① Architecture	Organization
Software	Hardware
eg Instruction set	eg Memory Technology
Visible to programmer	How features are Implemented
Logic	Physical components
High level	Low level

(2) Von Neuman Model

(1) Main Memory, ALU, Control Unit

(2) Registers in Control Unit

(i) P.C.

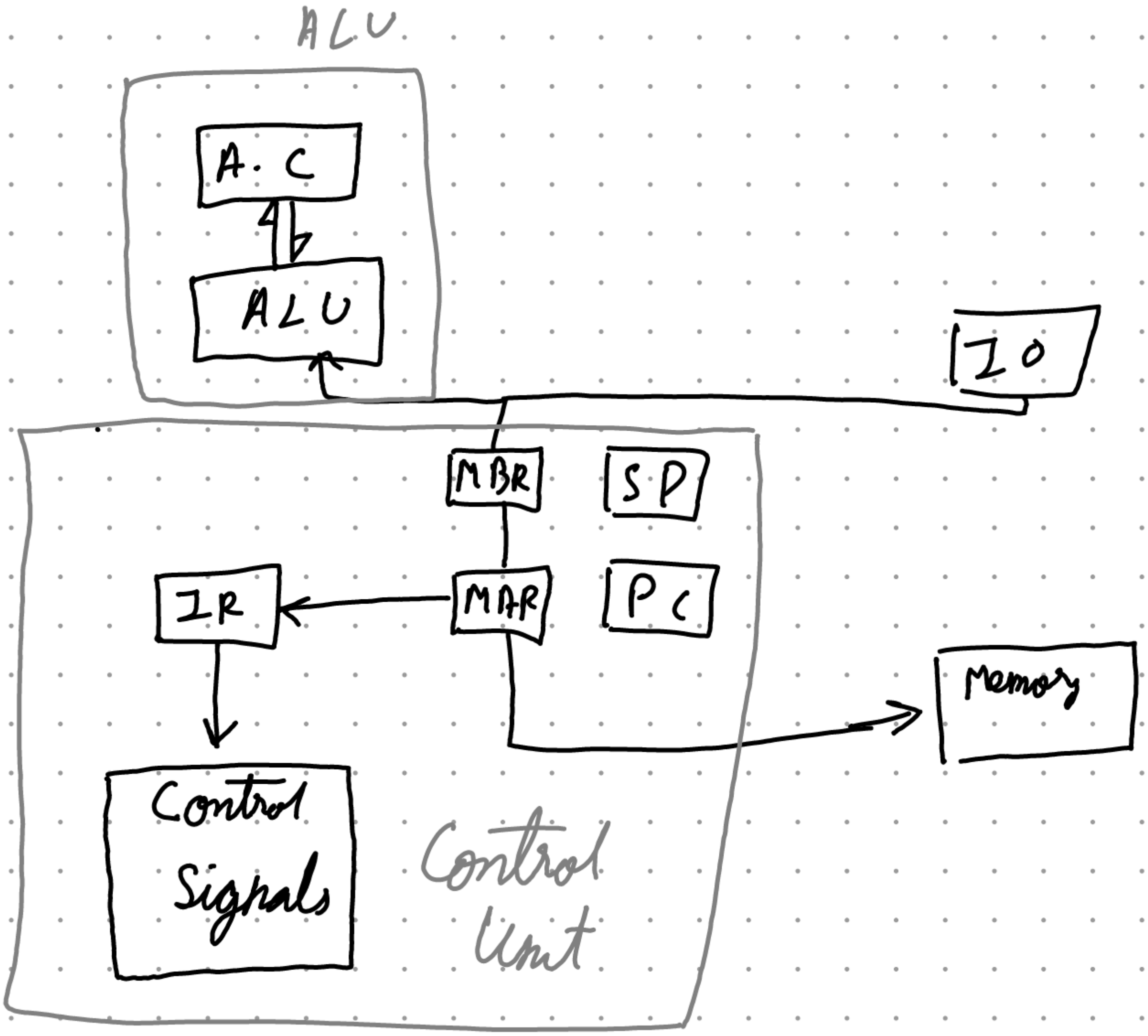
(ii) S.P.

(iii) Accumulator

(iv) MAR

(v) MBR

(vi) I.R.

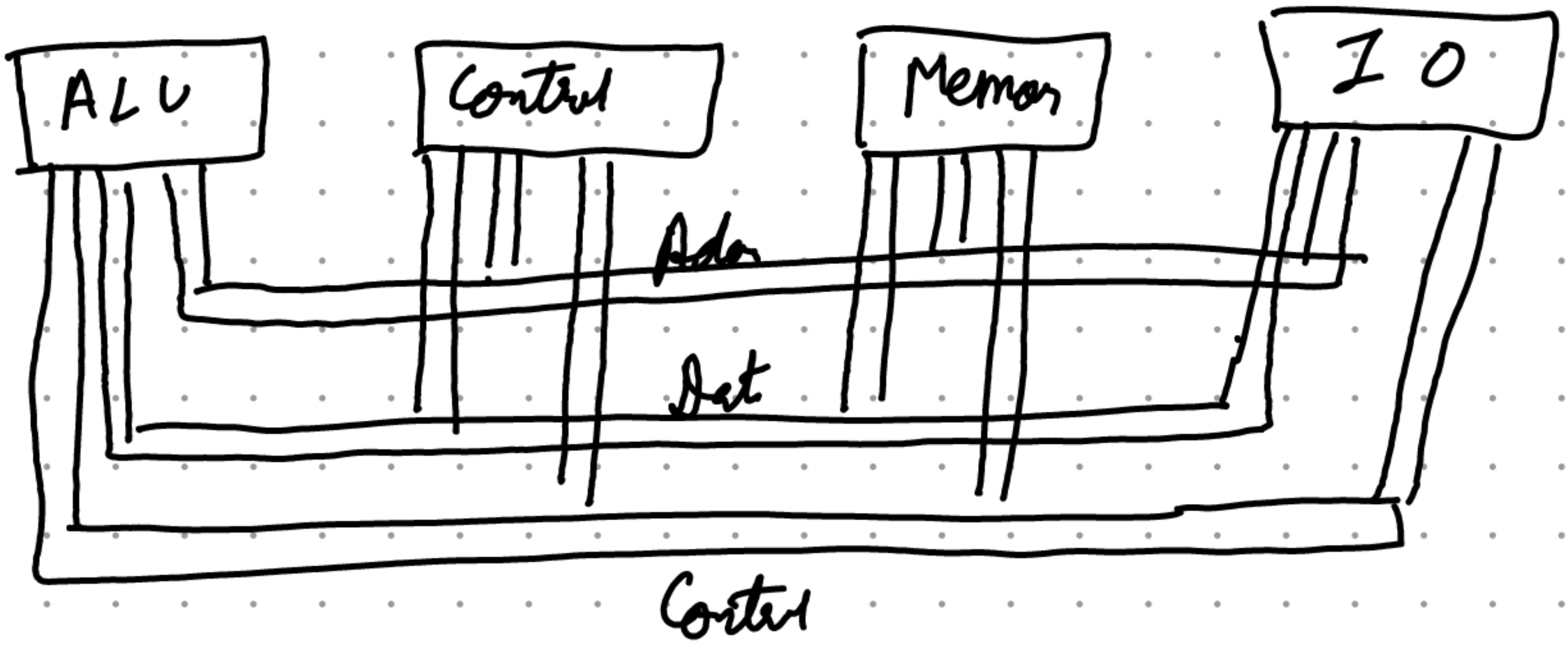

ALU
A.C
ALU
IO
MBR
SP
IR
MAR
PC
Memory
Control
Signals
Control
Unit

③ Bus

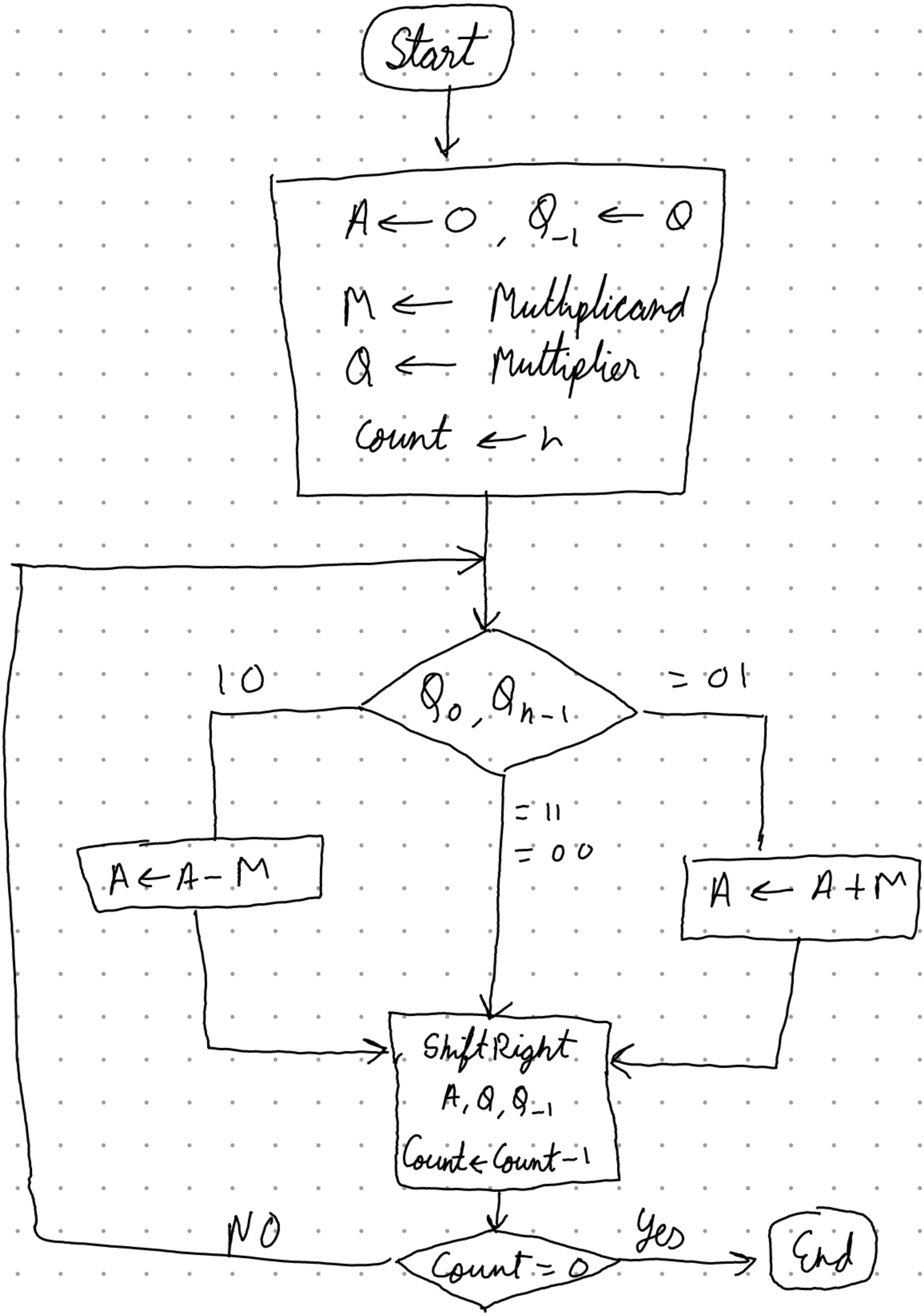
① Data Bus

② Address Bus

③ Control Bus



(4) Booths Multiplication



eg 7×-5

P Q

A 0111

+ M 1011 -Q 0101

P				q_n				q_0	
0	0	0	0	0	1	1	1	0	step 1
0	1	0	1	0	1	1	1	0	
0	0	1	0	1	0	1	1	1	
0	0	0	1	0	1	0	1	1	step 2 11m
0	0	0	0	1	0	1	0	1	step 3
1	0	1	1	1	0	1	0	1	step 4 An
1	1	0	1	1	1	0	1	0	Sh

00100011 | 0 Ignore q_0

= -35

Booths Recoding

Q : 9 = 01001

M : 5 = 00101

-5 = 11011

0 1 0 0 1 0 — Recoded q

+1 −1 0 +1 −1

16 8 4 2 1

16 − 8 + 2 − 1 = 9

0 0 1 0 1

1 −1 0 1 −1

1 1 1 0 1 1

0 0 0 0 1 0 1

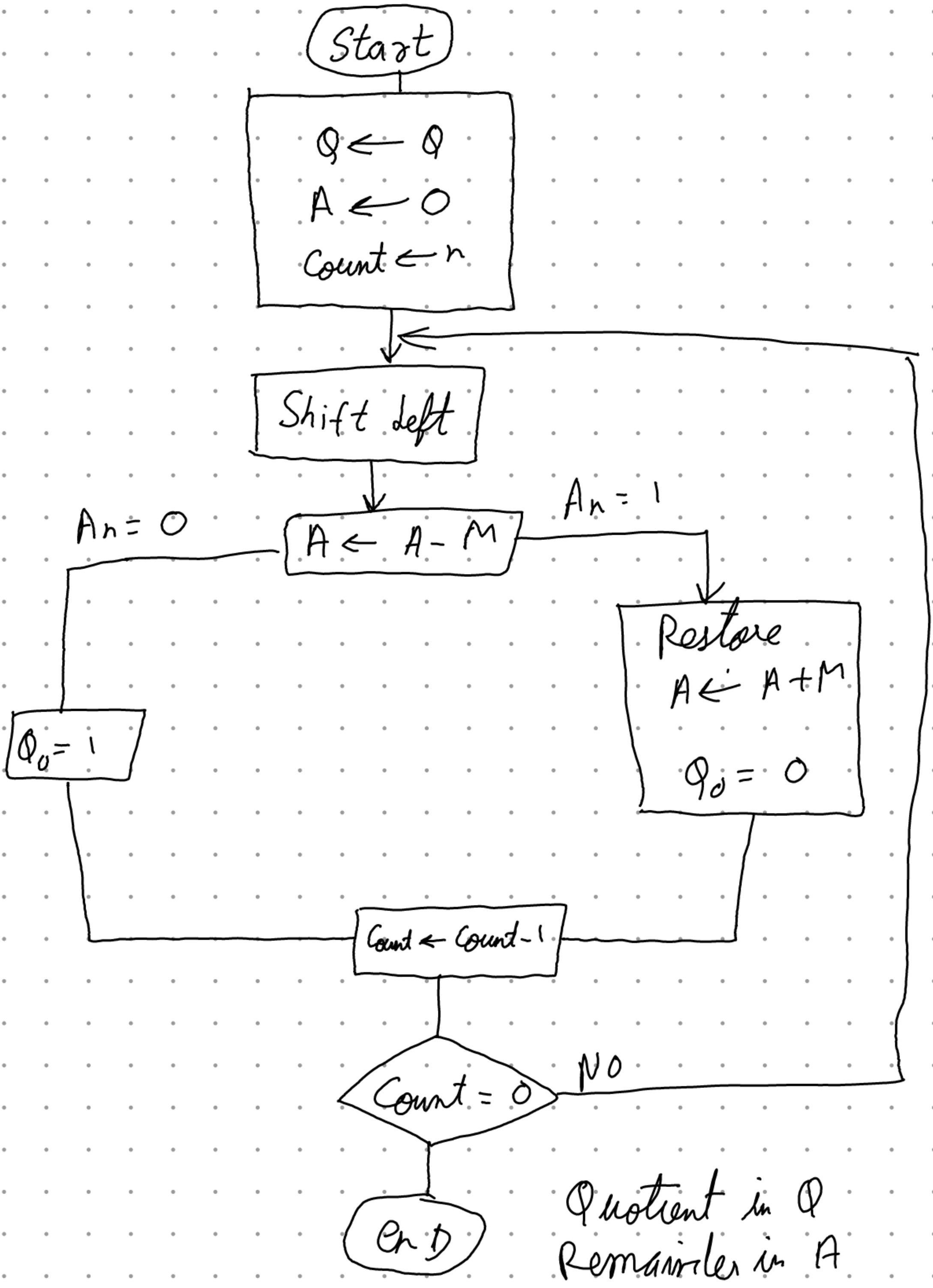
1 1 0 1 1

0 0 1 0 1

1 0 0 1 0 1 0 1 ← 45

64 32 16 8 4 2 1

(6) Restoring Division



Quotient in Q
Remainder in A

7 / 3

Q Dividend 7 0111

M Divisor 3 0011

−3 1101

A Q

0000 111

0001 11 □

111 0 1 □

Restore $q_0 = 0$

0011 10 □

1101

0000 10 □

$q_0 = 1$

0001 01 □

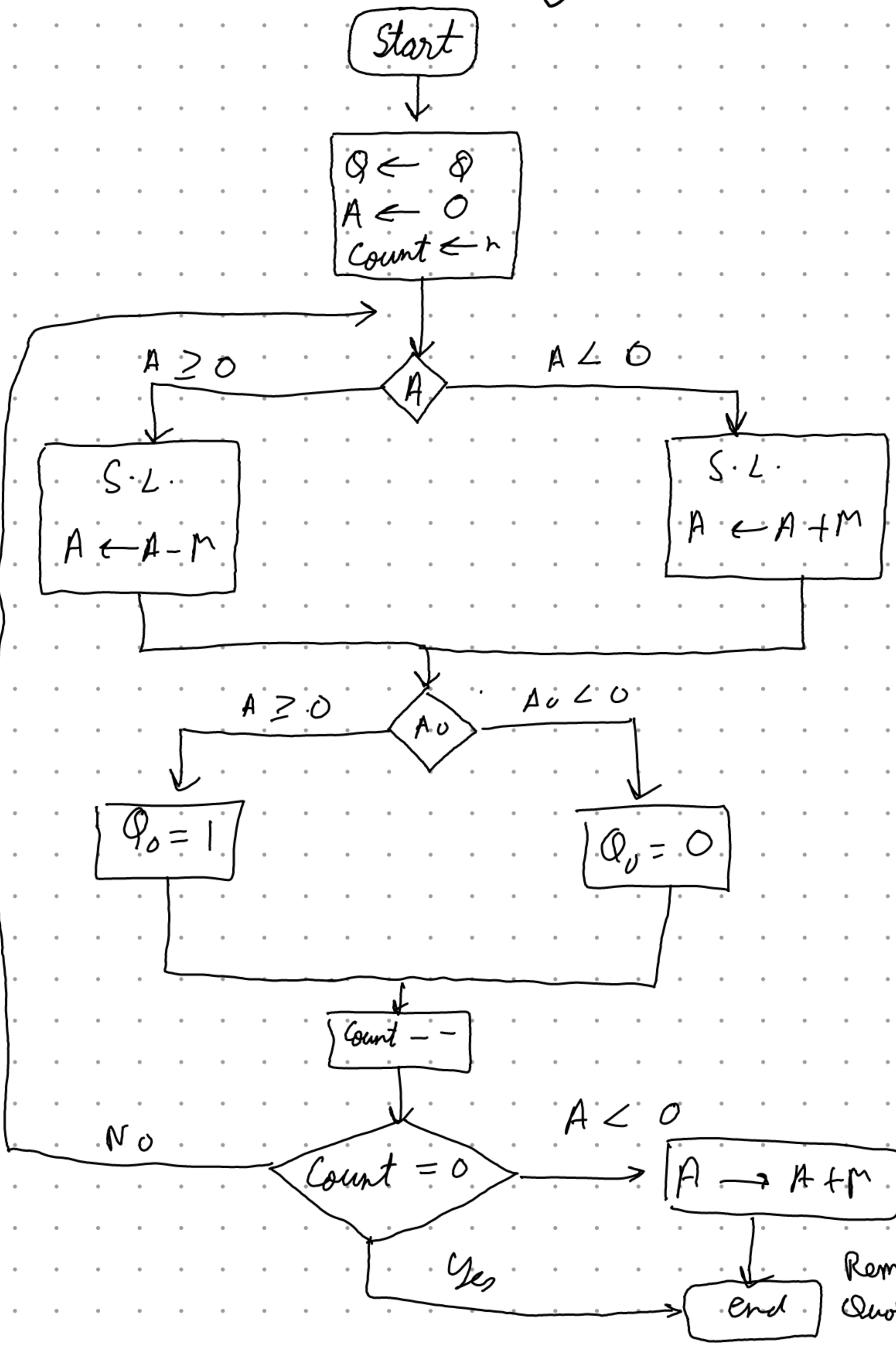
1101

$q_0 = 0$

resto

0010

(7) Non Restoring Division



7/3

Q ← Dividend 7 0 1 1 1

M ← Divisor 3 0 0 1 1

−3 1 1 0 1

0 0 0 0 0 1 1 1

0 0 0 0 1 1 1 □

1 1 0 1 1 1 1 0

1 0 1 1 1 1 0 □

0 0 1 1

1 1 1 0 1 1 0 0

1 1 0 1 1 0 0 □

0 0 1 1

0 0 0 0 1 0 0 1

0 0 0 1 0 0 1 □

1 1 0 1

1 1 1 0 0 0 1 0

0 0 1 1

0 0 0 1 0 0 1

$A_0 = 0 \therefore$ SUB

$A_0 = 1 \therefore$ ADD

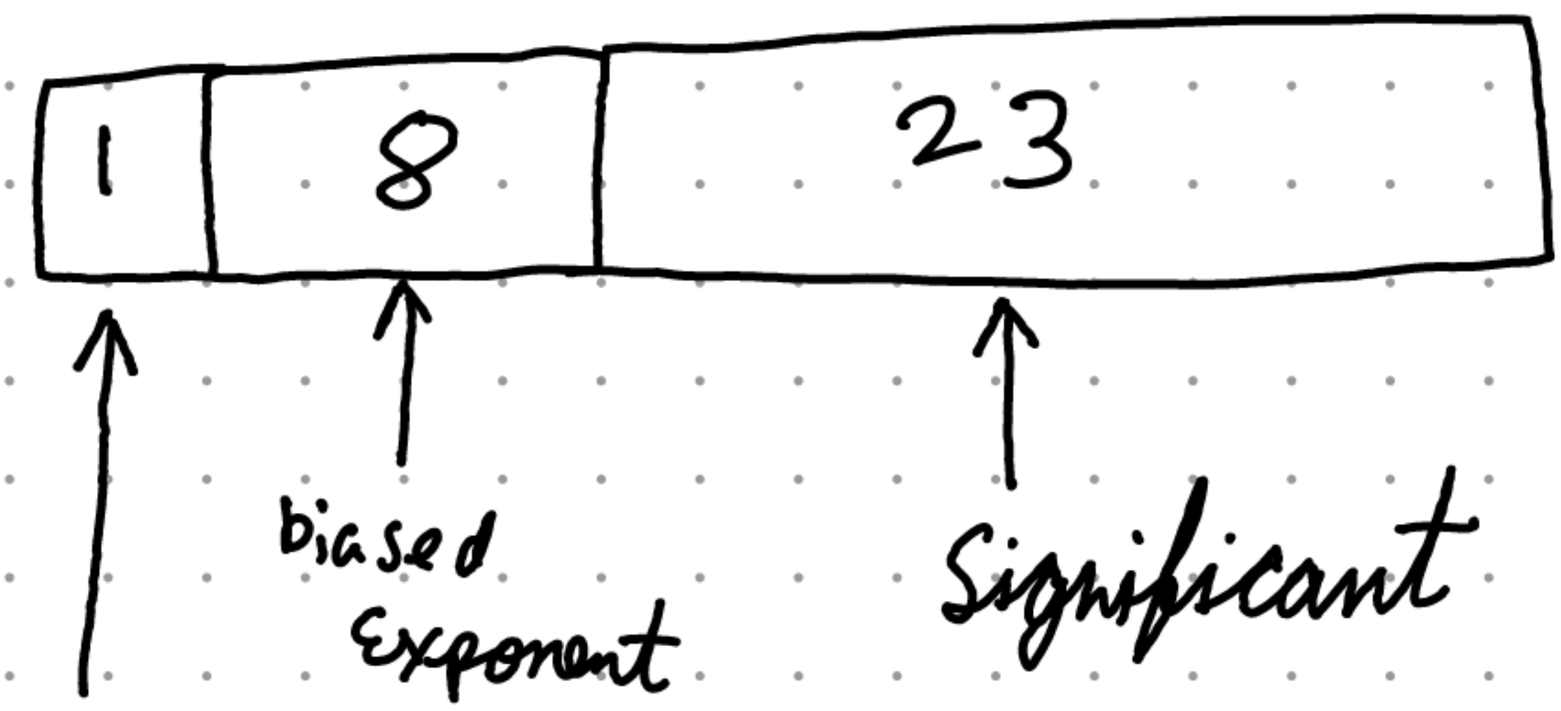
$A_0 = 1 \therefore$ ADD

$A_0 = 0 \therefore$ SUB

$A < M \rightarrow A + P + 1$

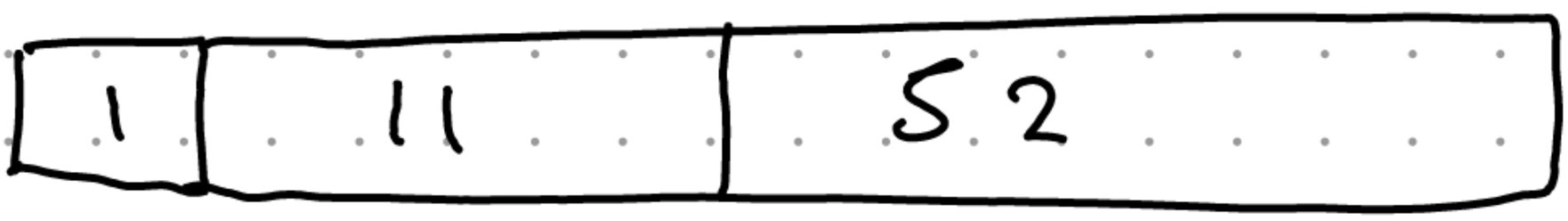
⑧ IEEE floating

① 32 bit Single precision



Bias 127

② 64 bit Double precision



Bias 1023

eg 309.175 in 32

$309 \rightarrow (1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1)_2$

0.175	0
0.35	0
0.7	0
1.4	1
0.8	0
1.6	1
1.2	1
0.4	0
0.8	0
⋮	⋮

$100110101.0001\ldots$

$= 1.00\ 1101\ 0100\ 0\ldots \times 2^8$

$8 + 127 = 135$ Exponent

$136 = (10000111)_2$

Sign = 0 (+ve)

0 10000111 00110101....

32

$8 + 1023 = 1031$ Exp

$1031 = (10000000111)$

Sign = 0 (+ve)

0 10000000111 00110101...

64

Multiplication → Add Exp
Sub Bias

Division → Sub Exp
ADD Bias

Addition
Substraction → ① Check 0
② Normalize exp
③ ADD/SUB Significant
④ Overflow Exp ++ SR
Underflow Exp -- SL
⑤ Find Ans
Normalize

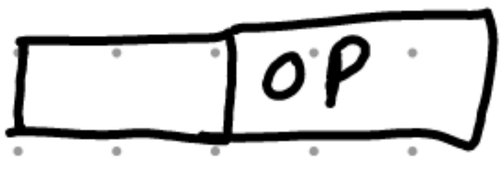
Module 3

- ☆ Addressing Modes
- ☆ Instruction Cycle
- ☆ Register organization
- ☆ Risc vs Cisc
- ☆ Control Unit

① Addressing Modes.

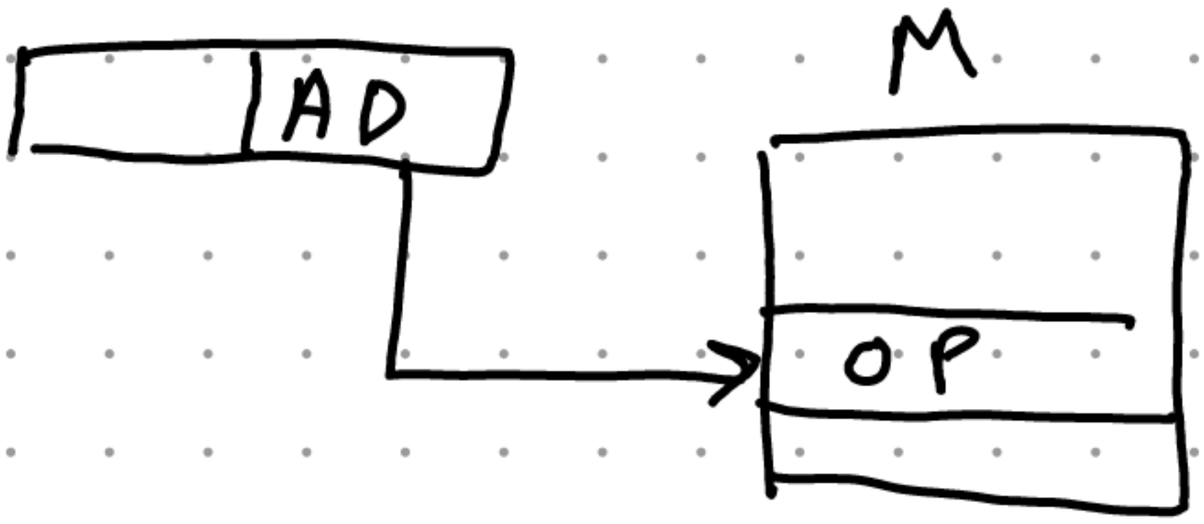
M → Memory
OP → operand
AD → Address

(A) Immediate



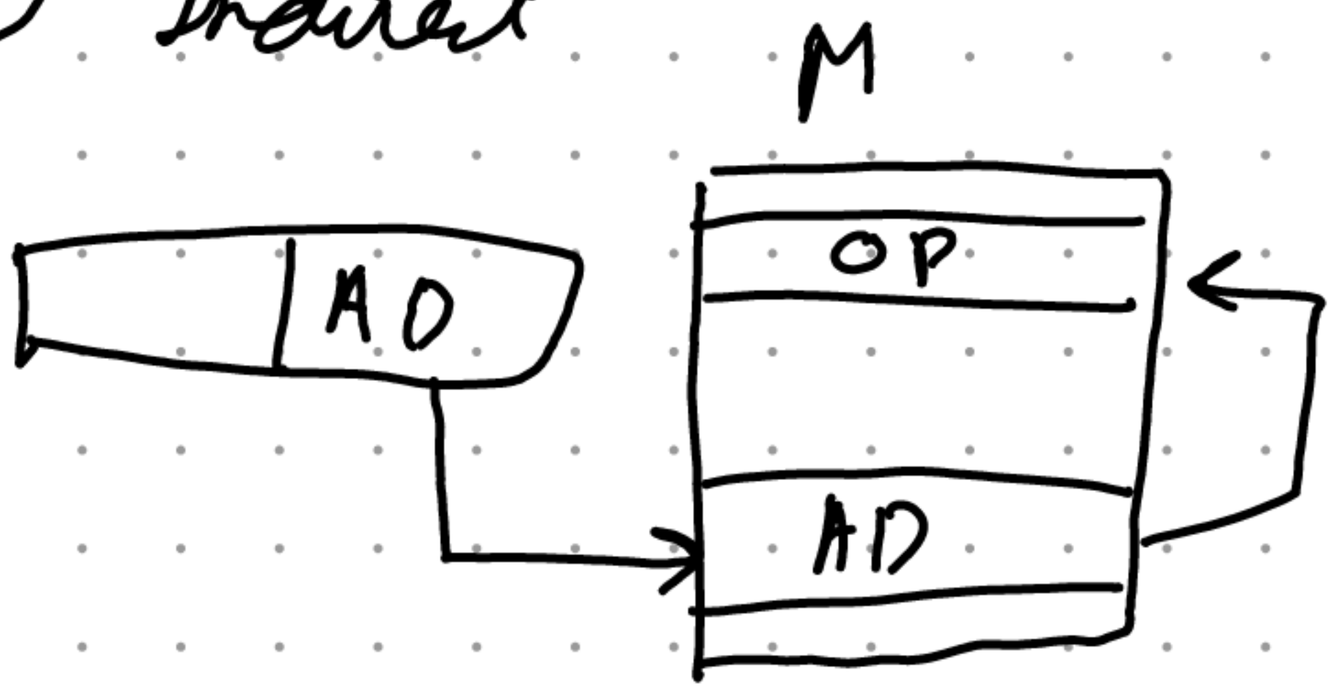
eg ADD 47, #32H

(B) Direct



ADD 47, [32H]

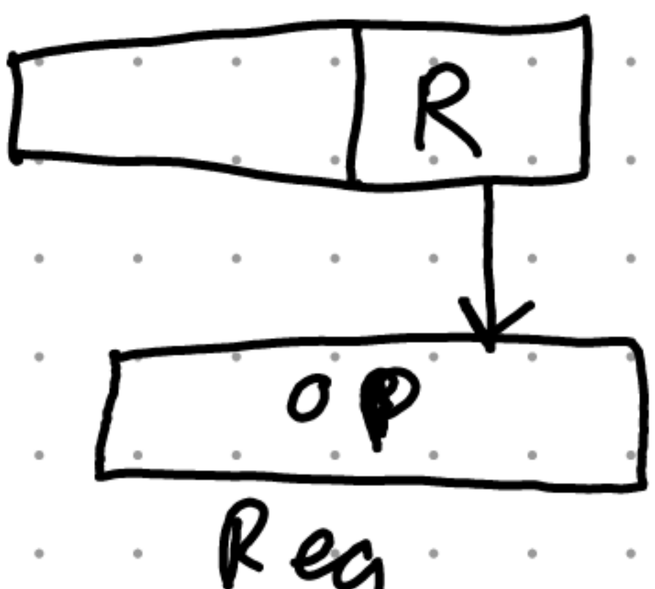
(C) Indirect



R → Reg Name

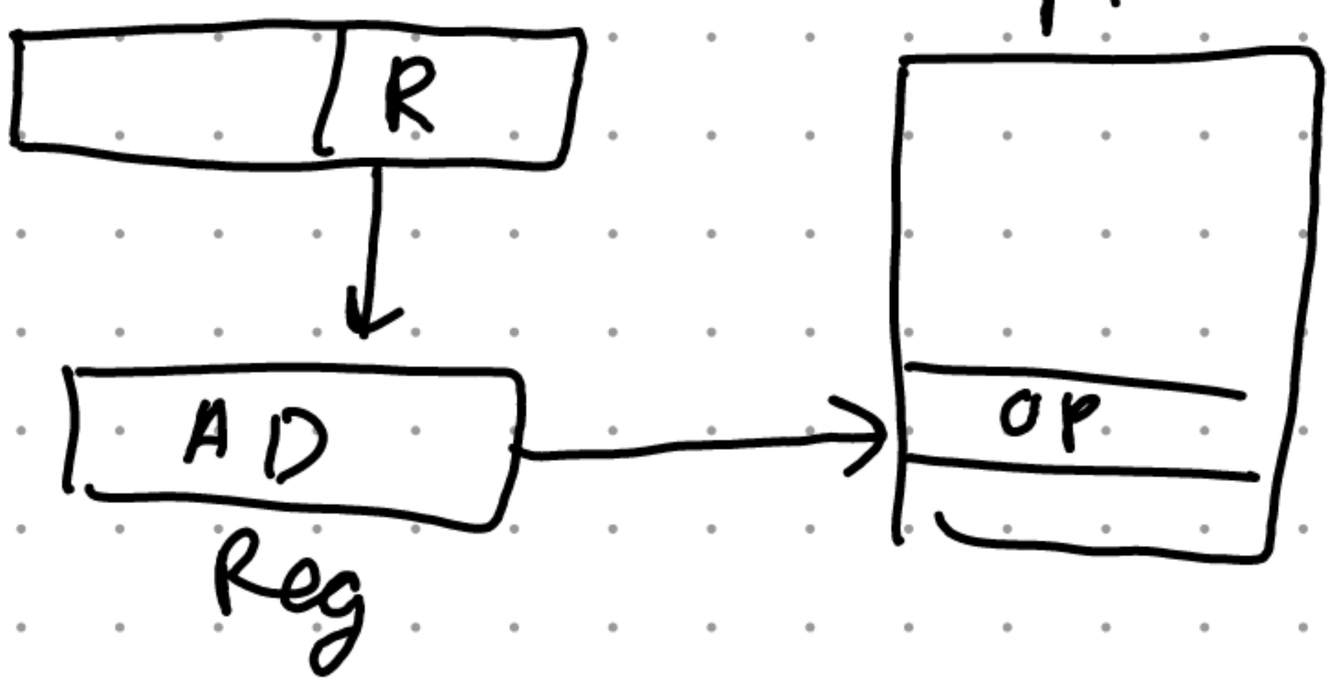
(D) Register

MOV AX, BX

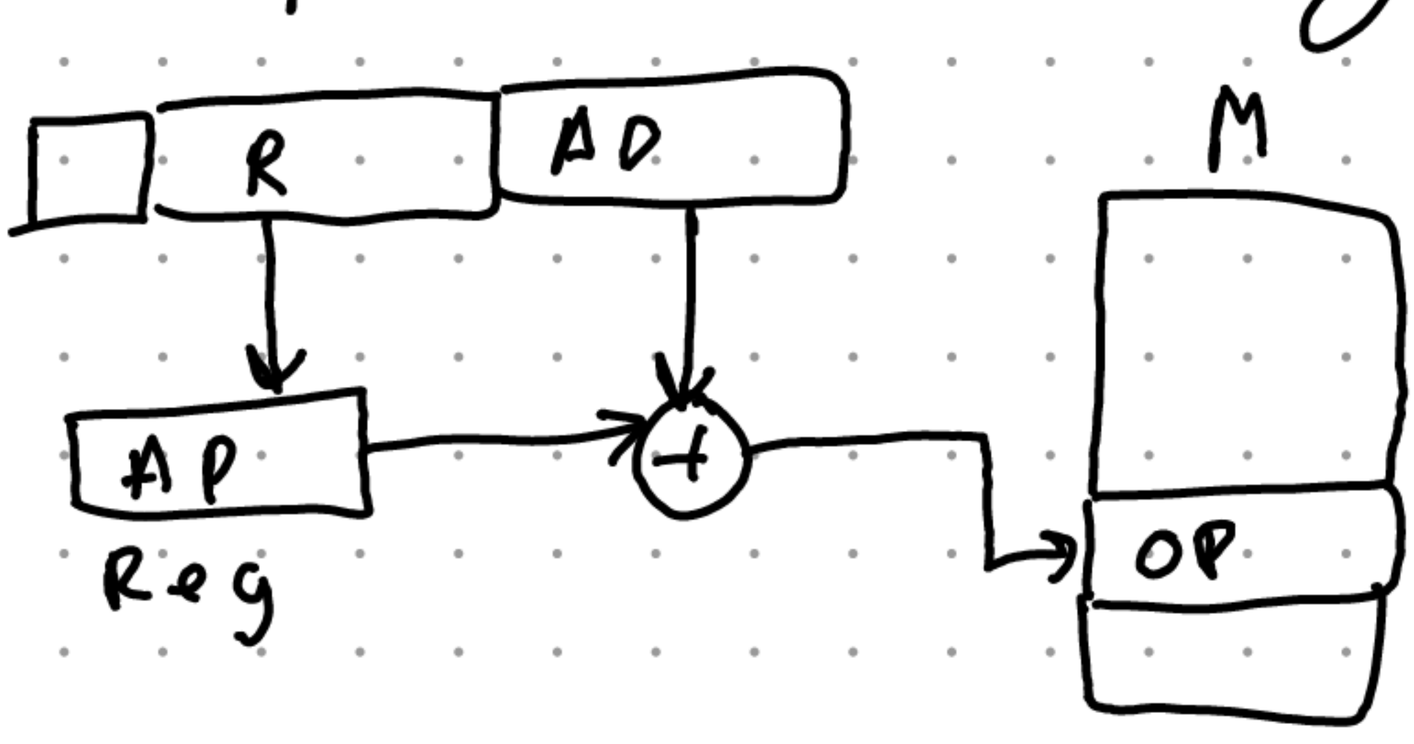


(E) Register Indirect

MOV AX, [BX]



(F) Displacement Addressing



④ Stack Addressing

eg POP Present in stack Reg

(2) Types of Registers

(A) Address Registers

(B) Base Registers

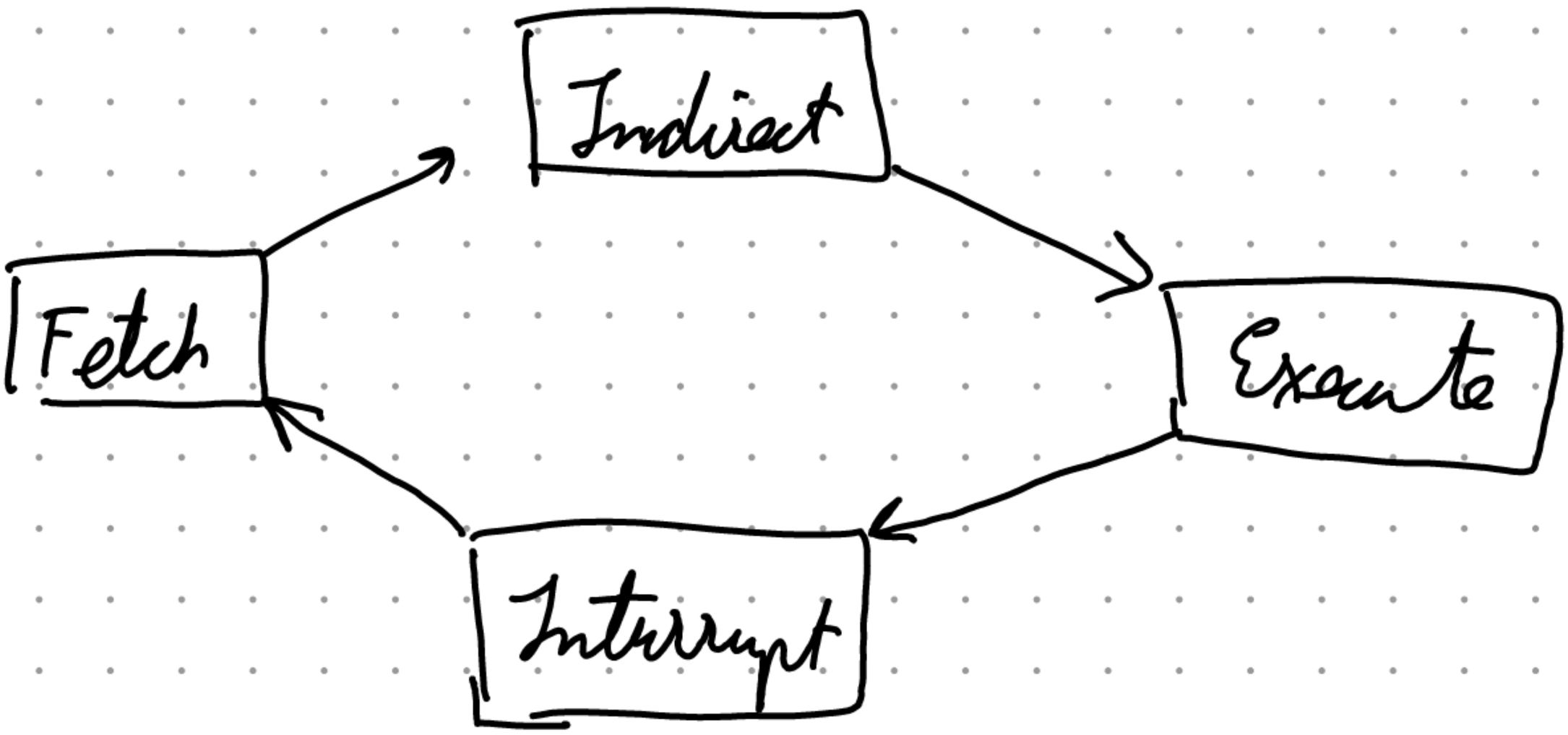
(C) Control Reg Pointers & Index — SP, PC + PSW

(D) Data Registers

(E) PSW (flags)

(F) General Registers

③ Instruction Cycle



Fetch → loads Instruction

Indirect → loads operands from memory

Execute → Execute the Instruction

Interrupt → Check for any Interrupt

④ MicroInstructions

(A) Fetch

MAR ← PC

MBR ← Memory

IR ← MBR

PC ← PC+1

(B) Indirect

MAR ← Address (IR)

MBR ← Memory

IR (Adress) ← MBR

(C) Interrupt

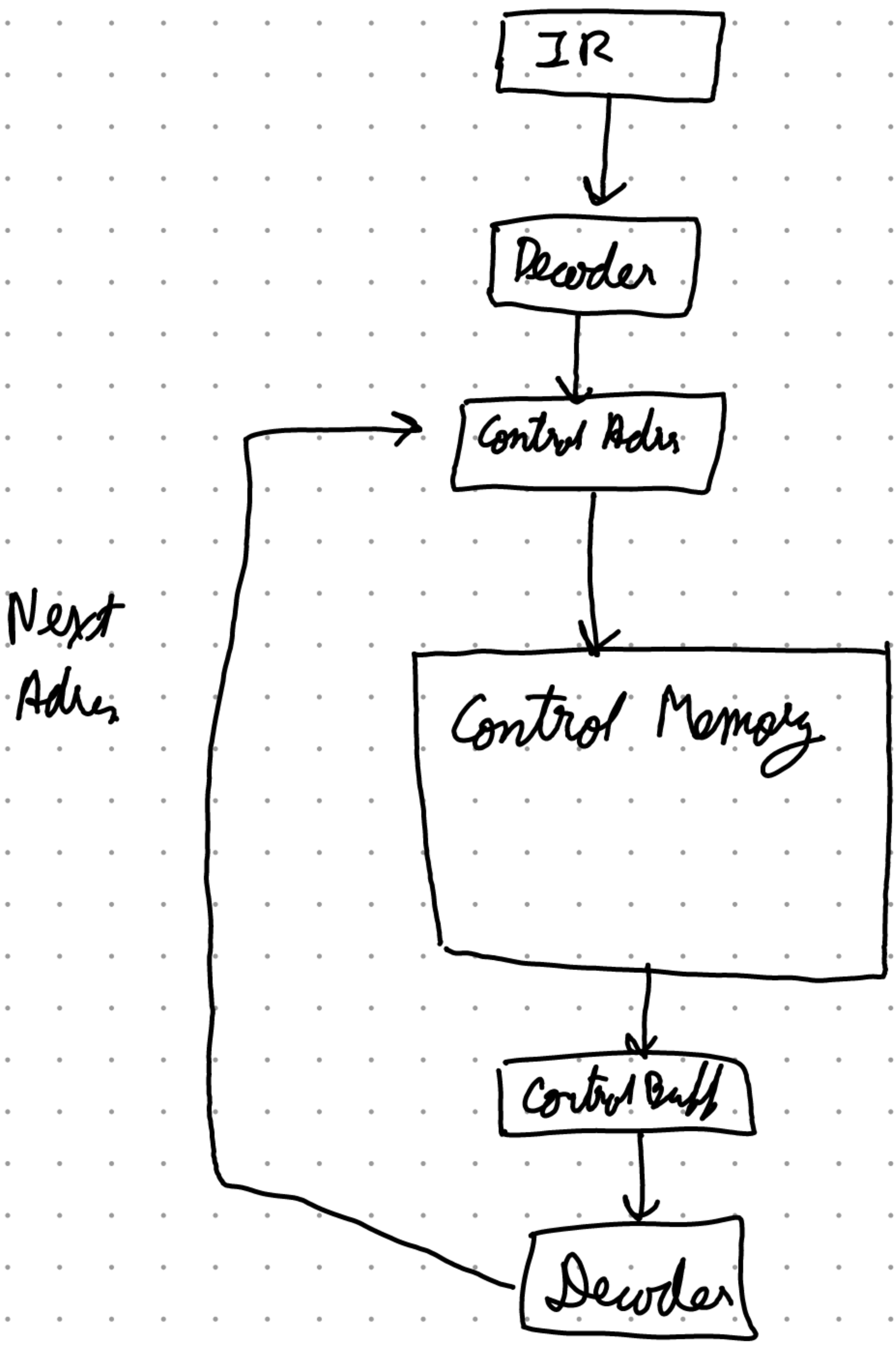
MBR ← PC

MAR ← Save

PC ← Routine

Mem ← MBR

(5) Microprogrammed Control Unit



⑥ RISC vs CISC

CISC	RISC
① Complex Instruction set	① Reduced Instruction set
② More number of Instructions	② Less number of Instructions
③ Smaller programs	③ Larger programs
④ Many Addressing modes	④ Less Addressing modes
⑤ Complex pipelining	⑤ Simple pipelining
⑥ Microprogrammed Control Unit	⑥ Hardware Control Unit
⑦ Many Instructions accessing memories	⑦ Only load & Store access Memories called load Store Architecture
⑧ Instruction last many cycles	⑧ Instructions last 1-2 clock cycles
⑨ Requires More Power	⑨ Less Power requires
⑩ Less registers	⑩ More Registers
eg Intel	eg Apple, ARM
Simple Compiler	Complex Compiler

Module 4

- ☆ Cache Memory
- ☆ Cache policies LRU FIFO
- ☆ Cache Mapping
- ☆ SRAM Vs DRAM

① Access Methods

① Sequential eg Tape

② Direct (Random + Sequential) eg Disk

③ Random eg RAM

④ Associative eg Cache.

② SRAM vs DRAM

SRAM	DRAM
① Static RAM	① Dynamic RAM
② No refreshing logic	② Needs Refreshing
③ Charge in transistor	③ Charge in capacitor
④ 2 FlipFlop used	④ Charge leaks
⑤ More expensive	⑤ Less expensive
⑥ Faster	⑥ Slower
⑦ eg cache	⑦ eg RAM

③ Types of ROM

PROM → Programable once

EPROM → Erasable programable (UV)

EEPROM → Electrically Erasable

Flash → Erase whole memory

(4) what is Cache

① Cache is small highspeed Memory

② Cache works on principle of locality

(3) 1. CPU Request

2. check in cache

3. Give data (Fast)

4. Else fetch from Memory

5- Give data (slow)

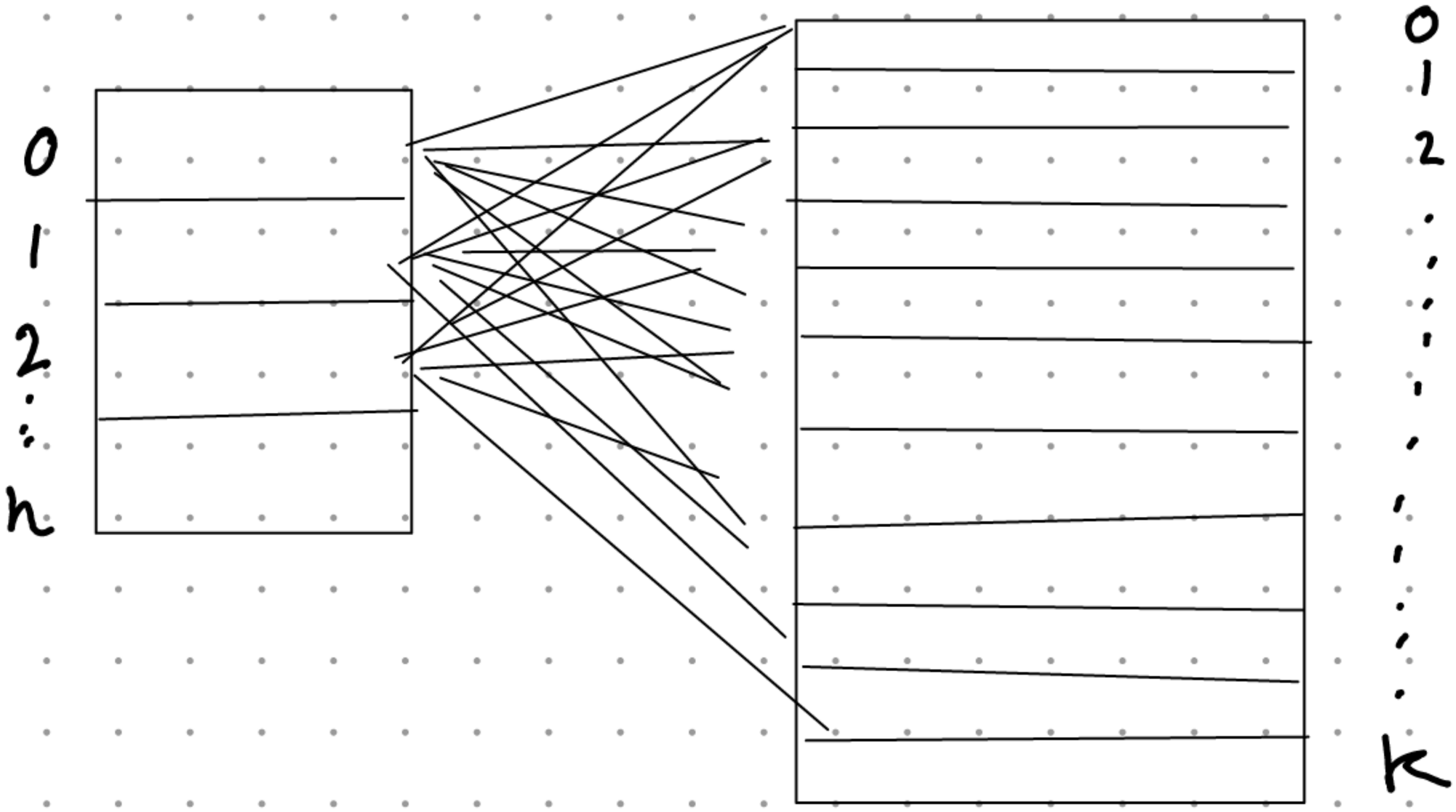
(5) L_1 L_2 L_3 Cache

L_1 → embedded in chip

L_2 → located in CPU

L_3 → outside CPU (Near)

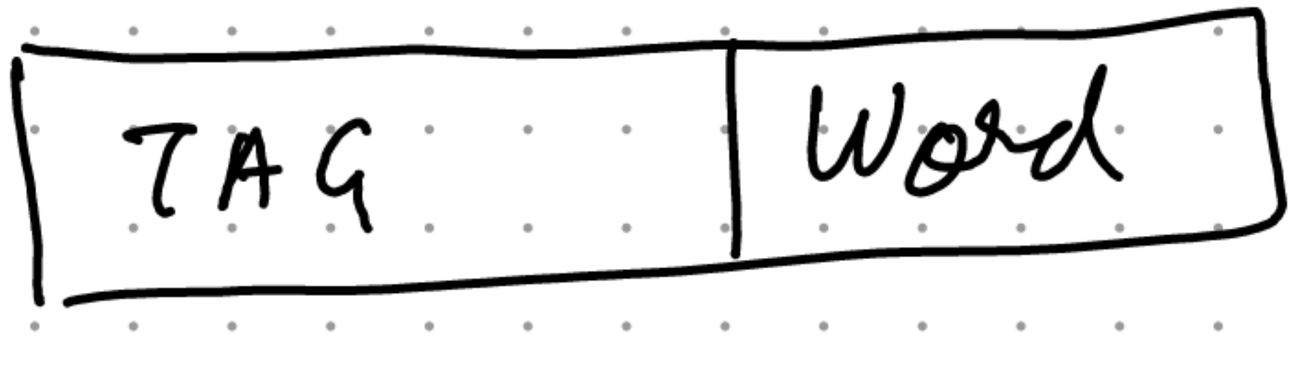
⑥ Cache Mapping



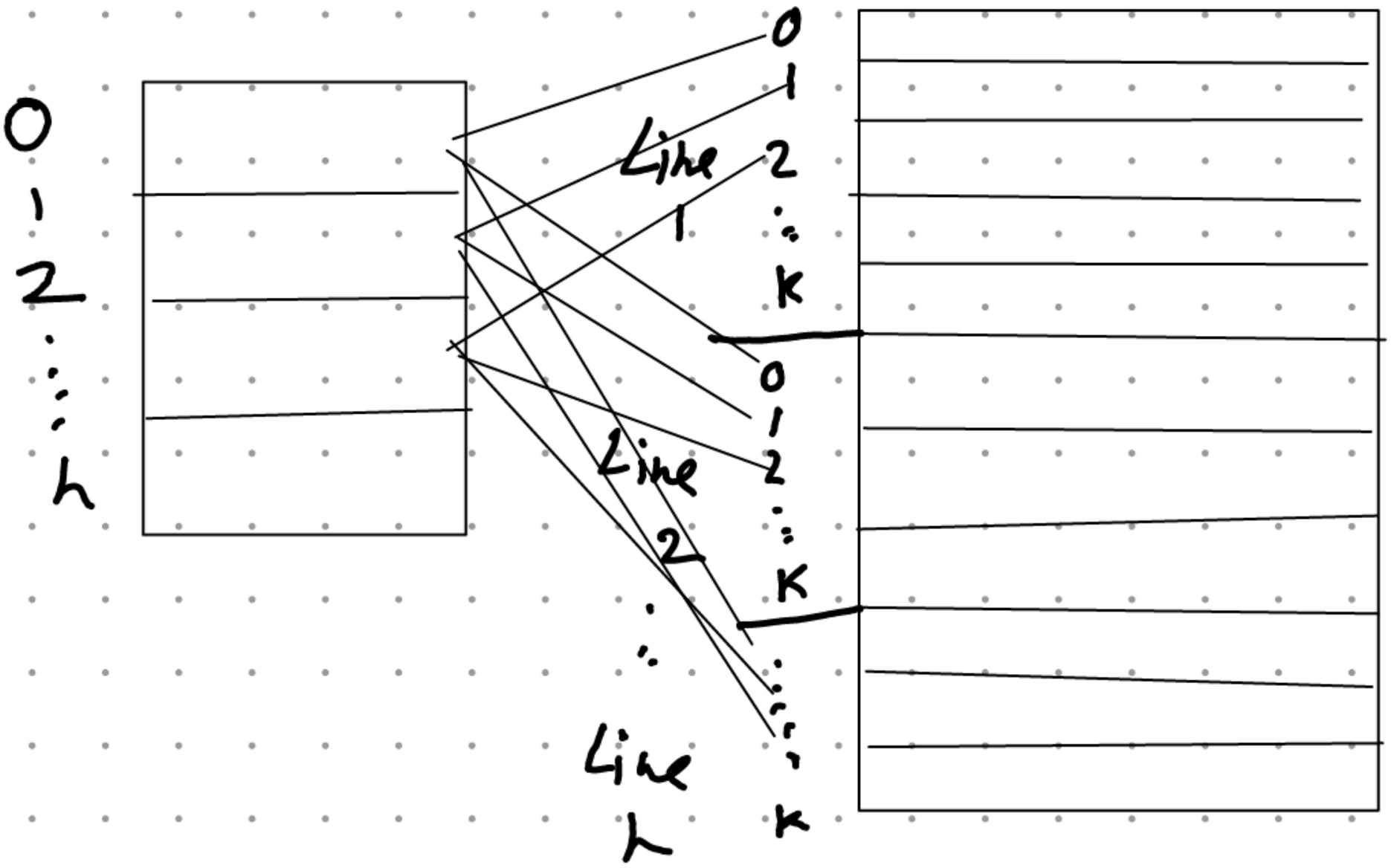
Fully Associative Mapping

Tag size $\log_2 k$

Word size : $\frac{\text{Memory Size}}{k}$

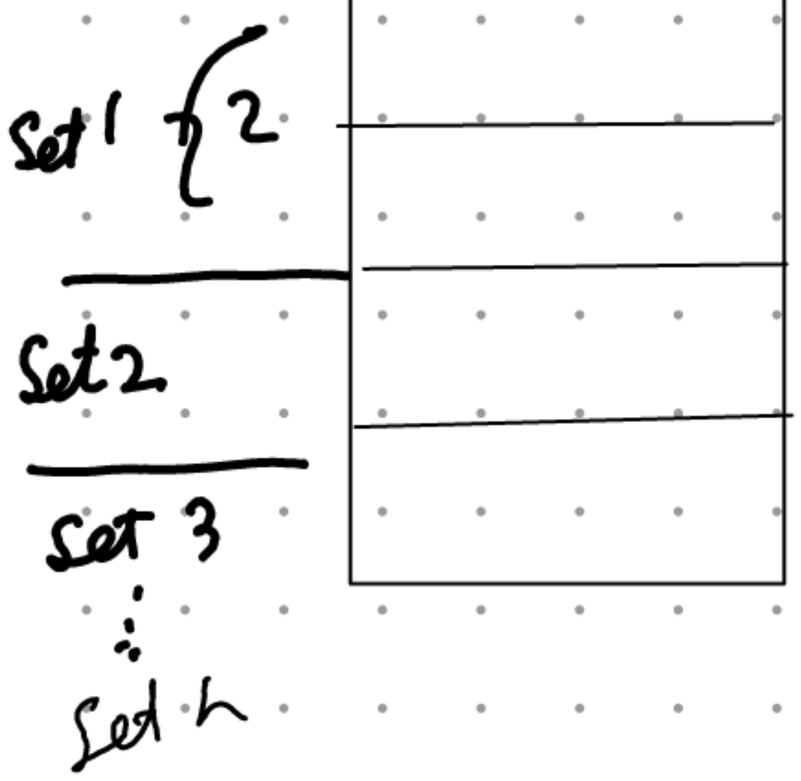


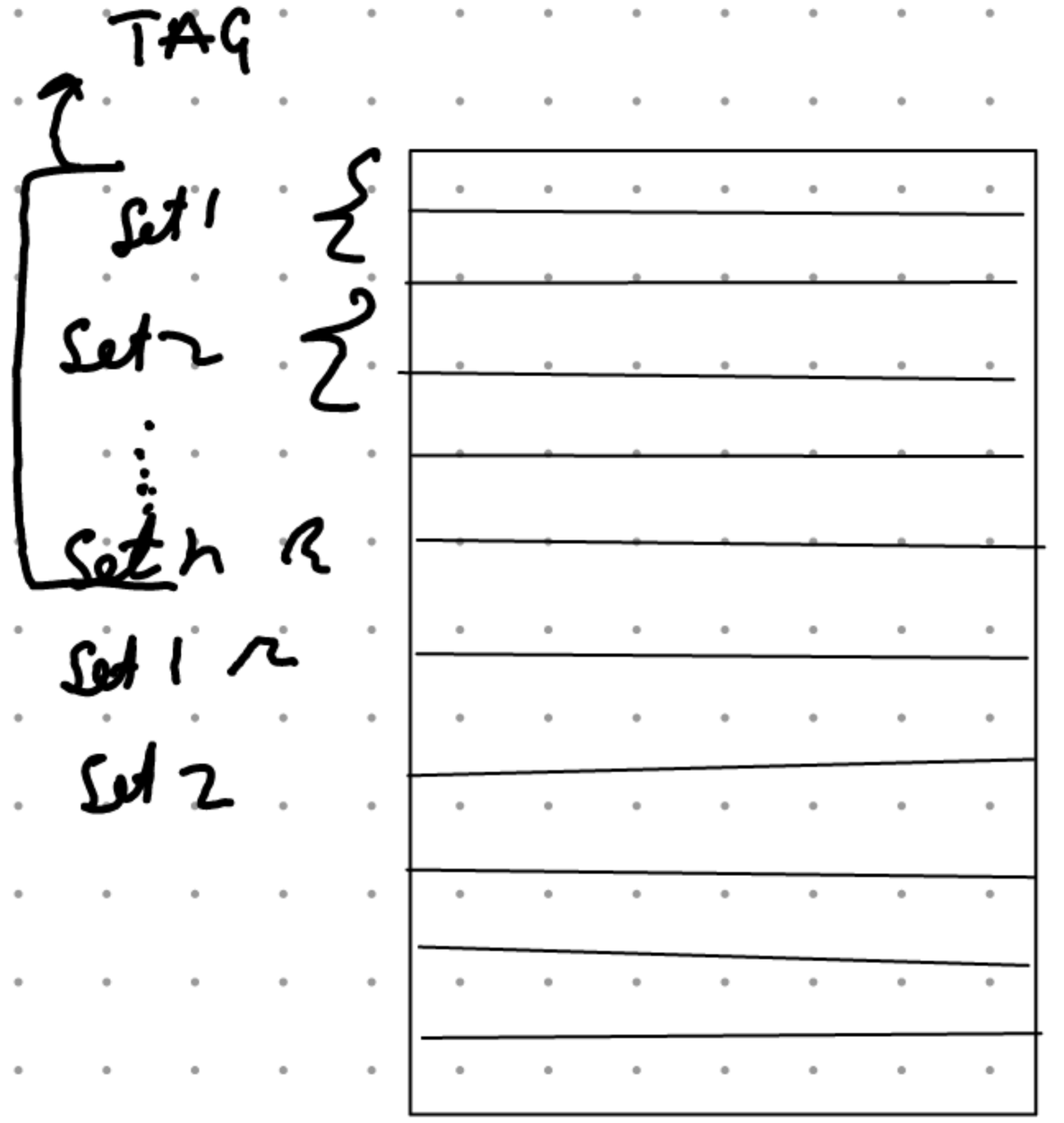
Direct Mapping

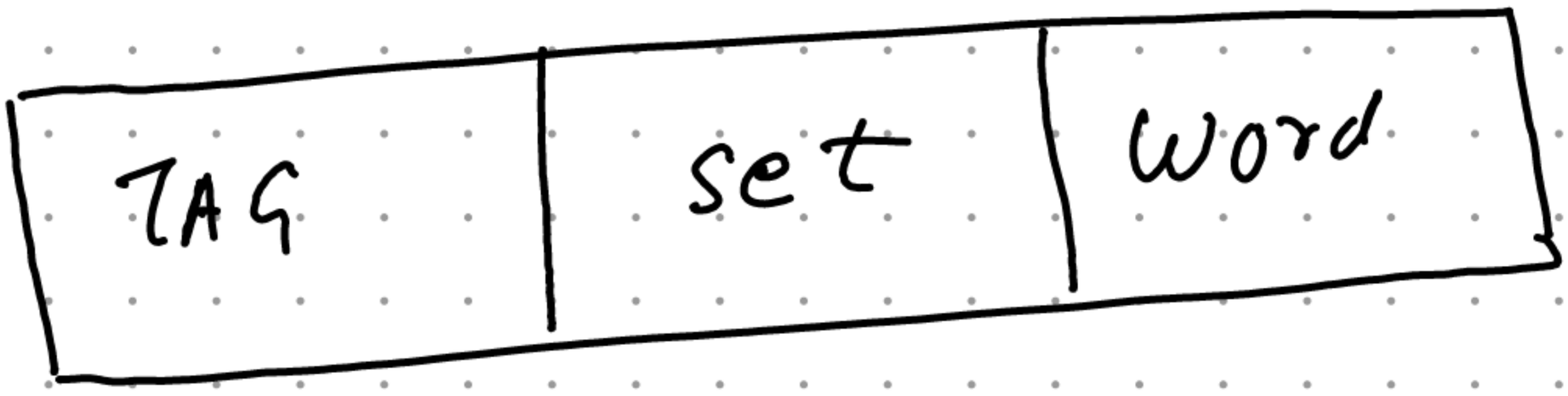


Line $\log_2 n$

The Memory is divided into n lines


Set 1
2
Set 2
Set 3
Set h


TAG
Set 1
Set 2
Set n
R
Set 1
Set 2
2 lines


TAG
set
word

eg Cache : 256 blocks
16 words

Memory : 16 bit adres
: 4kb block (4096)
16 words

Word length → $\log_2 16 = 4$ Always.

For Fully associative → TAG = 16 − 4 = 12.

For Direct : Cache size : 256 blocks

∴ Block size = 256

$\log_2 256 = 8$

∴ 4 TAG

For 2-set Associative, Block Memory into 2 Parts

∴ No of sets will be $\frac{256}{2} = 128$

∴ $\log_2 128 = 7$

TAG = 5

By Cache 128 blocks

Memory 2k blocks

32 words each.

① Associative Mapping

$\log_2 32 = \underline{\underline{5}}$ word

TAG = $\underline{\underline{11}}$

② Direct

Block = 7 $\log_2$ by

TAG = 4

Word 5

③ 4 way set

Set : $\frac{128}{4}$ = 32 = $\underline{\underline{5}}$ bit

TAG = 6 bit

Word : 5 bit

(5) MESZ Protocol

Used to prevent Cache Coherence

M	Modified	local copy is dirty
E	Exclusive	local copy exists & is update
S	Shared	local & Main Memory
I	Invalid	Copy doesn't exist

(6) Interleaved Memory

Spreading of Memory

Reduced waiting

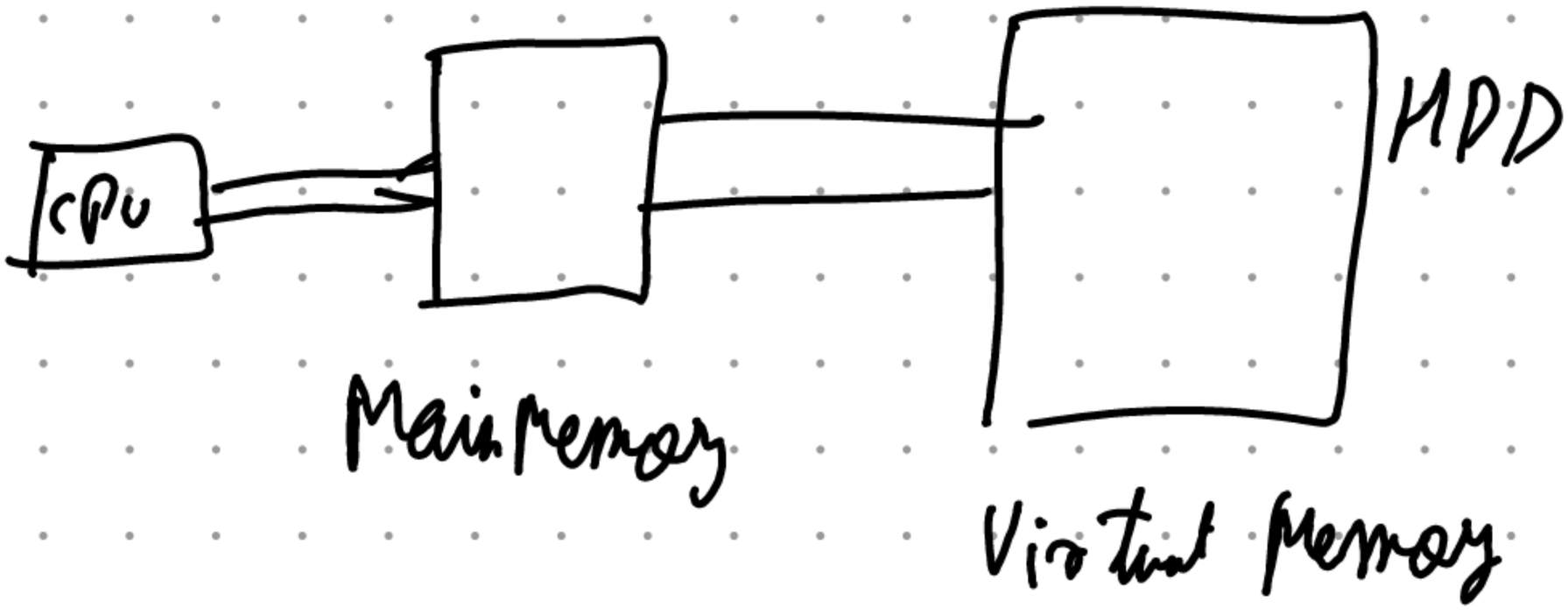
Parallel access

(7) Virtual Memory

(1) Used to increase size of Memory

(2) Secondary memory is used as primary memory

(3) Use: Paging & Segmentation



⑧ Paging & Segmentation

fixed size

① Primary Memory & Secondary M- is divided into Pages

② Pages are small unit of data, Faster

③ Internal fragmentation

④ Physical divisions

⑤ Needs page - table

① Segmentation is Variable Size paging

② Slower than paging

③ External fragmentation

④ Logical divisions

⑤ Needs Segment table

⑨ Page replacement Policie

① FIFO

② LRU → last used element remove

③ OPT → look into future & chek

eg 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 4 4 0

3 seg.

FIFO

7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7	1	1	1
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0	0	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1	1	1	1

LRU

7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1	1		
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0	0		
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	7	7	4	4	4		

(10) RAID

Redundand Array of Independant Disks

Error proof

Faster Time access -

Raid 0 — Non Redundant

1 — Redundant via Duplication

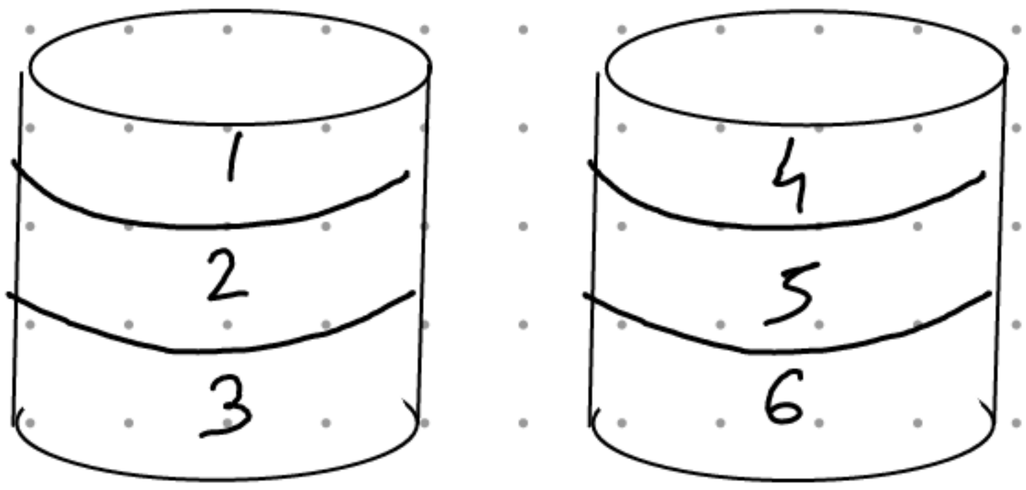
2 — Hamming Codes

3 — Bit interleaved-parity

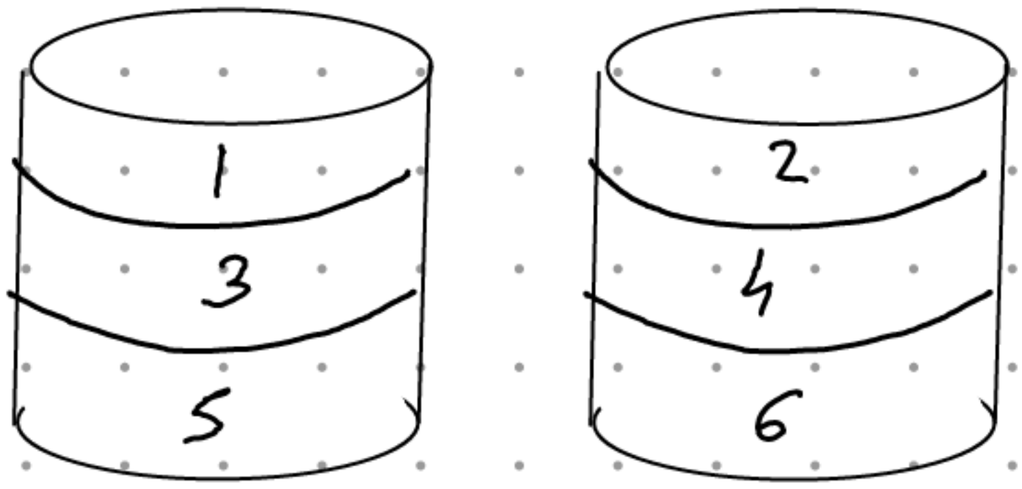
4 — Block parity

5 — Block distributed parity

6 — Dual distributed Parity

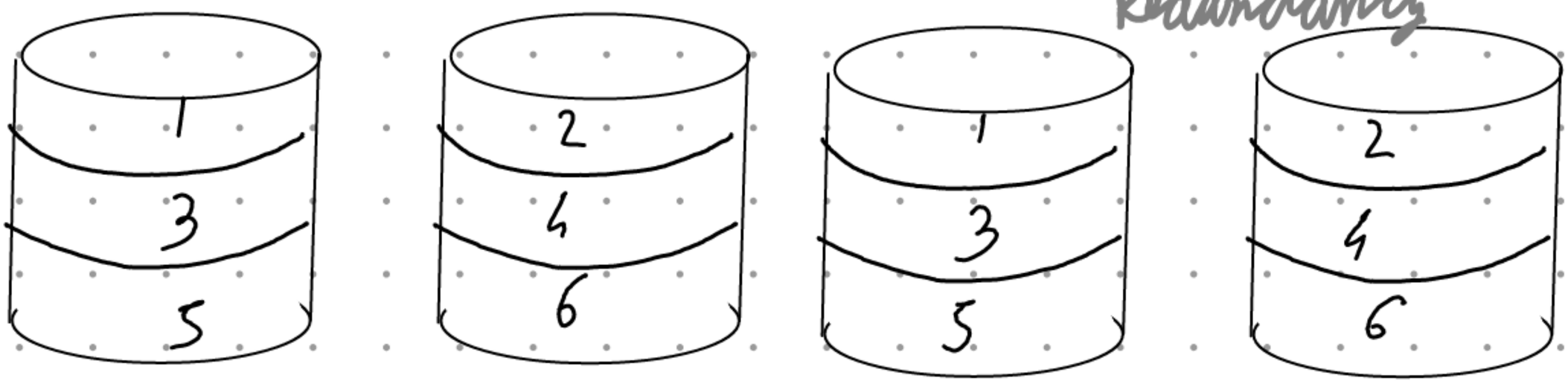


Normal Disk

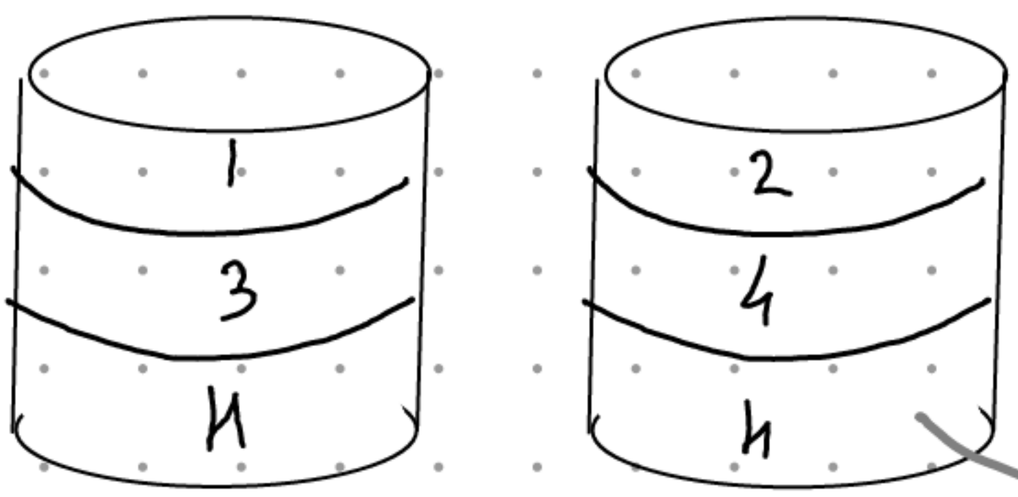


RAZD 0

Redundancy

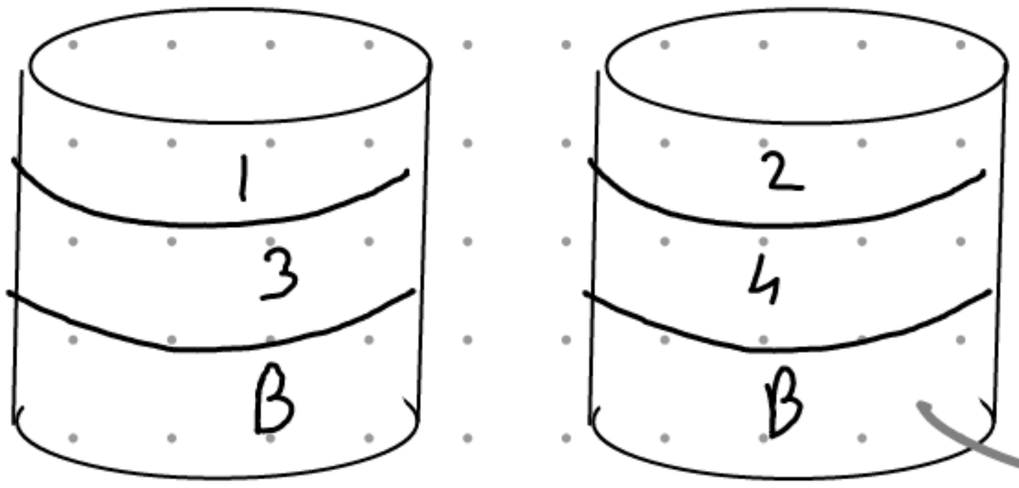


RAZD 1



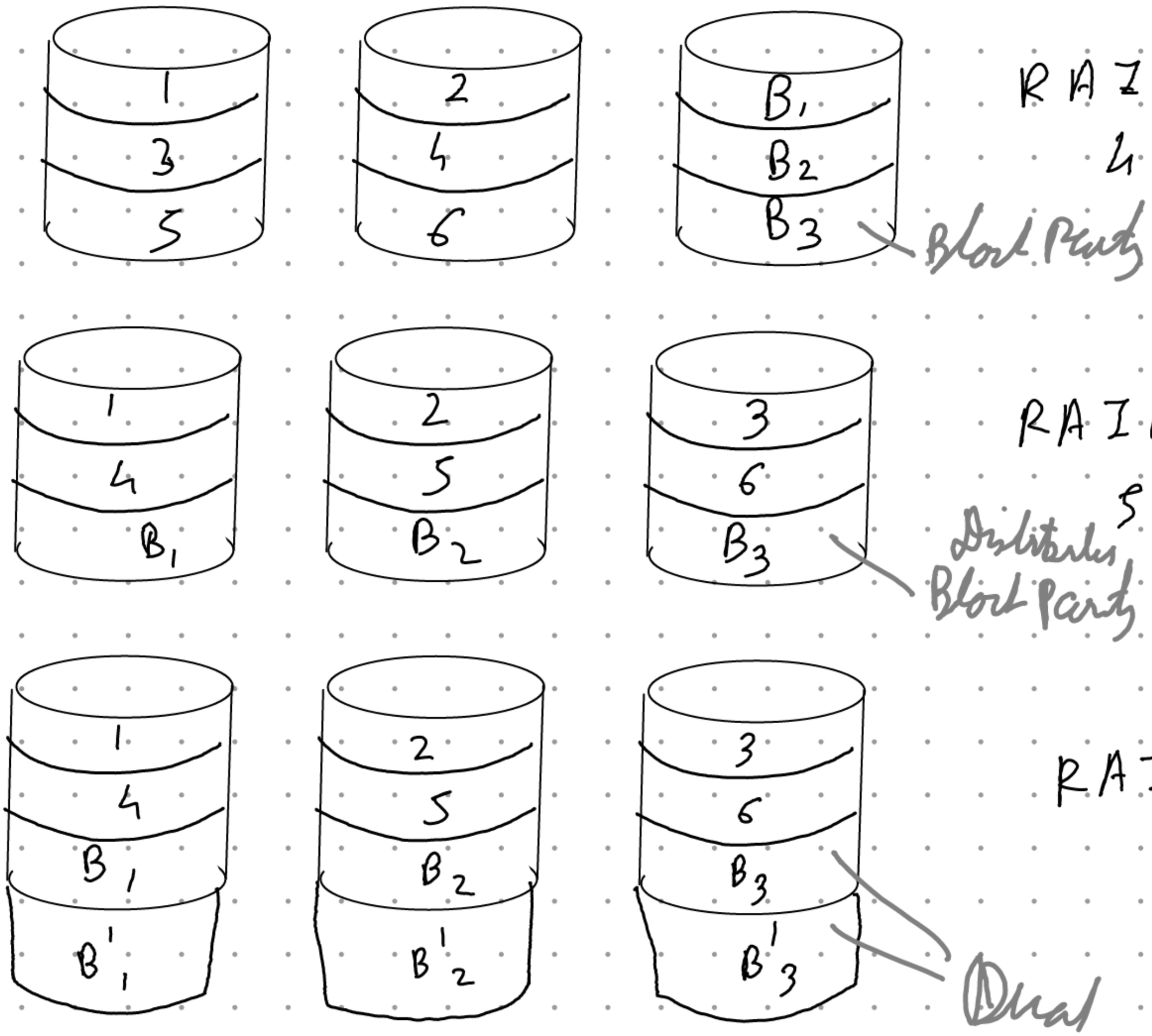
RAZD 2

Hamming Codes



RAZD 3

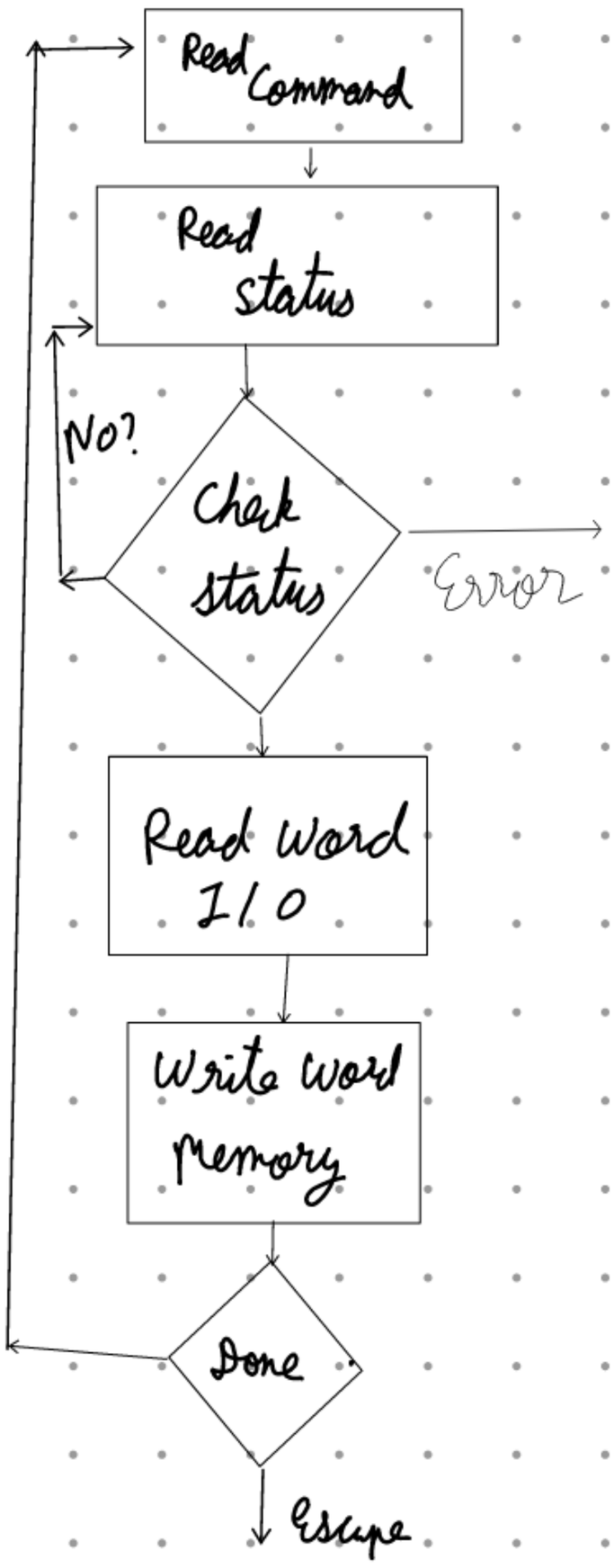
Bit Interleaved Parity


1
3
5
2
4
6
B1
B2
B3
RAID 4
Block Parity
1
4
B1
B'1
2
5
B2
B'2
3
6
B3
B'3
RAID 5
Distributed Block Parity
1
4
B1
B'1
2
5
B2
B'2
3
6
B3
B'3
RAID 6
Dual Distributed Block Parity

Module 5

* I/O Mapped I/O & Memory Mapped I/O
* Interrupt Devis I/O, DMA, Programm

(1) Programmed IO



i) When IO is ready, it sets status Bit

ii) After every interval, CPU checks bit

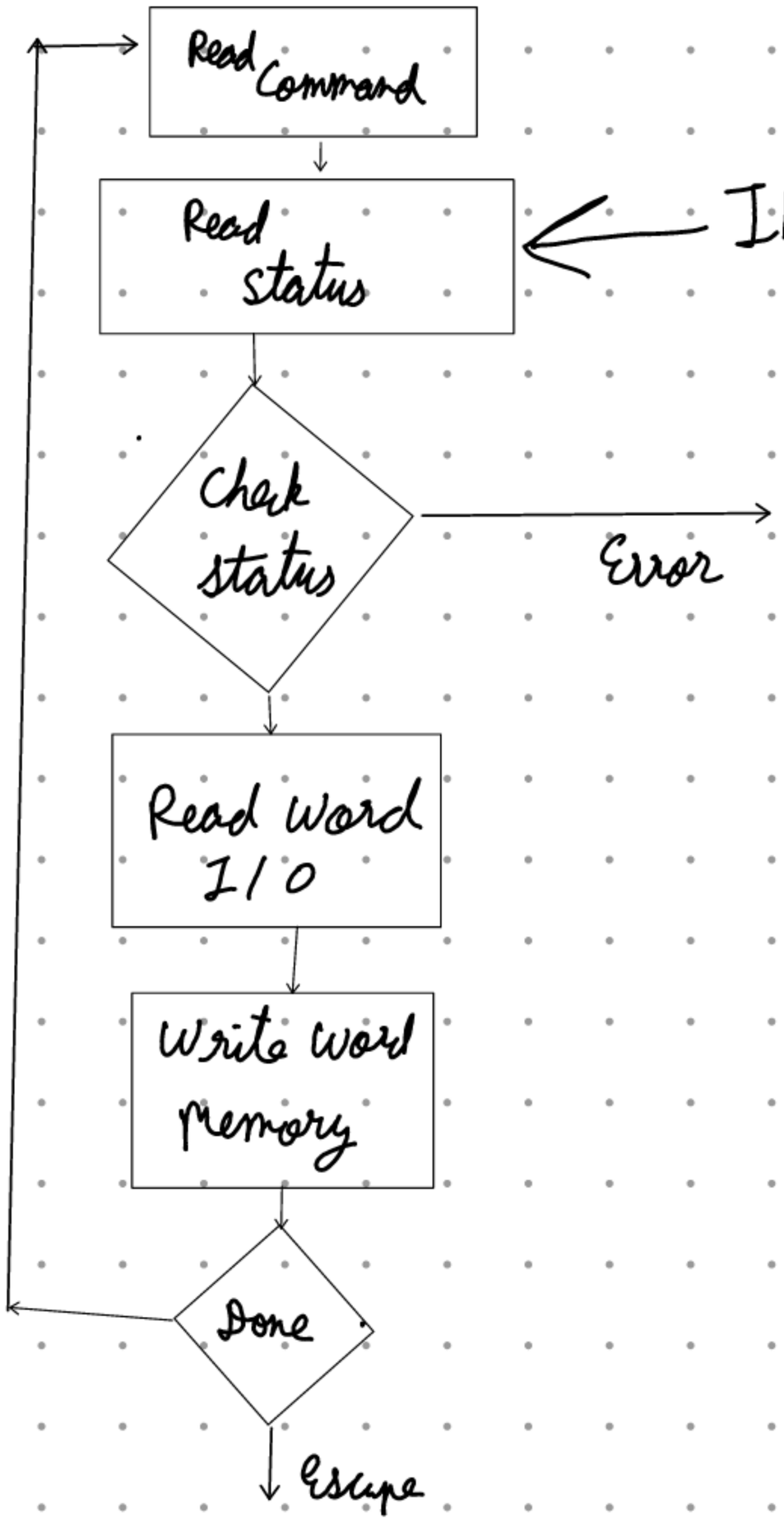
iii) Wastes CPU time

iv) Everything is done by CPU

v) I/O Module does not speak with CPU directly

vi) CPU may wait or come back later

② Interrupt driven I/O



① CPU sends Read Command

② I/O sends Interrupt when it finishes

③ CPU doesnt need to check everytime

③ DMA

DMA Controller does all work

(A) Burst Mode

CPU stops while DMA sends all data in a burst to Memory

(B) Cycle stealing Mode

sends data to memory anytime can stop CPU

(C) Transparent DMA

Sends data to Memory when CPU is not using.

DMA Causes Cache Coherence Problem

(4) IO Mapped IO vs Memory Mapped IO

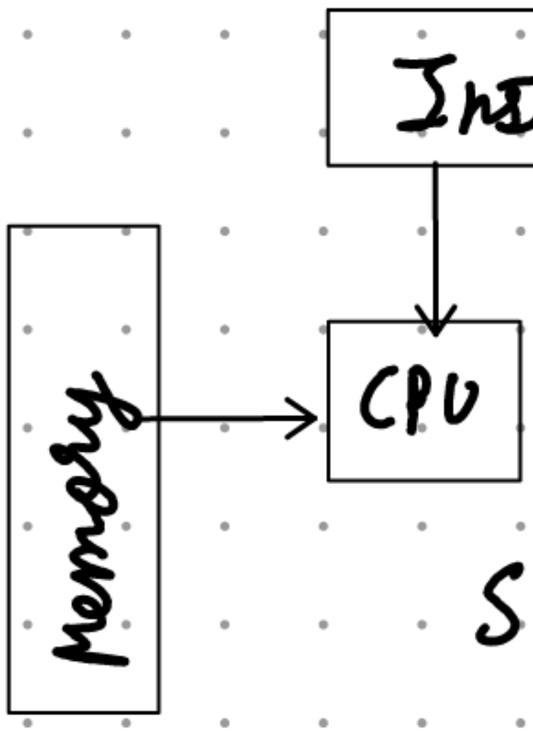
IO Mapped IO → seperate adress line for IO

Memory Mapped IO → Same address lines as IO

Module 6

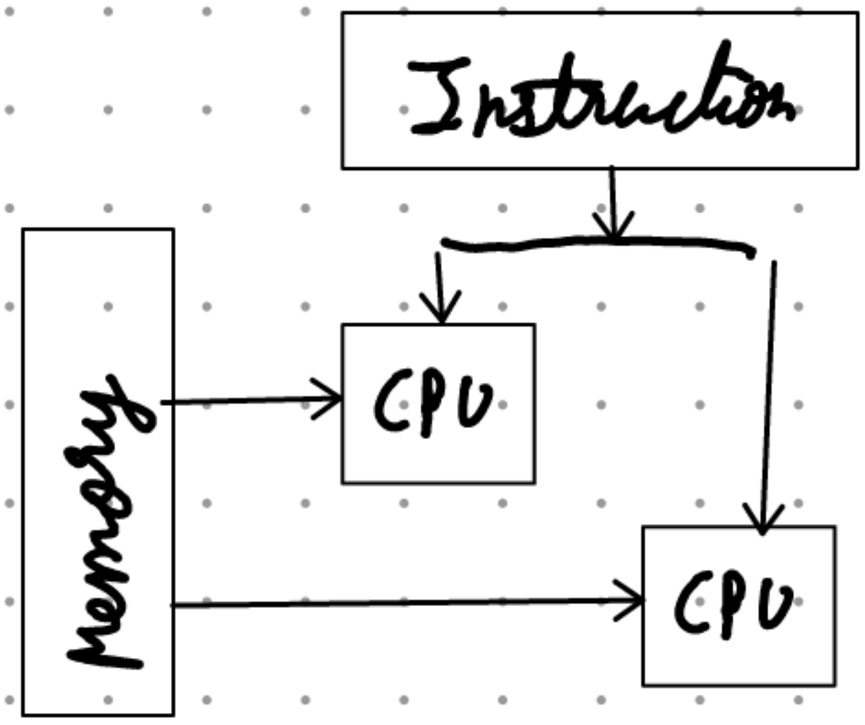
- ☆ Flynn classification
- ☆ CISC Pipelining

① Flynn classification



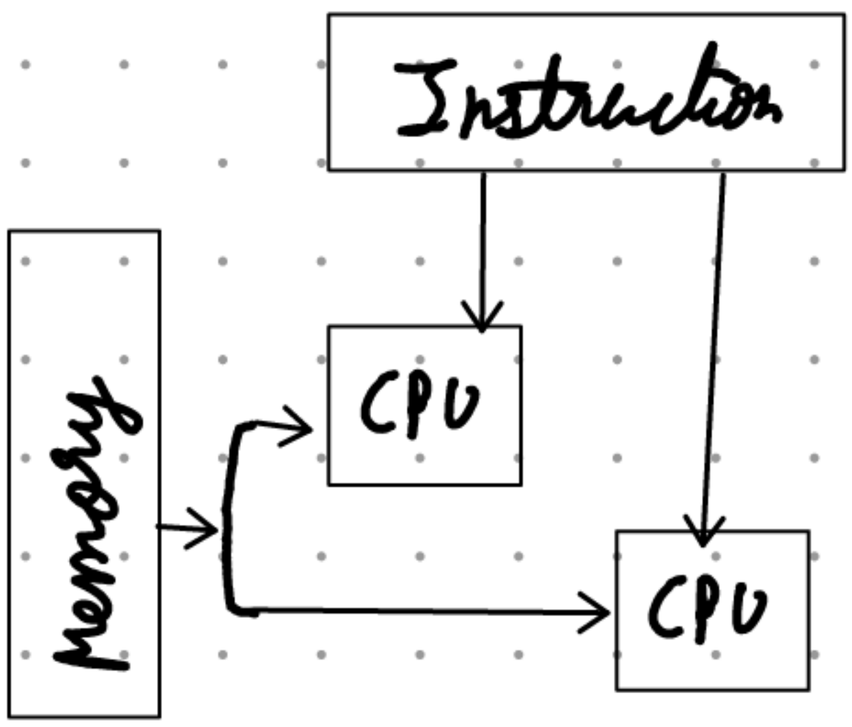
SISD

Von Neuman



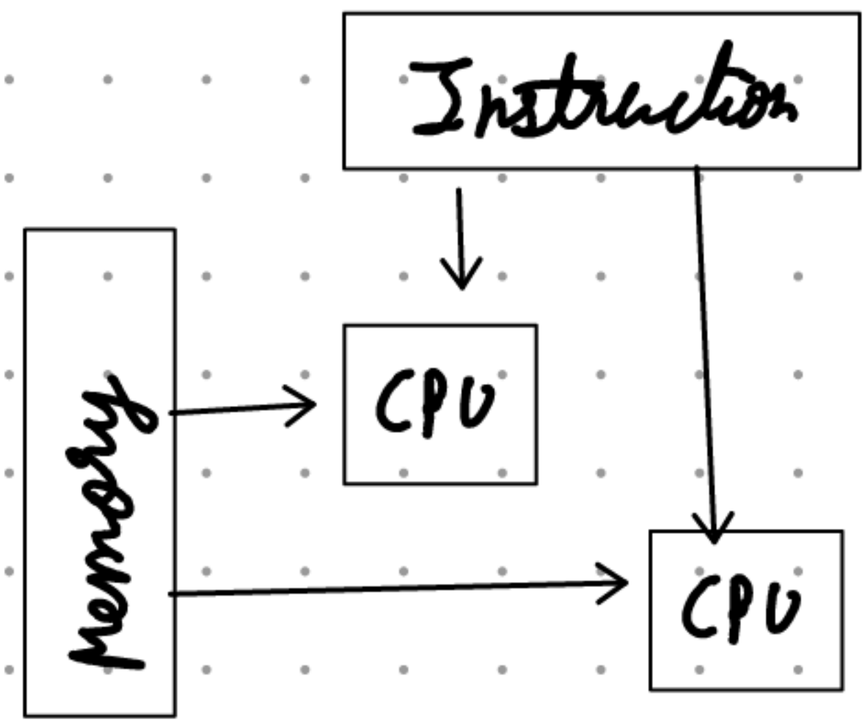
SIMD

eg vectors addition



MISD

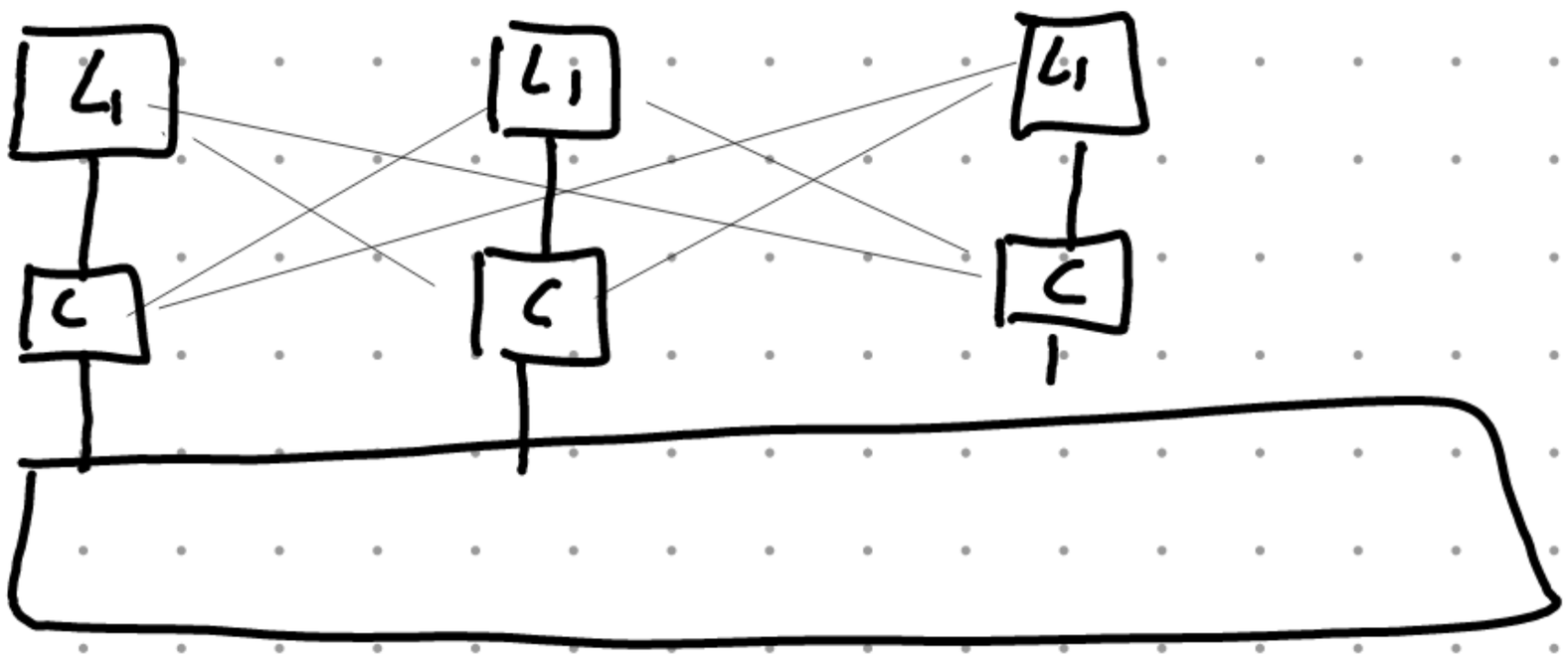
Never used



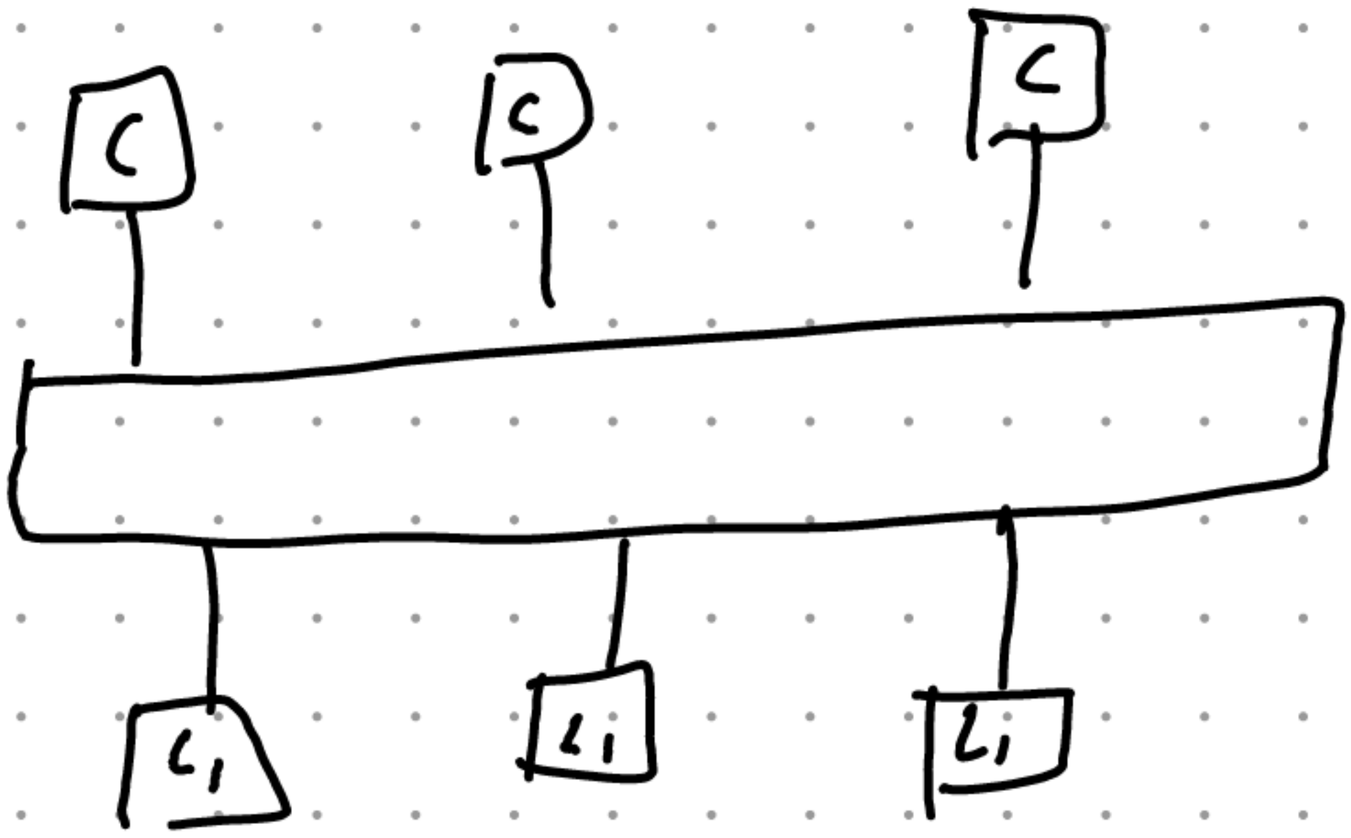
MIMD

used nowadays

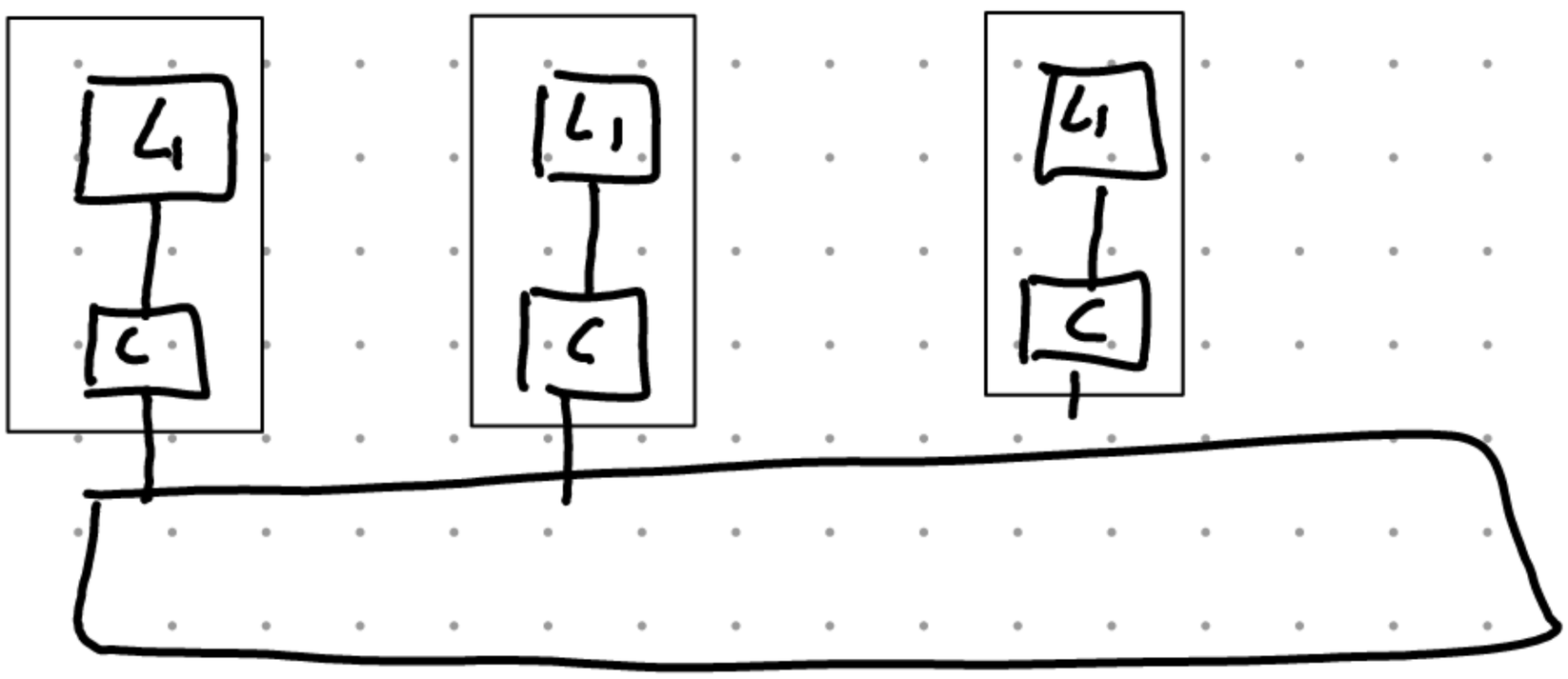
② Types of MZMD



NUMA



UMA



Distributed

③ CISC Pipeline

Fetch Instruction

Decode Instruction

calculate Instruction

Fetch operand

Execute Instruction

Write Operand

Types of hazard → Resource

→ Data

→ Control