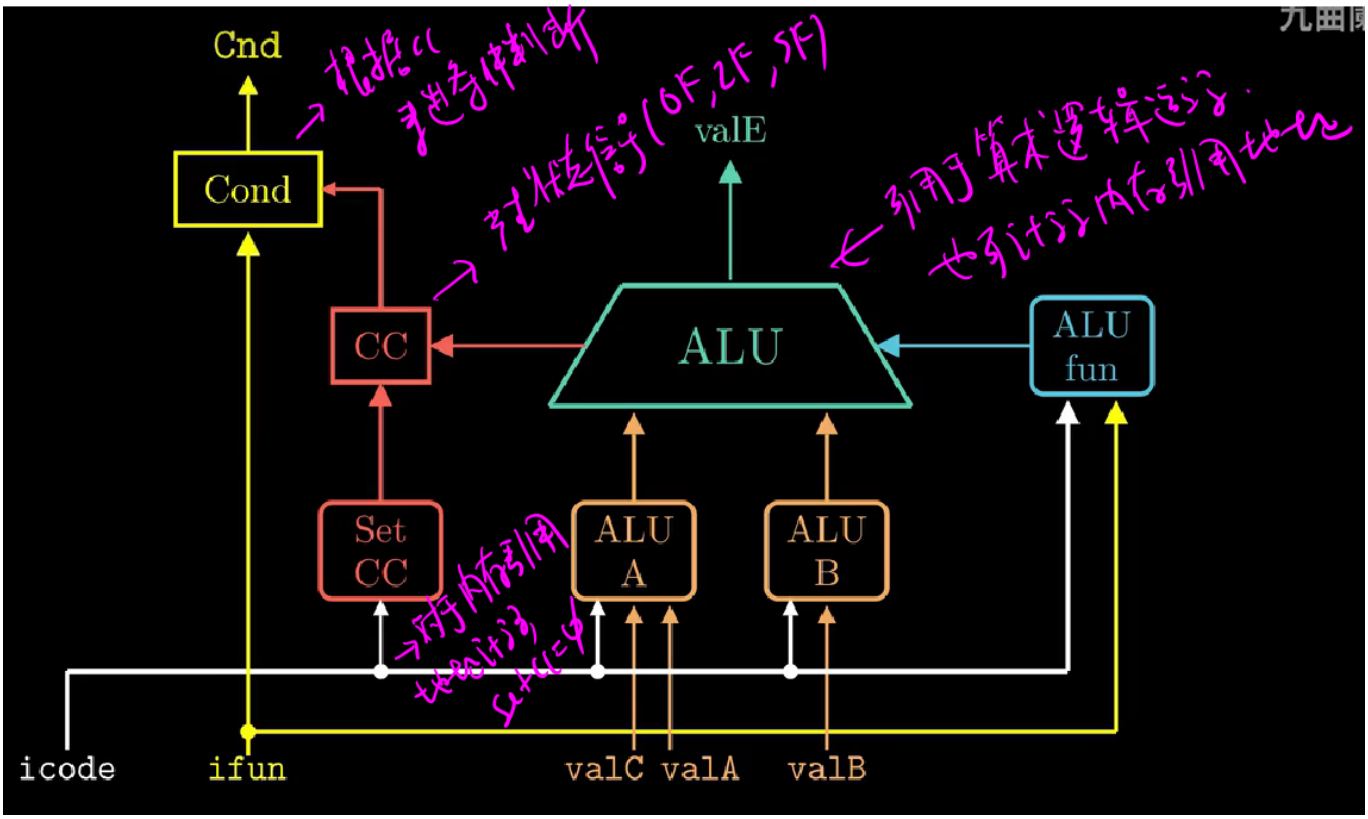
Stage	je 0x040
Fetch	icode=7, ifun=3 valC = 0x040 valP=PC+9
Decode	
Execute	cnd = Cond(CC, ifun)
Memory	
Write back	
Update PC	PC = Cnd ? 0x040 : valP

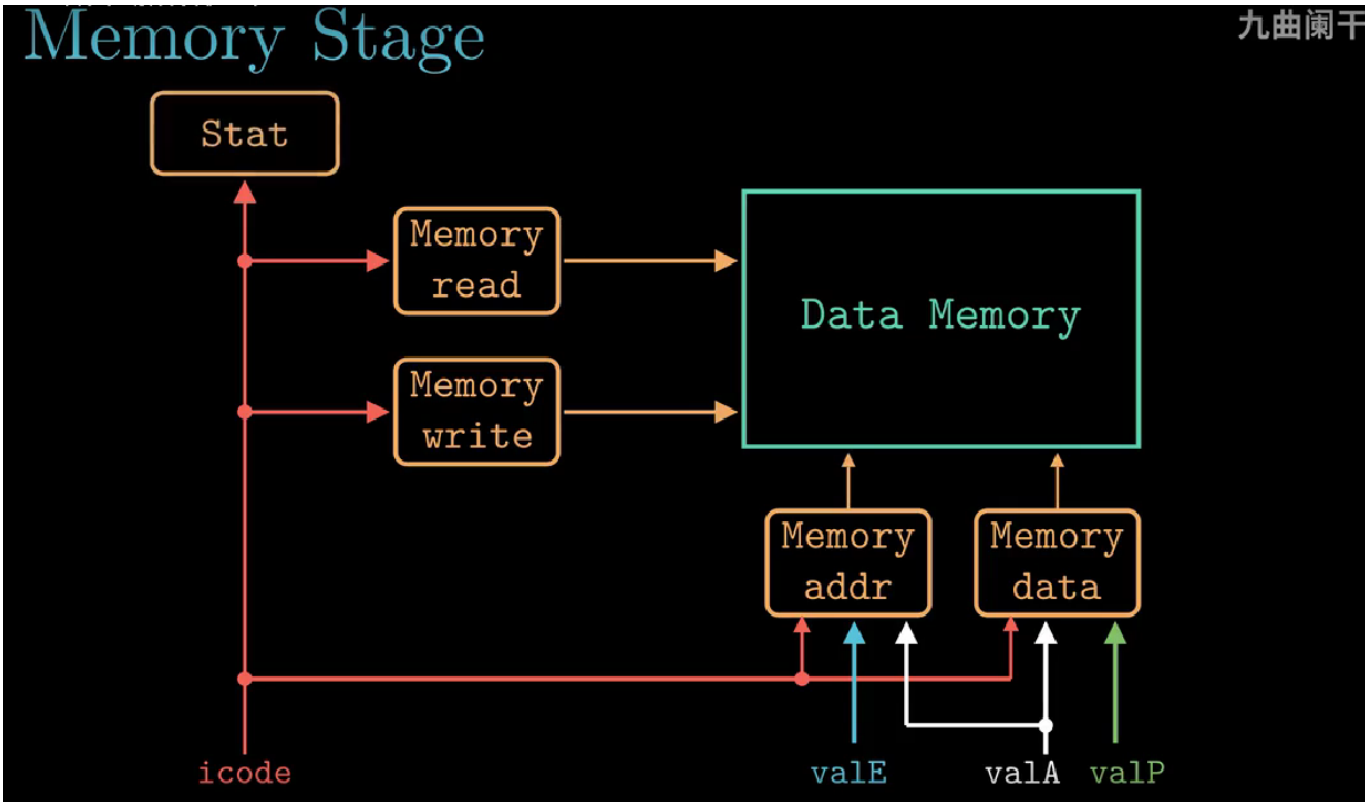


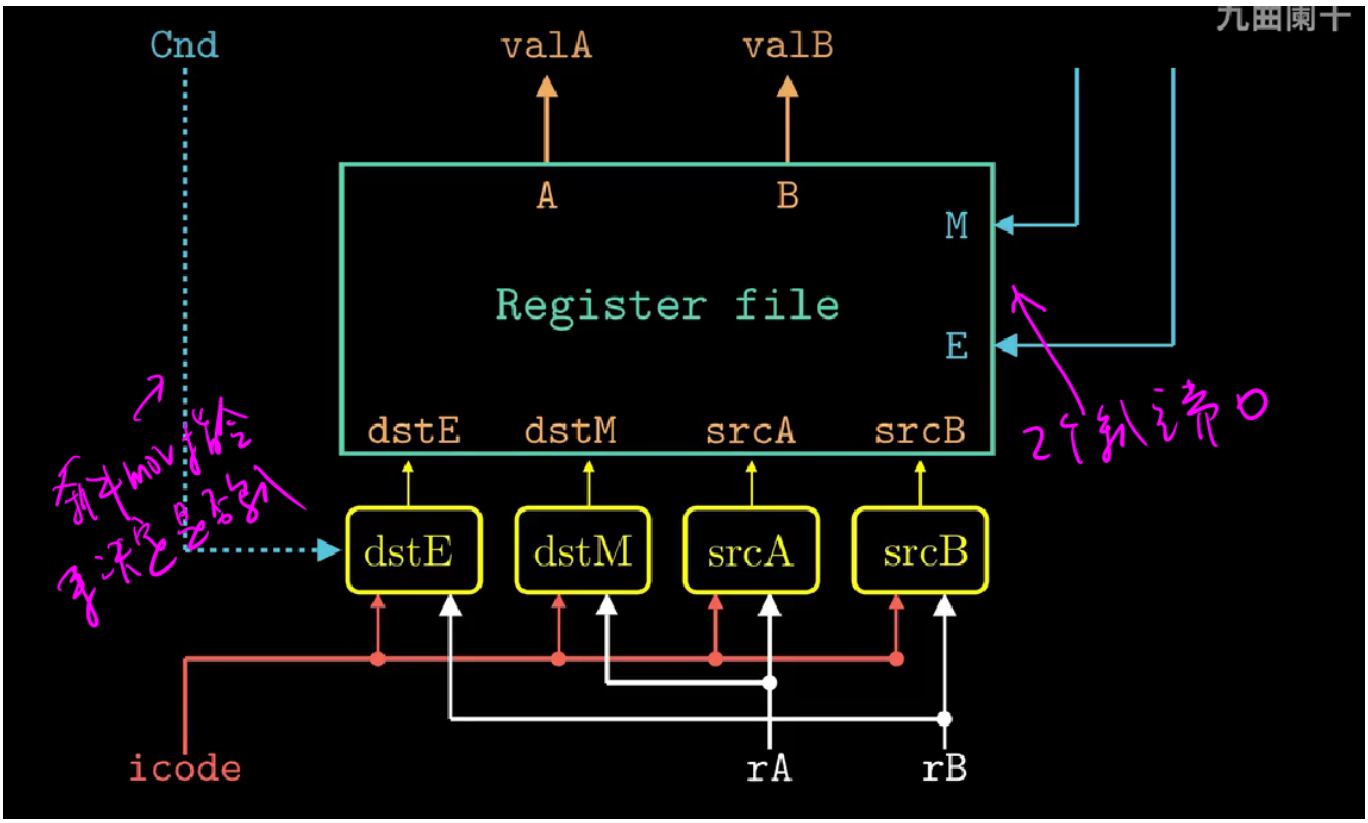

icode ifun
九曲阑干
Instr valid
Need valC
Need regids
PC Incre-ment
icode
ifun
Split
Align
Byte 0
Bytes 1~9
Instruction Memory
PC

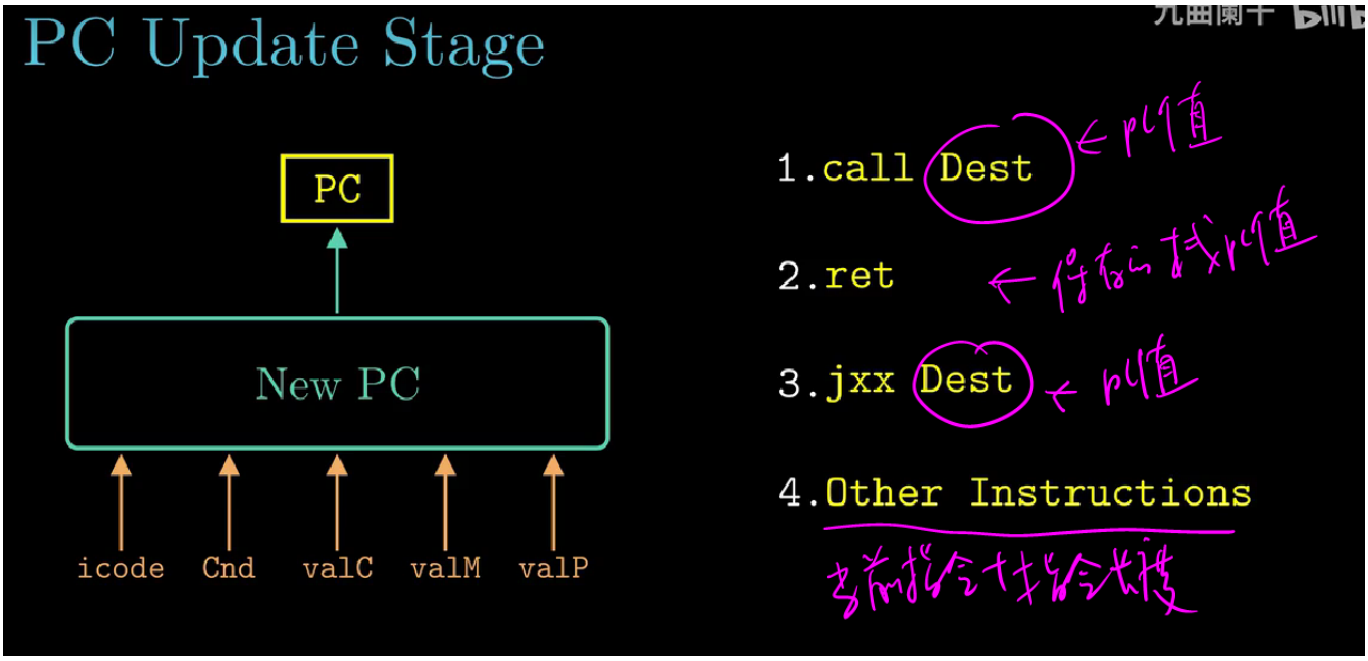

icode ifun
rA
rB
valC
九曲阑干
Instr valid
Need valC
Need regids
PC Incre-ment
icode
ifun
Split
Align
Byte 0
Bytes 1~9
Instruction Memory
PC

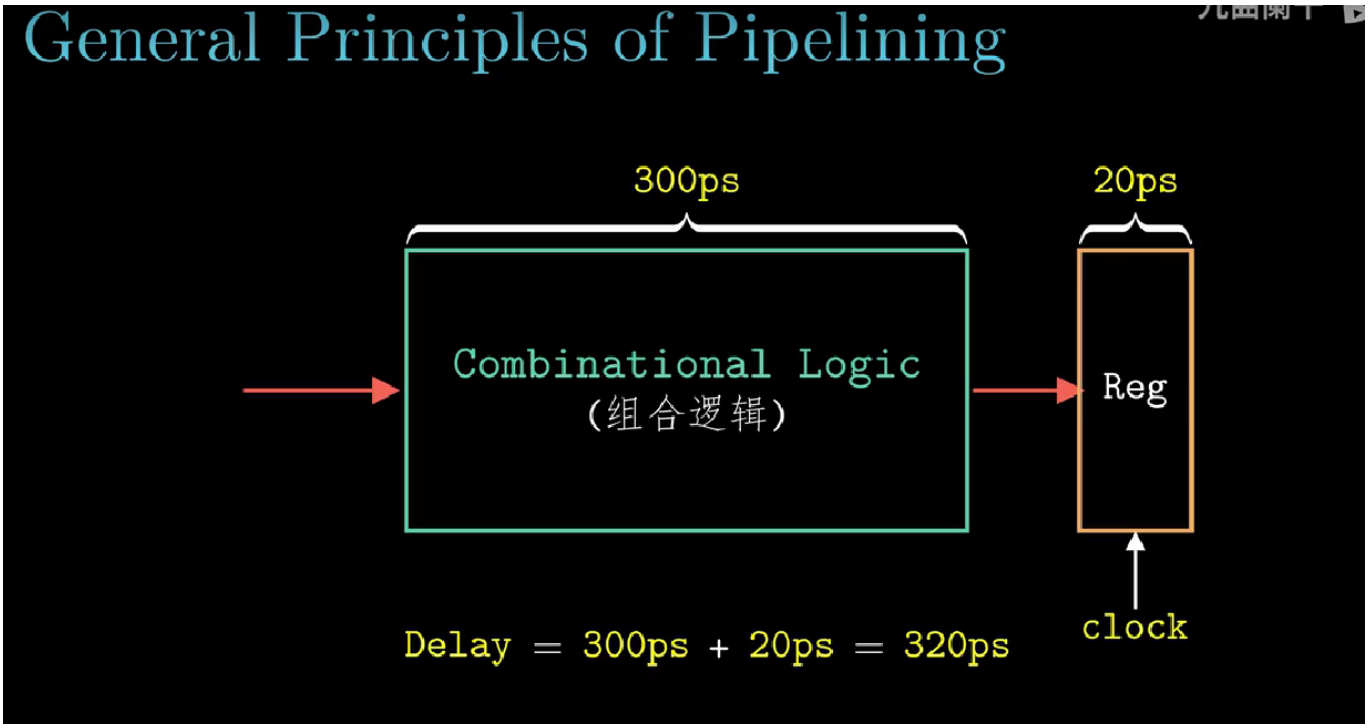

Decode Stage
valA
valB
A
B
Register file
srcA
srcB
srcA
srcB
icode
rA
rB
pushq rA
popq rA
call Dest
ret


Cnd
Cond
valE
CC
ALU
ALU fun
Set CC
ALU A
ALU B
icode
ifun
valC
valA
valB

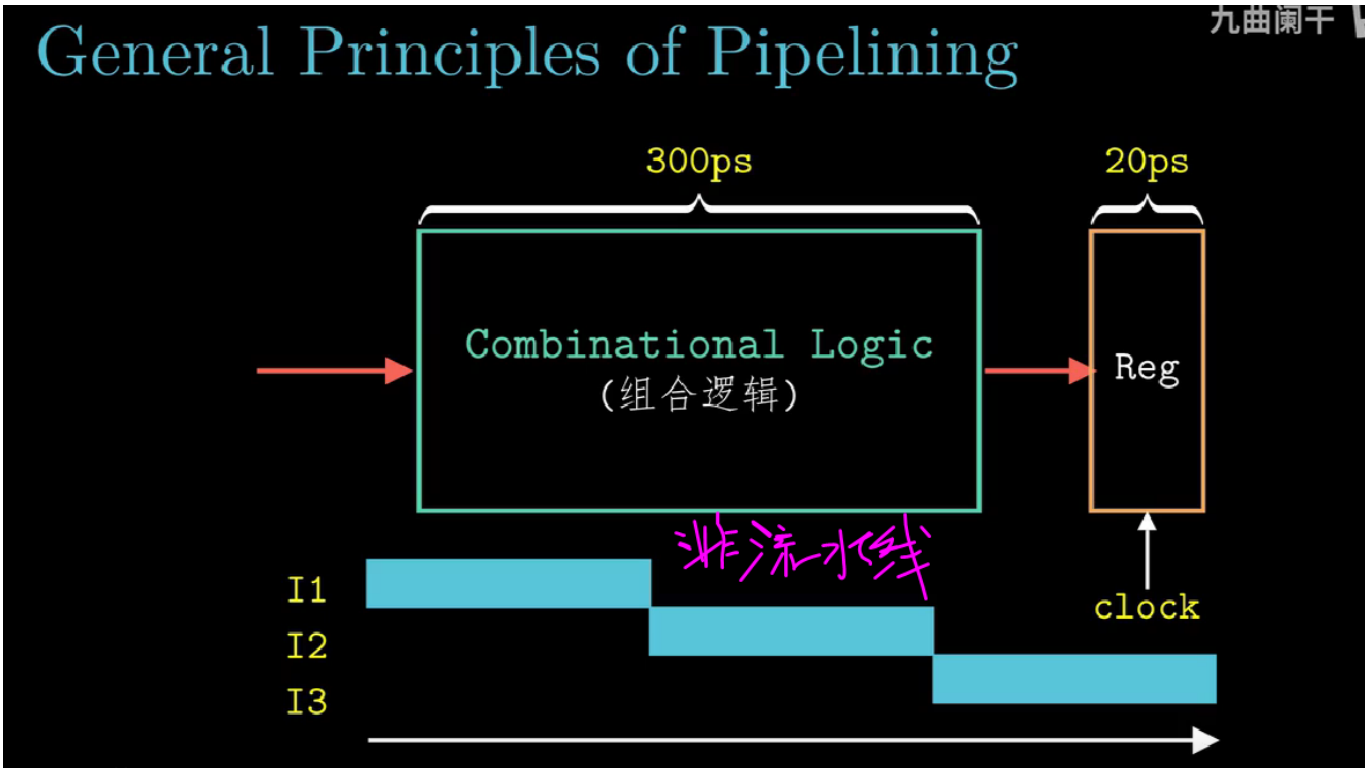

Memory Stage
九曲阑干
Stat
Memory read
Memory write
Data Memory
Memory addr
Memory data
icode
valE
valA
valP

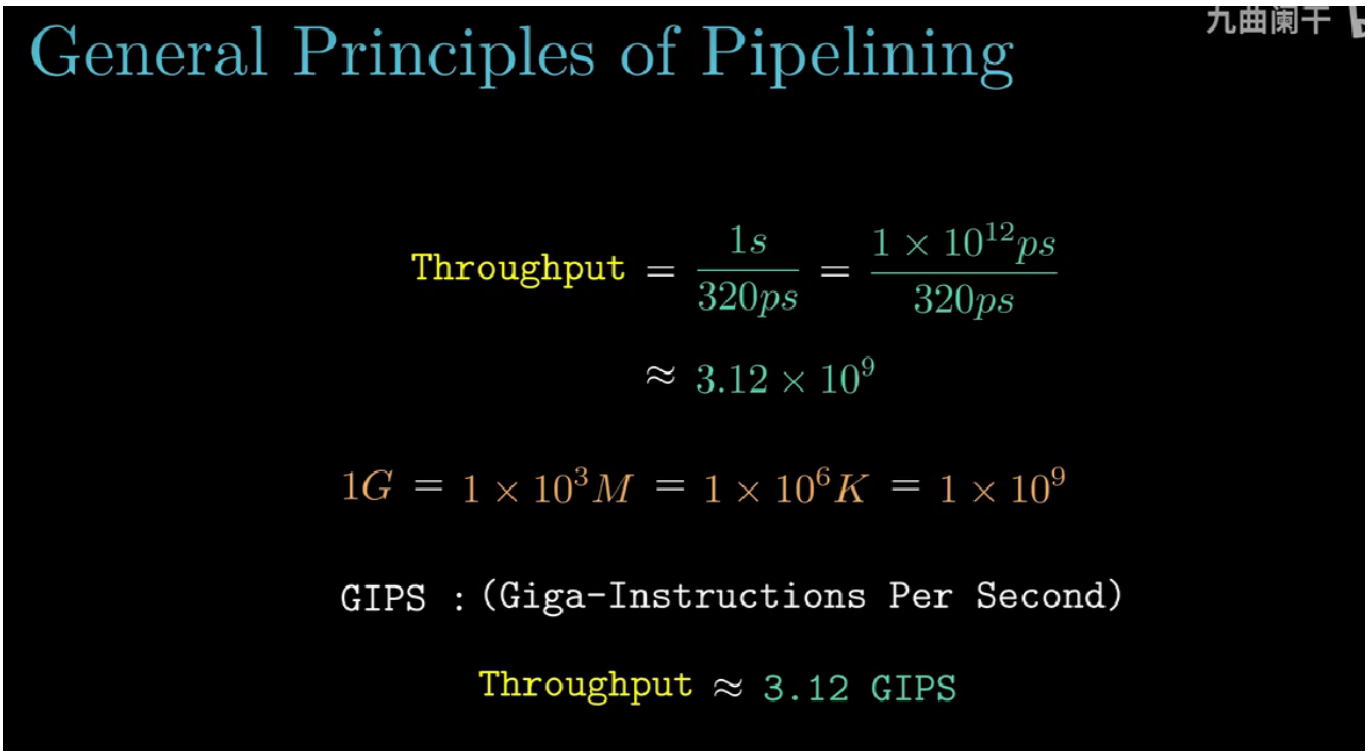

九曲阑干
Cnd
valA
valB
A
B
M
Register file
E
dstE
dstM
srcA
srcB
dstE
dstM
srcA
srcB
icode
rA
rB

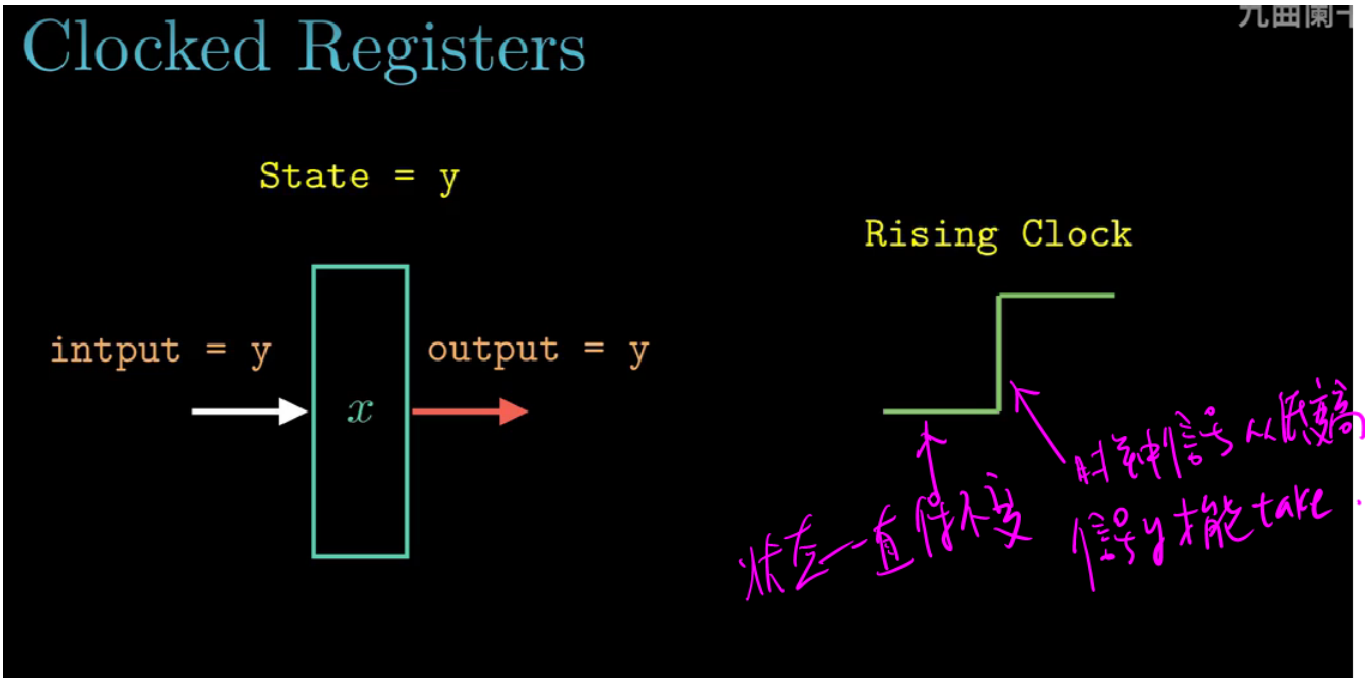


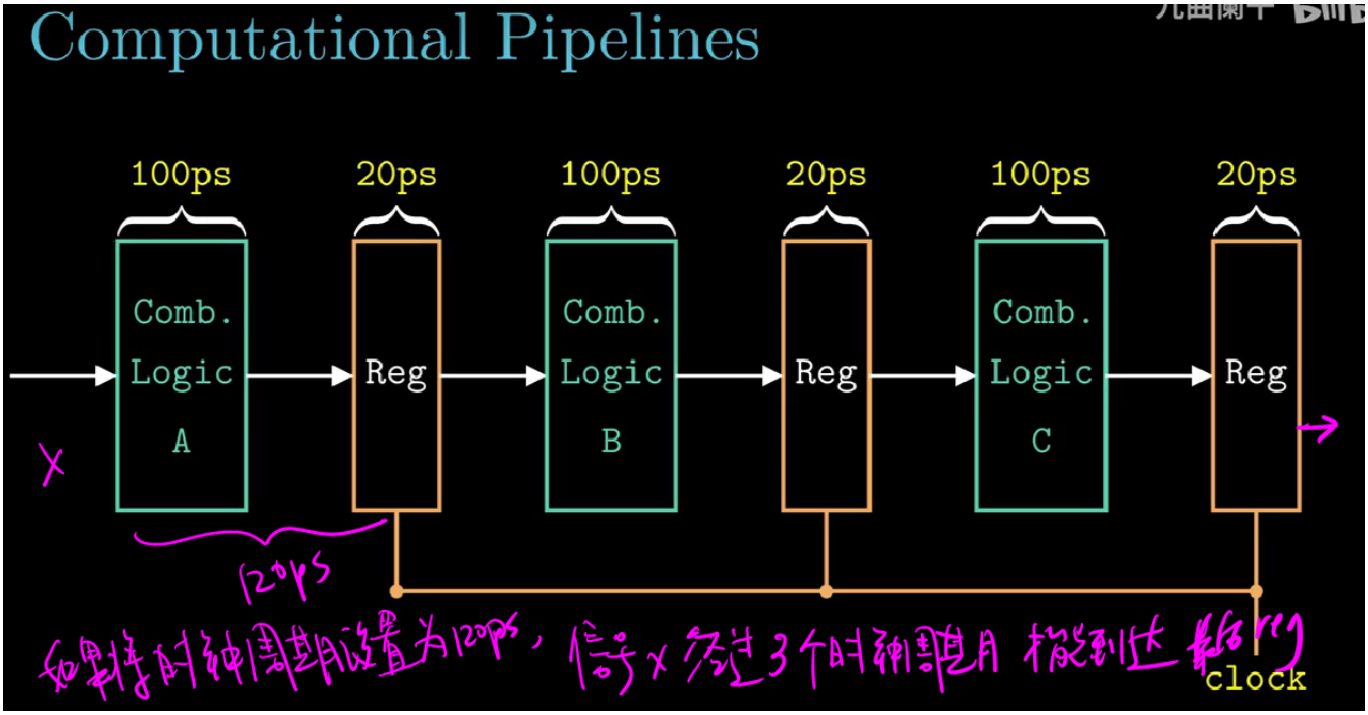


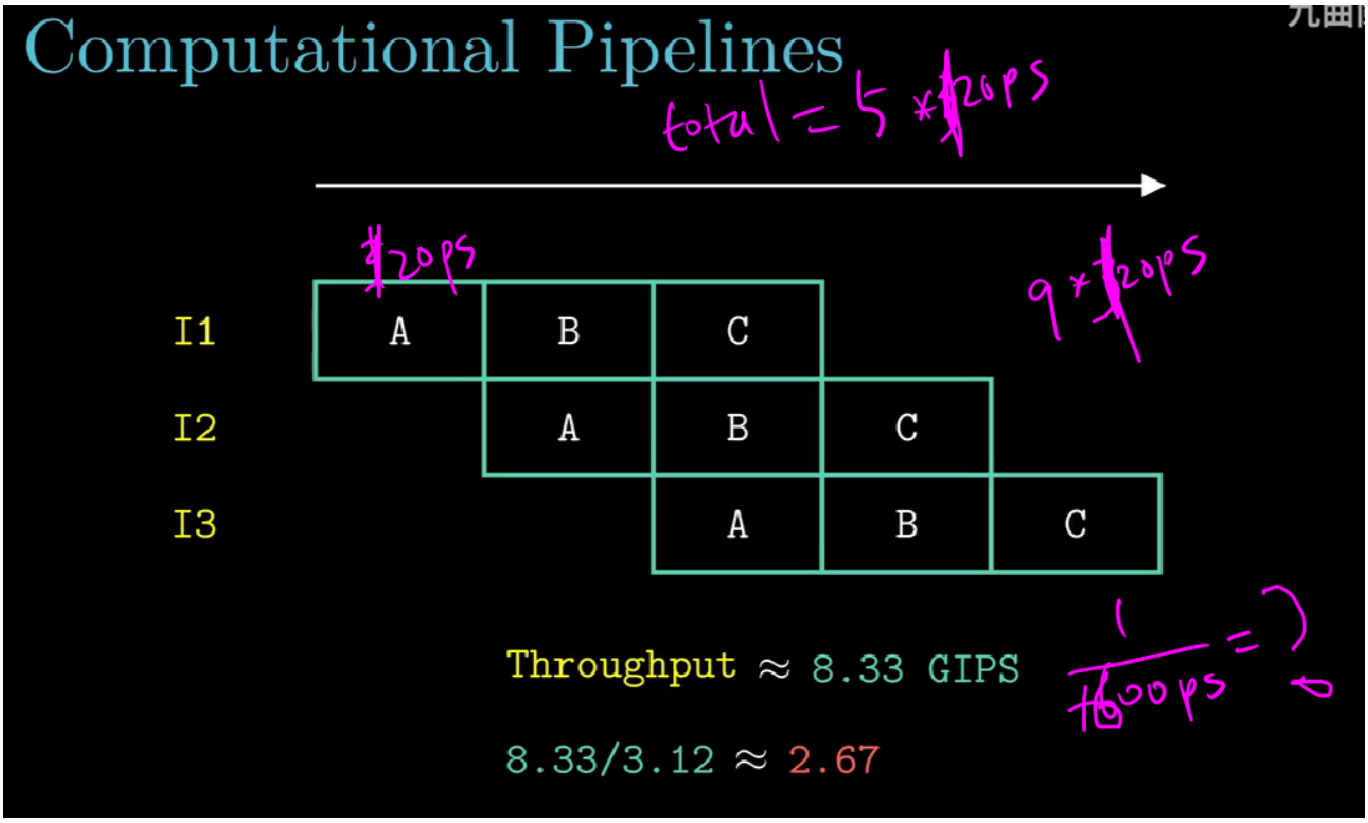
整个指令的执行时间为320ps (皮秒)

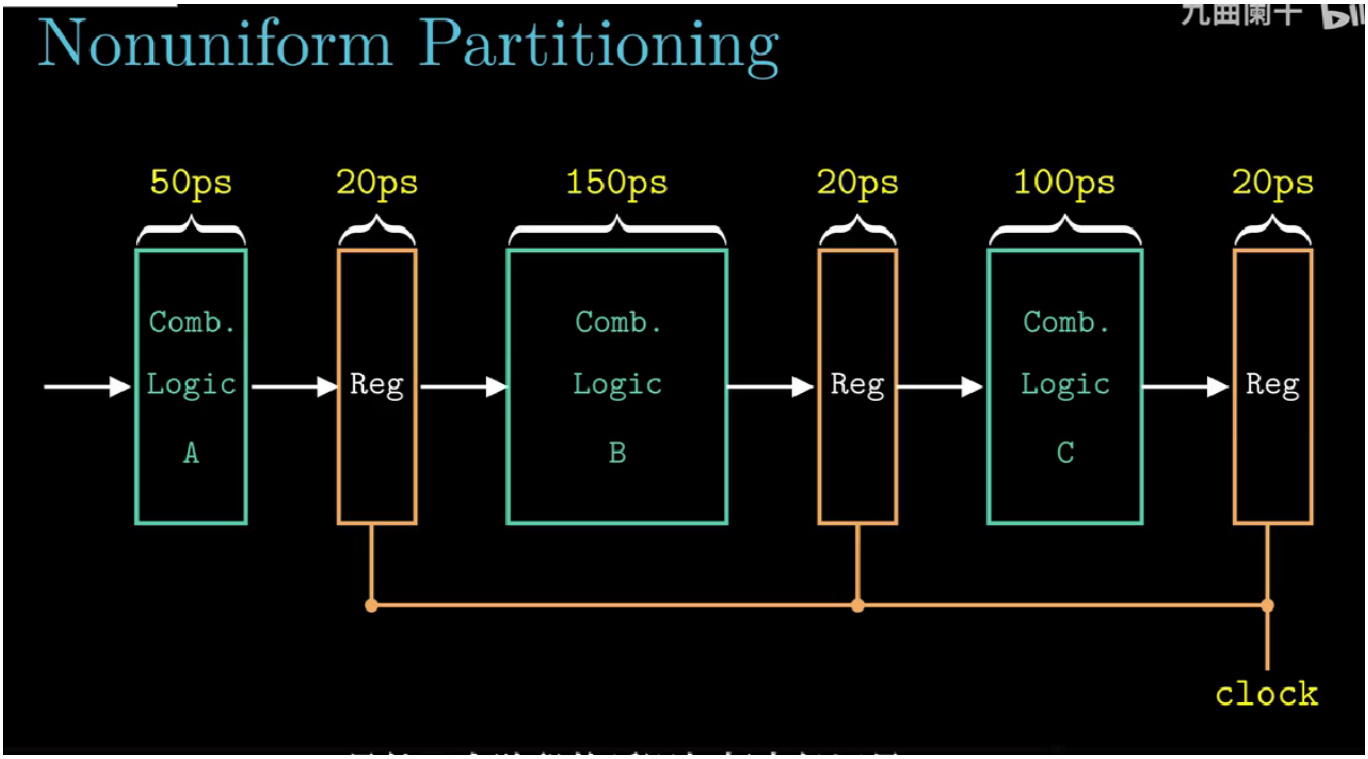

General Principles of Pipelining
九曲阑干
300ps
20ps
Combinational Logic
(组合逻辑)
Reg
非流水线
I1
I2
I3
clock

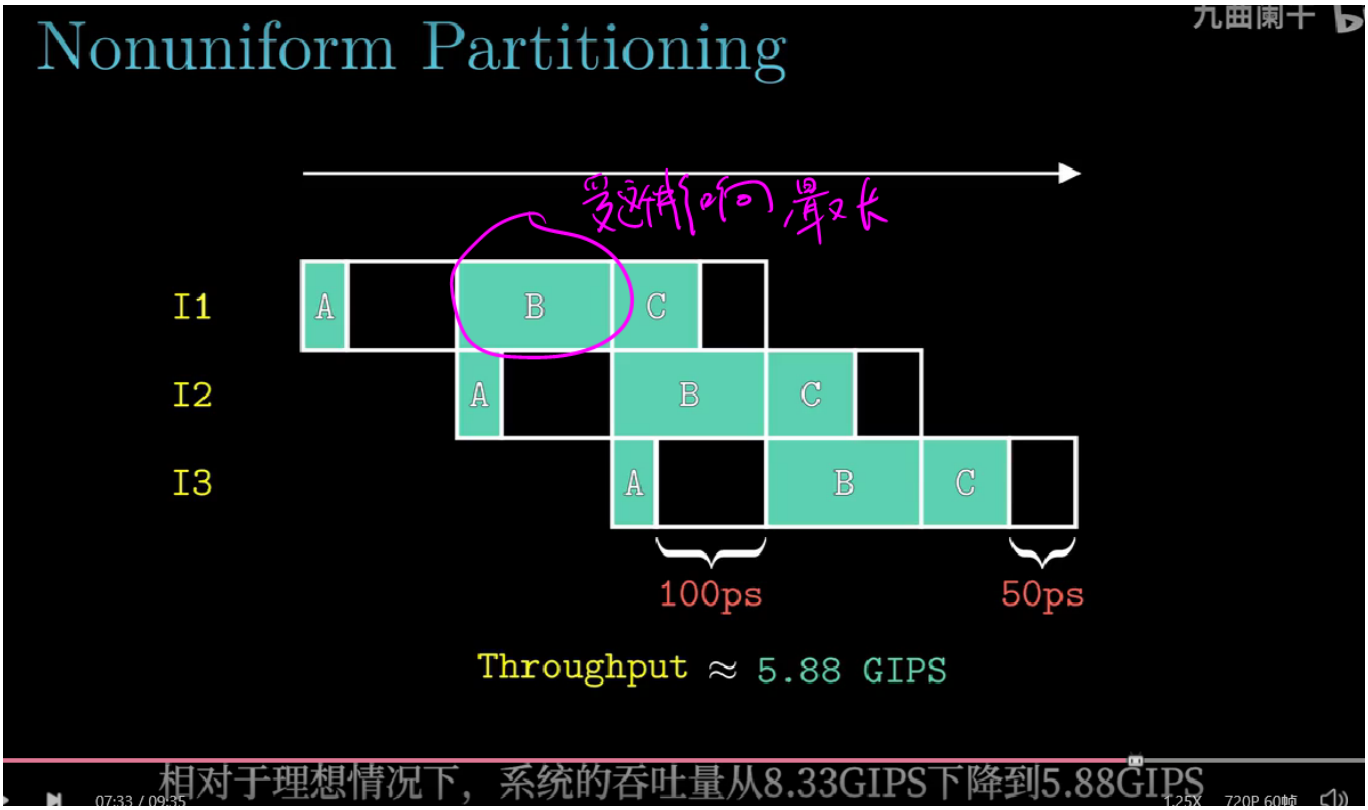

General Principles of Pipelining
九曲阑干
Throughput $= \frac{1s}{320ps} = \frac{1 \times 10^{12} ps}{320ps}$
$\approx 3.12 \times 10^9$
$1G = 1 \times 10^3 M = 1 \times 10^6 K = 1 \times 10^9$
GIPS : (Giga-Instructions Per Second)
Throughput ≈ 3.12 GIPS


Clocked Registers
State = y
intput = y
x
output = y
Rising Clock


Computational Pipelines
100ps
20ps
100ps
20ps
100ps
20ps
Comb. Logic A
Reg
Comb. Logic B
Reg
Comb. Logic C
Reg
X
120ps
clock


Computational Pipelines
I1
I2
I3
A
B
C
Throughput ≈ 8.33 GIPS
8.33/3.12 ≈ 2.67


Nonuniform Partitioning
50ps
20ps
150ps
20ps
100ps
20ps
Comb. Logic A
Reg
Comb. Logic B
Reg
Comb. Logic C
Reg
clock


Nonuniform Partitioning
I1
I2
I3
A
B
C
100ps
50ps
Throughput ≈ 5.88 GIPS
相对于理想情况下，系统的吞吐量从8.33GIPS下降到5.88GIPS