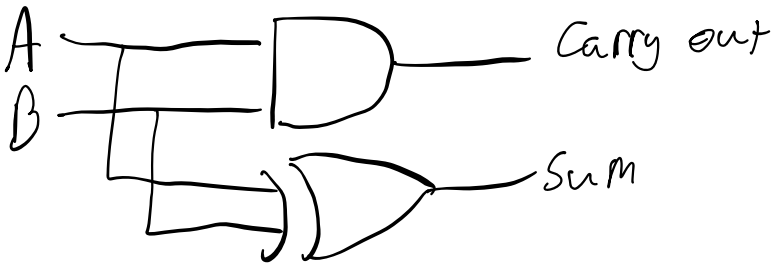
Digital logic design

Half adder



Truth Table

A	B	carryout	sum
		A and B	A xor B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Carryout = A • B

MyHDL

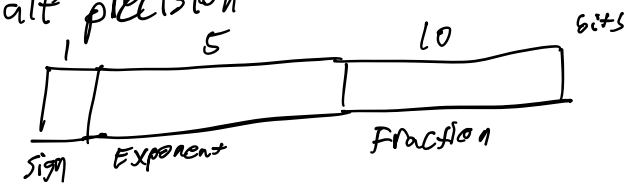
```
Carryout.next = ANDgate(z, A, B)
sum.next = Xorgate(z, A, B)
```

Binary multiplication

```
0 0 0 0 [0 1]
        0 1 0 1
-----------------
      . 1 0 1 1
    0 0 0 0 0 0
  1 0 1 1 0 0
+ 0 0 0 0 0 0 0
-----------------
  0 1 1 0 0 1 1
```

Question: why do we not need to load lower bounds but we have to load upper bound for MExtensions, Mul, Muic...?

IEEE Half precision



-12.5

↓

Sign = Negative = 1

12 = 1100

.5 = .1

1100.1 move up to first 1

3 2 1

1.1001×2^3

Bias = 15 3+15 = 18 = 10010

Sign: 1

Exponent: 10010

Fraction: 1001000000

1 10010 1001000000 Convert to Hex = CA40

0x8888 to Half precision

0x8888 = 1000100010001000

1000100010001000

1 = Negative

00010 = 2-15 = -13

0001001

$11.001001 \cdot 2^{-13}$

Write RISC V code

```
af[i] = af[i] * 9.0 # af[i] is float and s2 i = s3
addi t0,zero,9
fcvt.s.w ft0,t0 # ft0 = 9.0
slli t1,s3,2
add t1,t1,s2
flw ft1,0(t1)
fmul.s ft1,ft1,ft0
fsw ft1,0(t1)
```

performence

PC	I mem	Add	ALU	D-mem	control
40ns	30ns	20ns	60ns	50ns	20ns

1. Min cpu time

2. Min something

3. clock rate

4. ?

Average cpI

cpi 4 40%
cpi 5 5%
cpi 4 5%
cpi 3 30%
cpi 2 20%

$4 \cdot .4 + 5 \cdot .05 + 4 \cdot .05 + 3 \cdot .3 + 2 \cdot .2$

3.35

who is faster, X or Y?

X: CPI: 1 cpu time: 270

Y: CPI: 1.5 cpu time: 400

Amdahls law

How to make overall speed 3X faster?

Answer: 80% why? IDK

pipelining

what is ALUSrc for Add? 0

what is MemtoReg for lw? 1

If ALUSrc is stuck at 0 what commands wouldn't work? lw, sw

what commands don't care about MemtoReg? sw, beq

How to do this? use table:

Generating control signals from opcode

Inst.	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op
R-type	0	0	1	0	0	0	10
lw	1	1	1	1	0	0	00
sw	1	X	0	0	1	0	00
beq	0	X	0	0	0	1	01

Inst.	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	J	JR
JAL	X	0	1	0	0	0	1	0
JALR	X	0	1	0	0	0	1	1

Given code:

lw xc, 0(sp)

sub ...

And ...

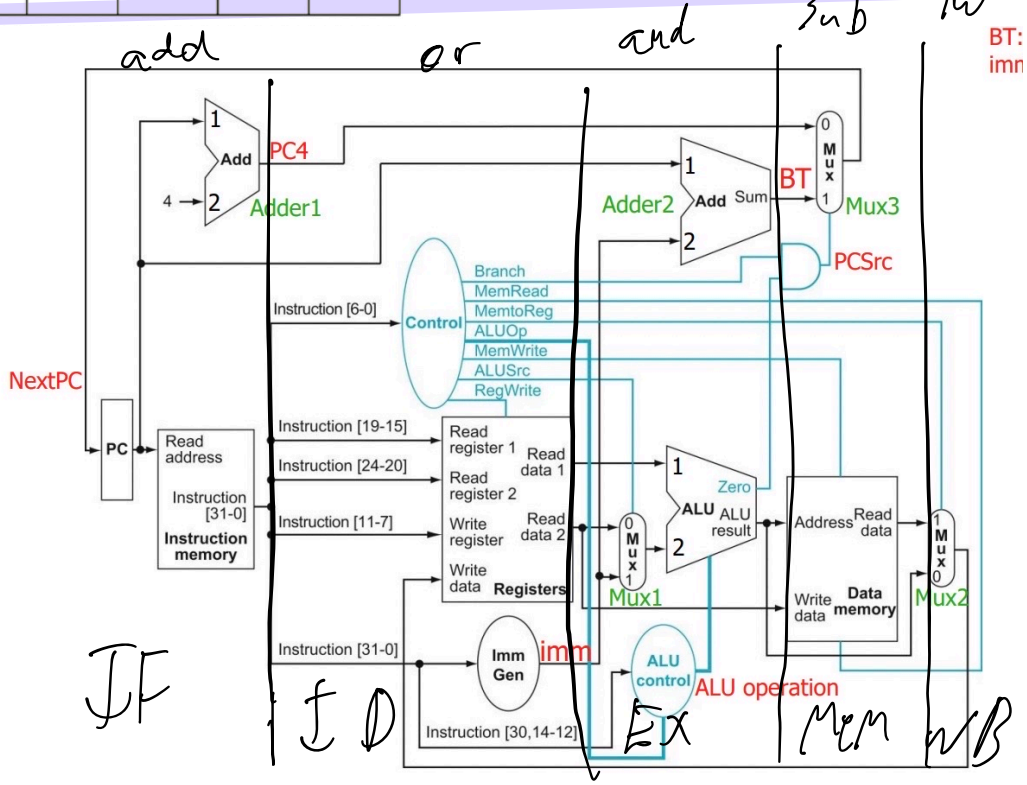
Or ...

add ...

Where is and? EX

where is lw storing its write? X

...



Non forwarding Implementation

```
lw x1, 0(x2)
add x3, x1, x2
sw x3, 0(x2)
```

Forwarding Implementation