Embedded Systems

Computer system on a non-computing device

Real time constrain: there is a deadline that the action should stick to:

To ~~easy~~ understand any E.S

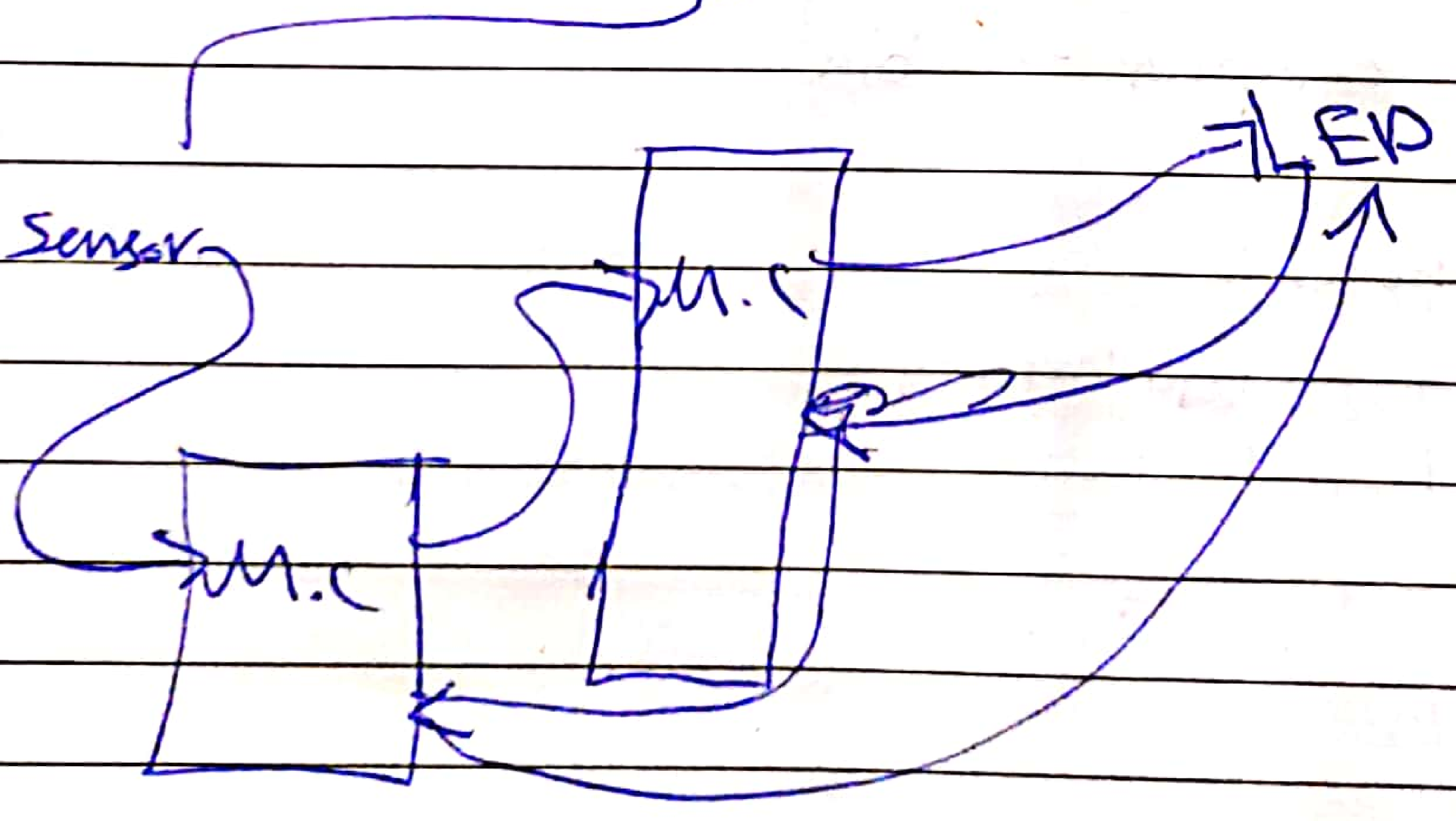
1- Input and output

2- User interaction

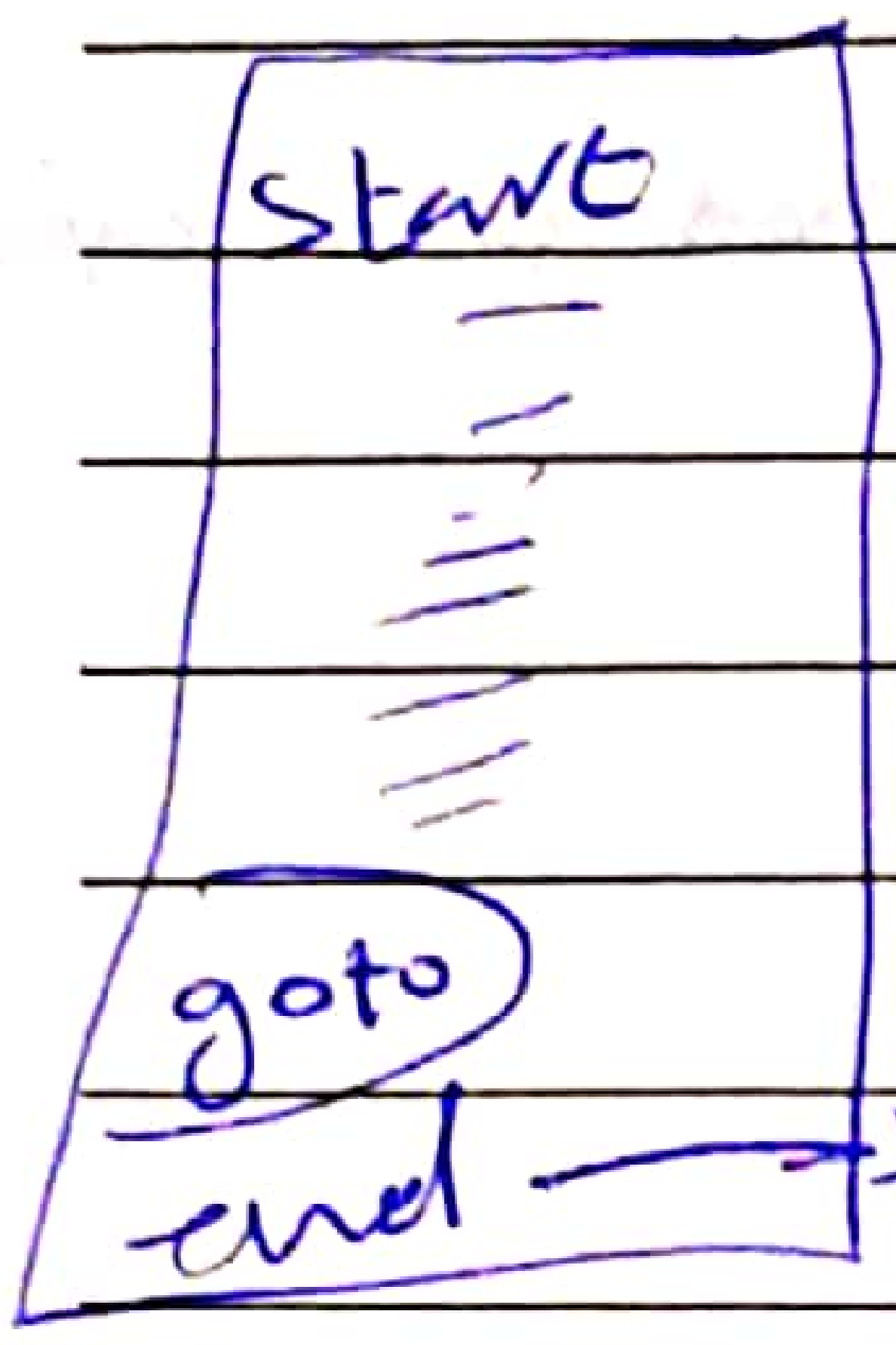
3- Link to other devices

4- Hardware

5- Code
- you have access to the code → backup then edit.
- you don't have access to the code



Micro Controllers of the same family have the same core and the same instruction set. slide 22 software driven : everything in the E.S is controlled by the code . (always running)



→ shouldn't reach the end

Computer components :

CPU / busses / I/O / Memory .

less writing cost in power and time

→ it holds values of variables (x, a)

→ DATA Memory (RAM) : volatile (lost when power is down)

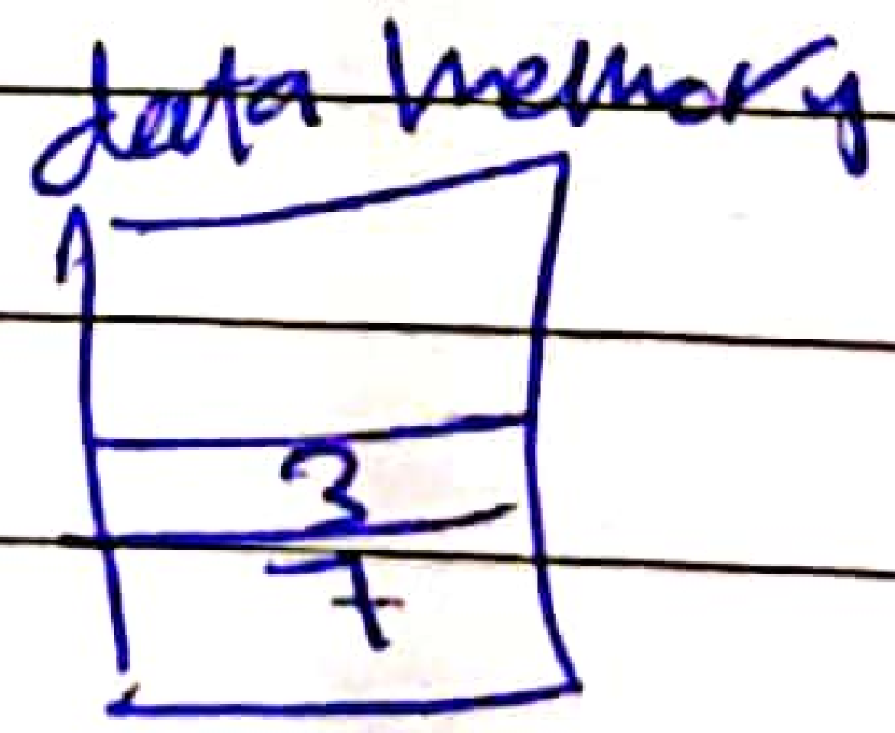
→ Program Memory (ROM) Permenant . (.bin) holds the code

compilation : compiler → .bin

1) checks for errors

2) Generates machine code

```
.cpp
void main()
int x , a ;
Cout ;
Cin ;

X = a * 2 ;
→ power down
  5
}
```



Computer's components

CPU
I/O
busses

Memory ——— data memory : volatile , holds value of variables , faster, less power in writing

prog memory : permenant ; holds instructions

On power up ; the CPU fetches the first instruction from a defined address , this address is called the reset vector

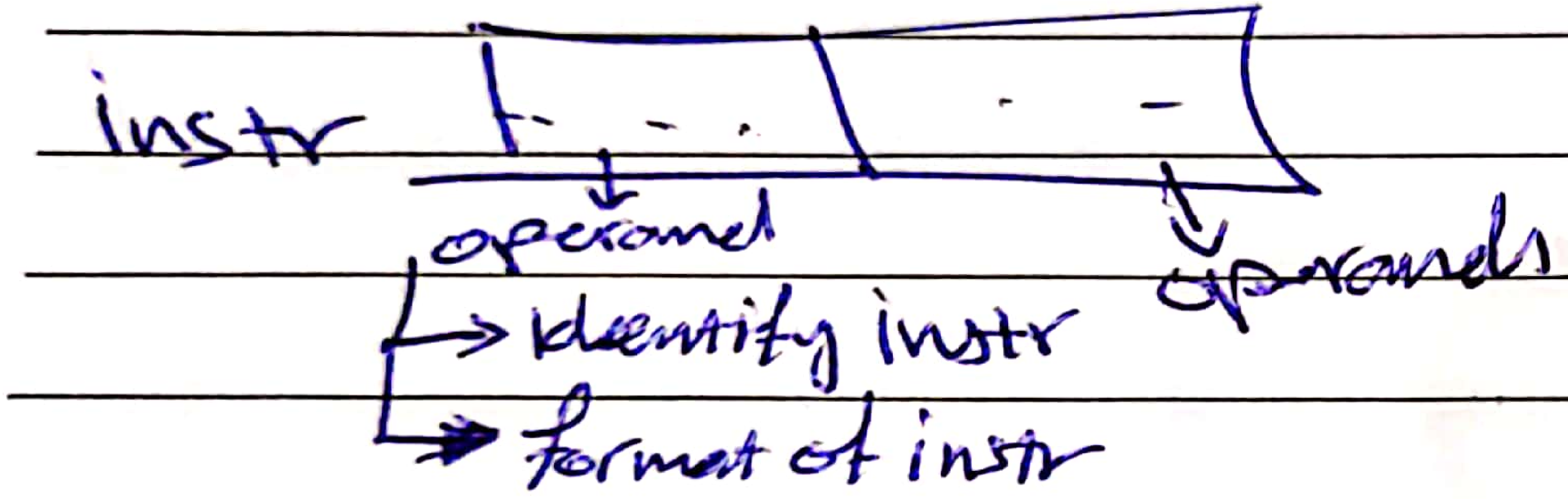
Reset vector is the address of the first instruction to be executed on power up & reset

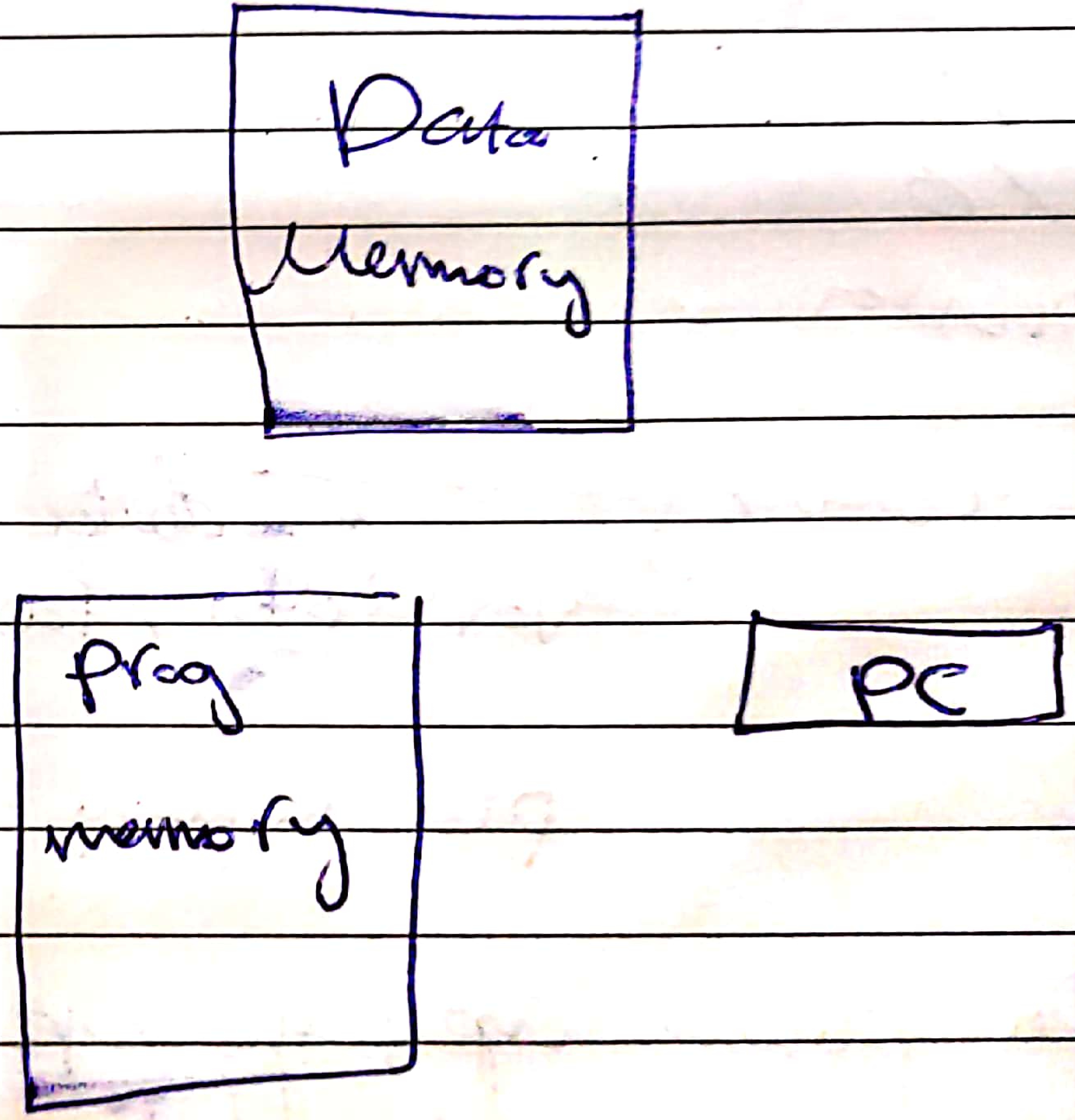
While the CPU executes the current instr , the next instr is being fetched (pipelining

* Program Counter (PC) : holds the address of the next instruction to be executed

* PC is auto incremented

PC : reset vector (0000) ⟶ On power up i[illegible]


instr
operand
operands
identify instr
format of instr


Data
Memory
Prog
memory
PC

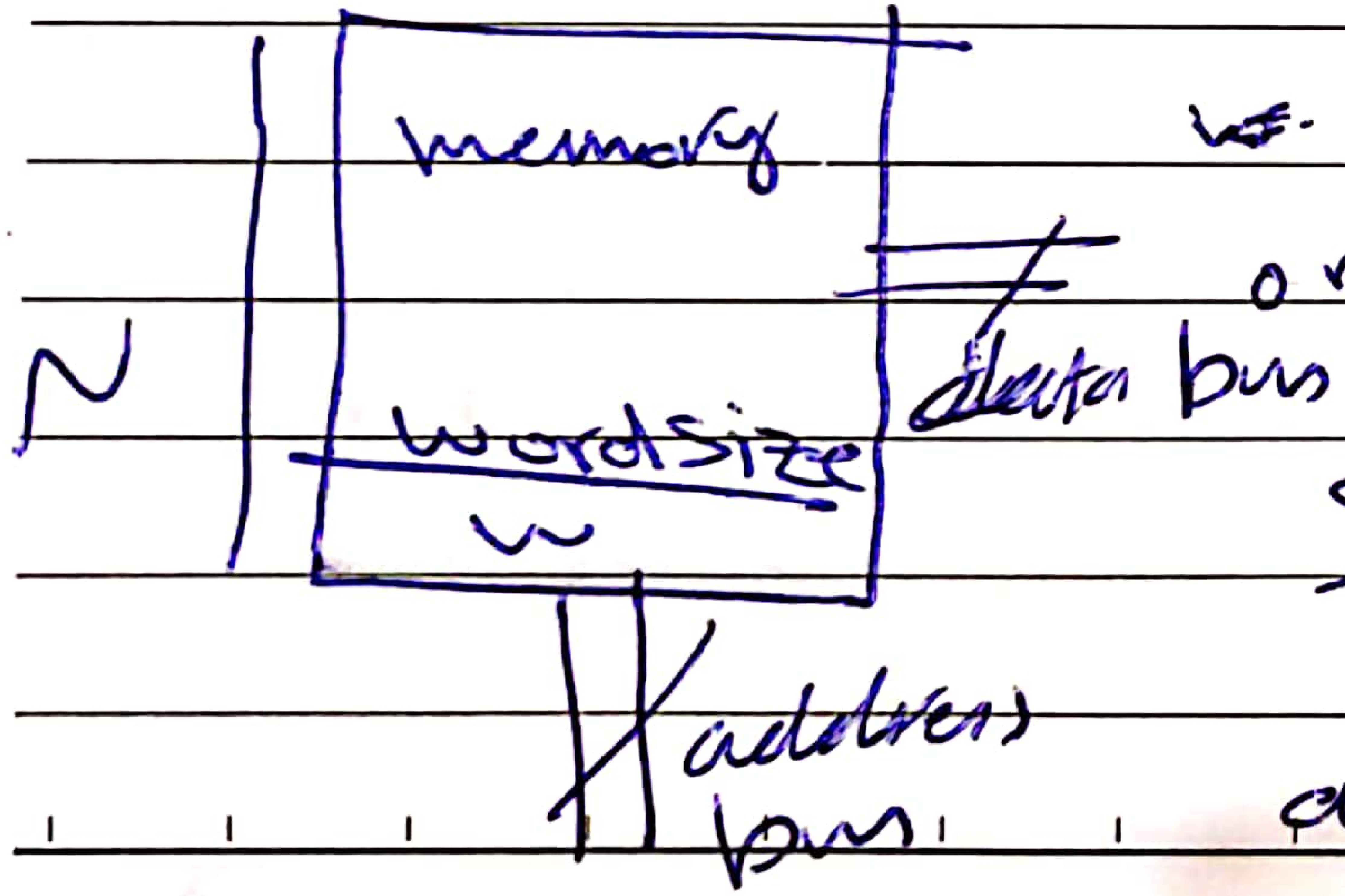
Van neumann Arc :

2 buses for all

less complexity due to less buses

Harvard: 2 buses for each module

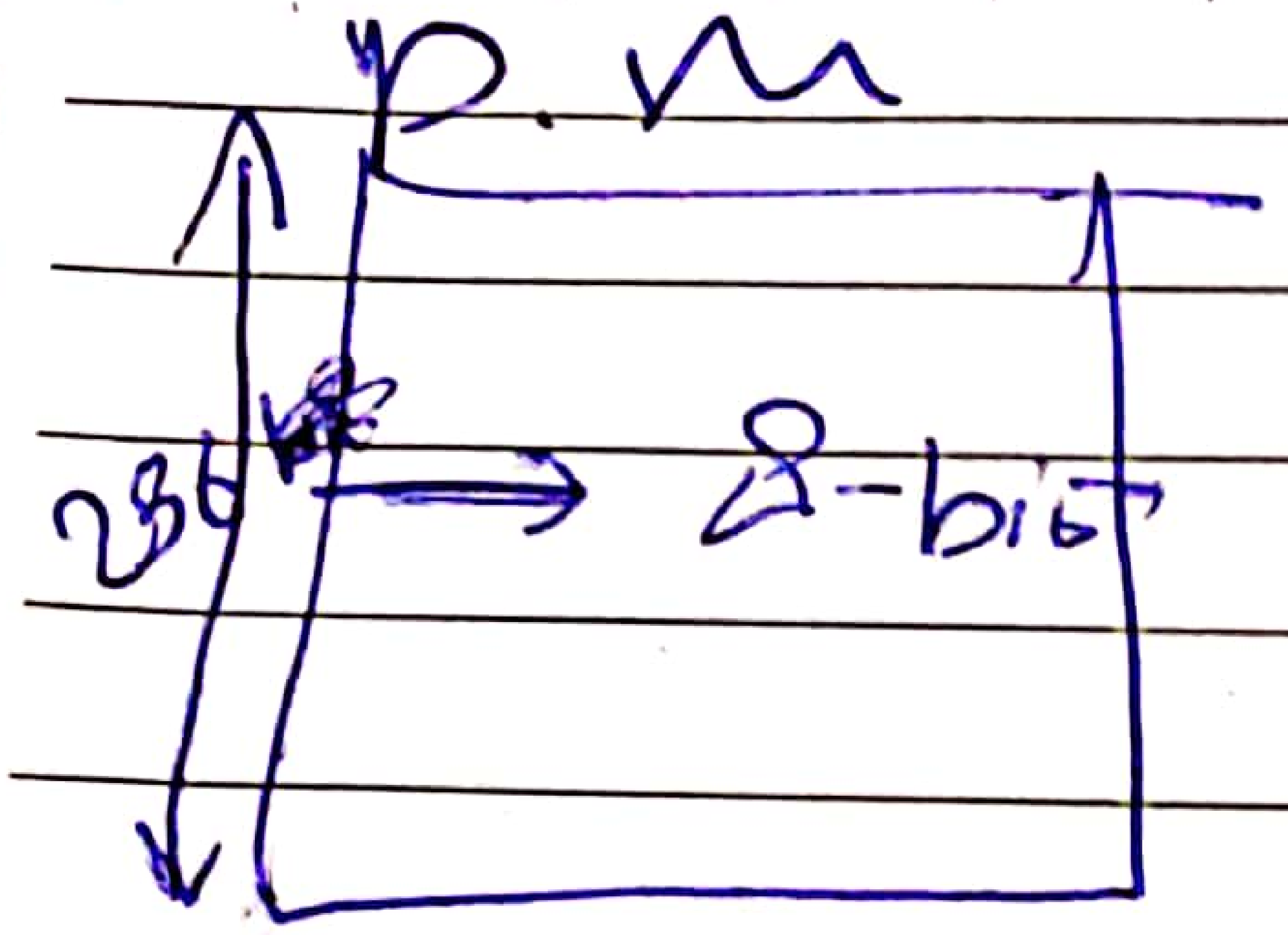
faster due to pipelining

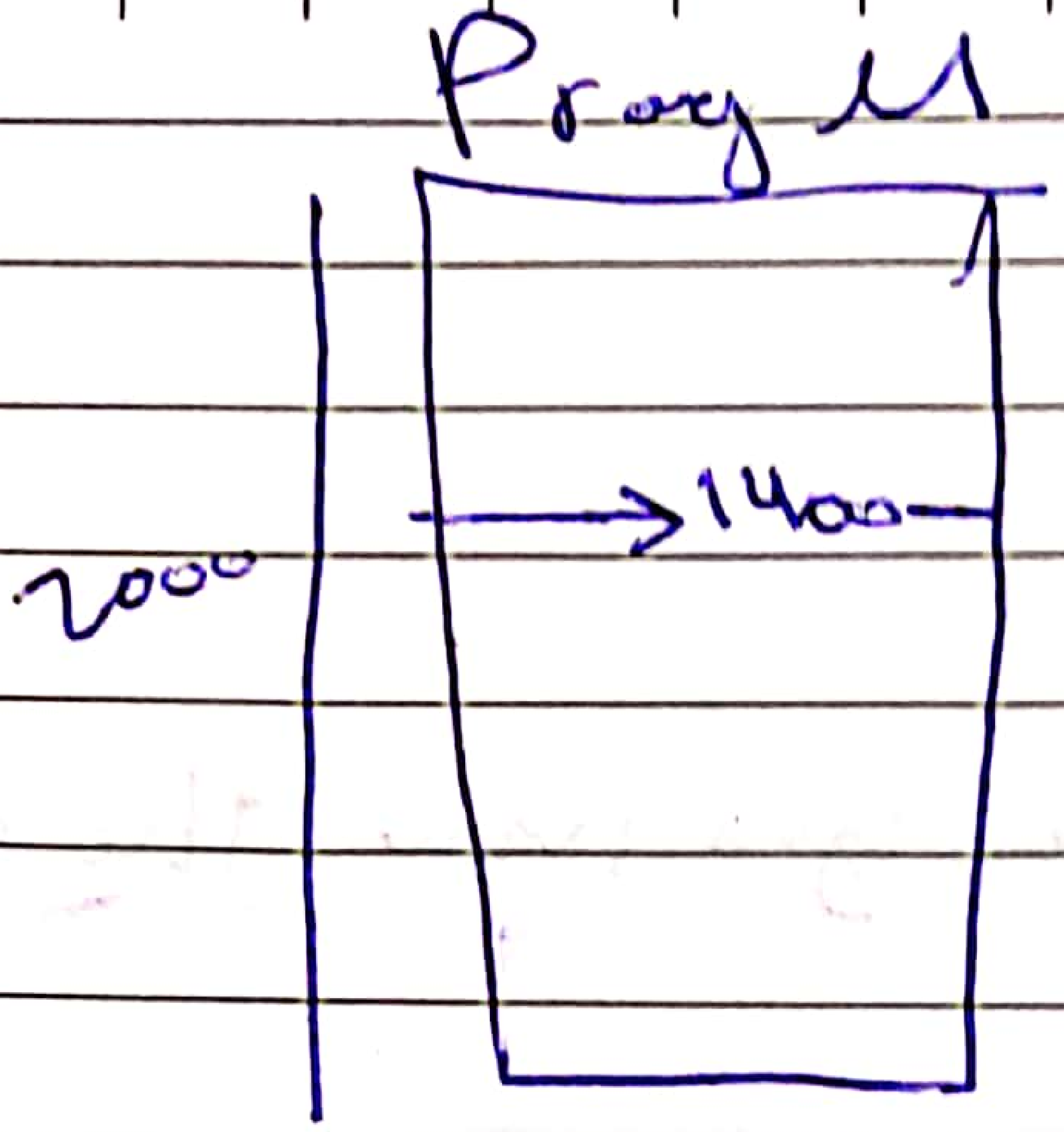


word size: when you write a value or read it then you deal with w-bits

Size of memory = $w * N$

address bus $\geq \lceil \log_2 N \rceil$





Data Memory

Prog Mem

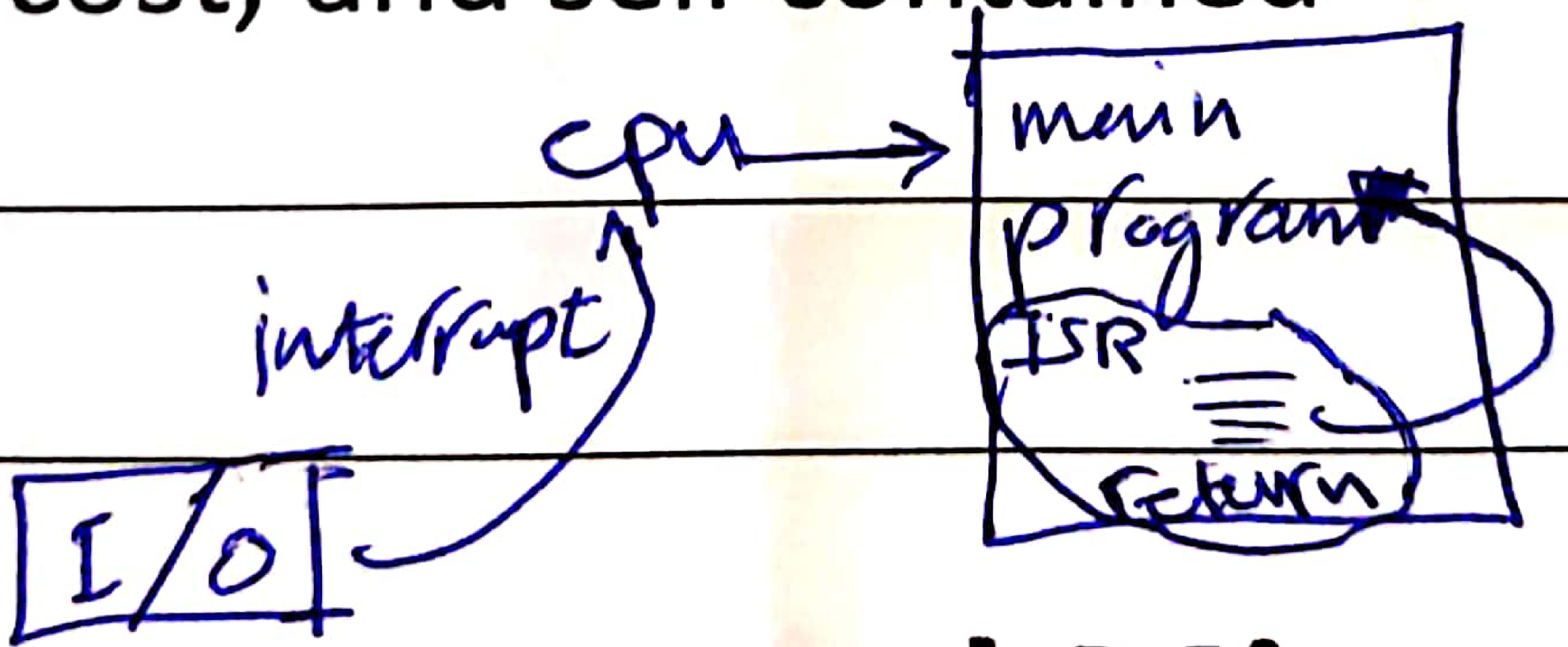
Instruction Set : detailed description for each instr we can use to write programs

CISC : too many types of instructions, more functionality
many addressing modes, with less prog time

RISC : few Simple Instrs.
longer code
faster execution time due to more optimised pipelining

more self-contained, powerful, and faster

- **A special category of microprocessors emerged**
  - Microcontrollers
  - Intended for control purposes
  - *No high computational power, huge memories, or high speed is required*
  - *Has excellent I/O capabilities*
  - Small, low cost, and self contained

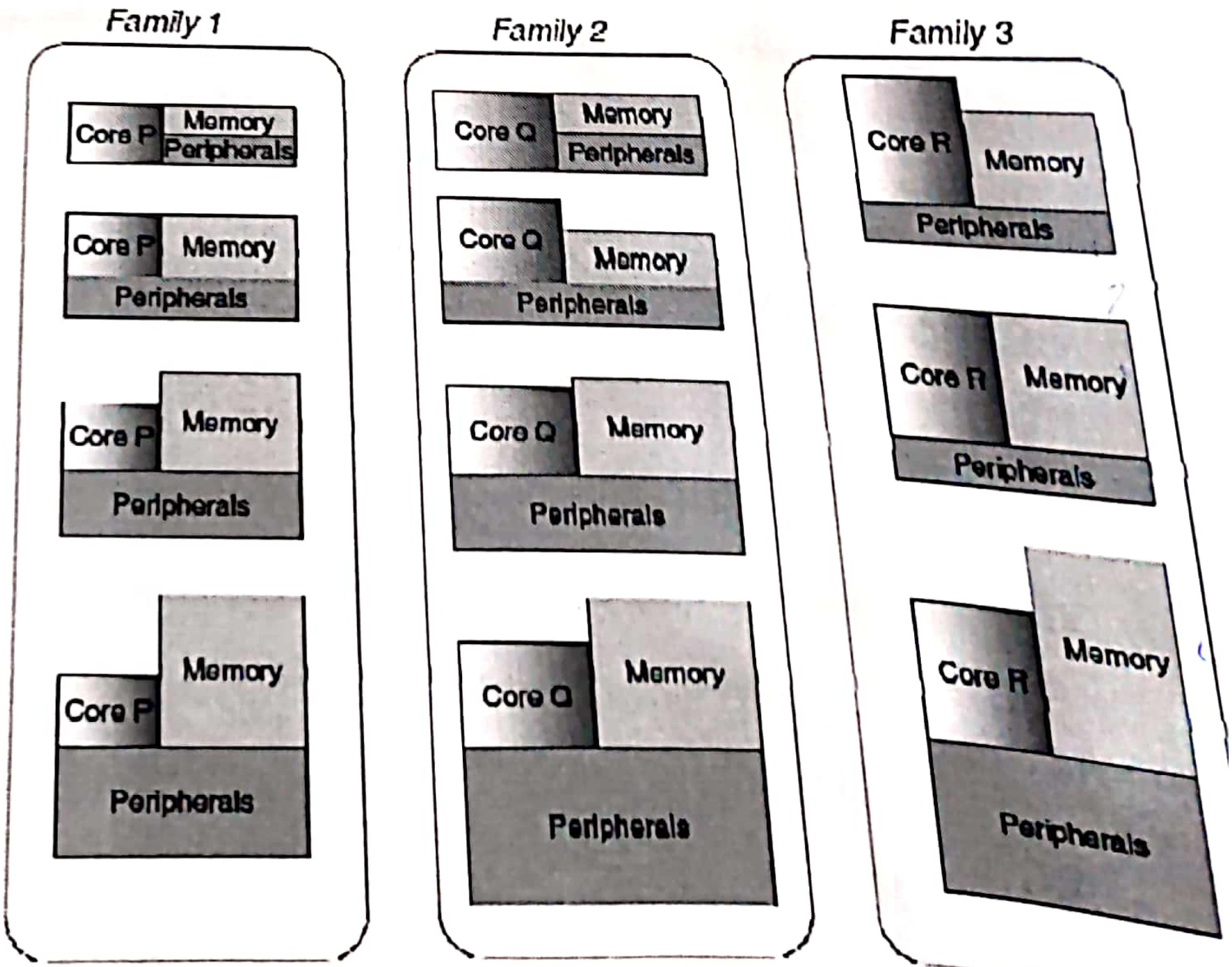


Microprocessors and Microcontrollers

Microprocessors and Microcontrollers

- Microcontroller Families
  - Different families with *each family built around the same core*
  - Family members differ in *memory size* and peripheral capabiliti

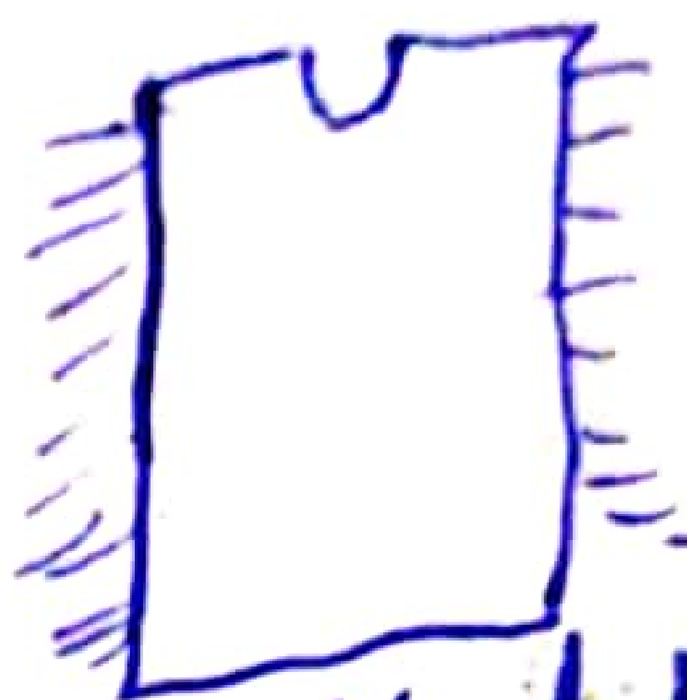
same family
= same core
& Same ISA



Scanned with CamScanner

Microprocessors and Microcontrollers

- Microcontroller Packaging
  - Plastic packaging
  - Pins for I/O, clock, communication, and Power.
  - The number of pins usually determines the size of the chip



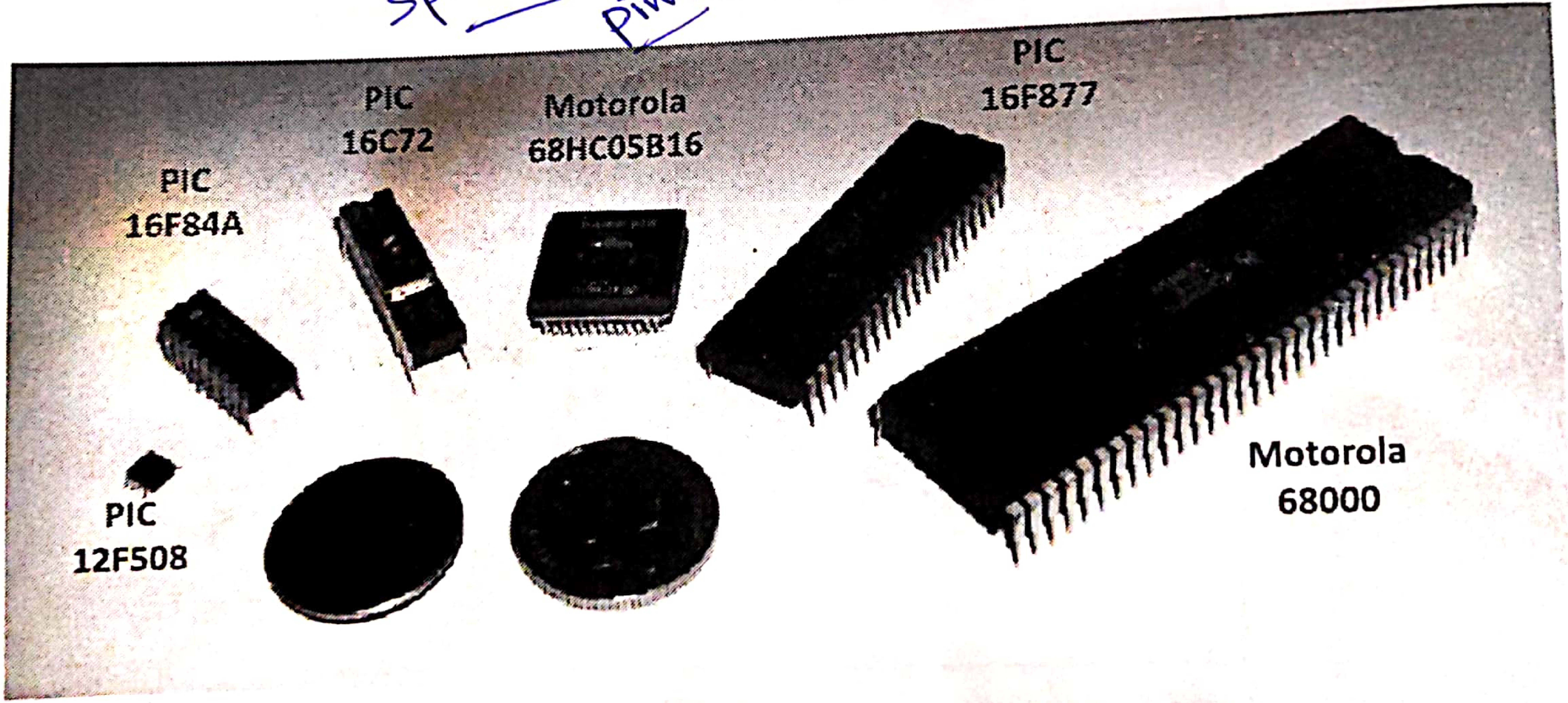
dual inline packaging

called DIP

D

D

space between pins



Microchip and the PIC Microcontrollers

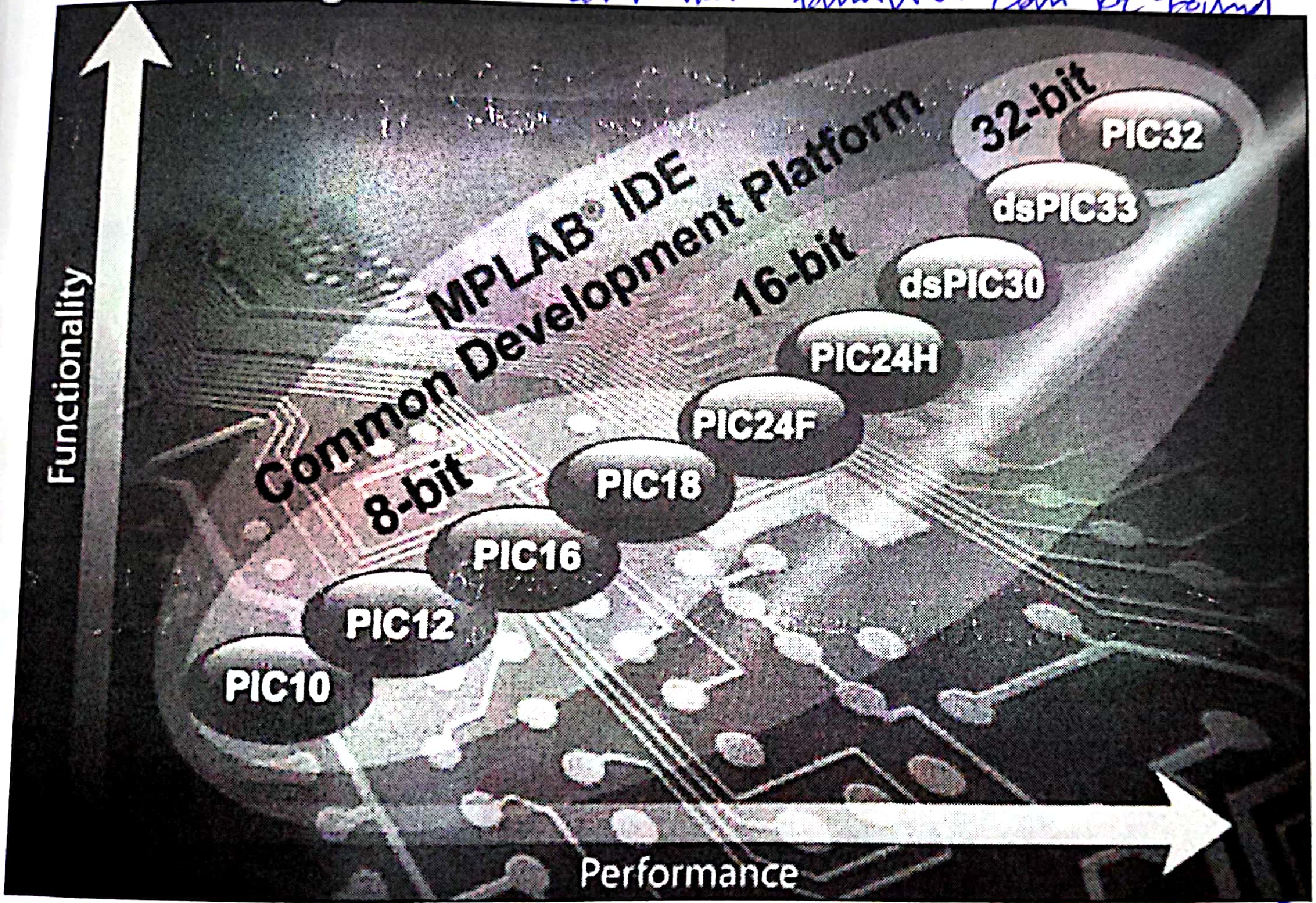
- Peripheral Interface Controller (PIC) was originally a design by General Instruments intended for simple control applications

- In the late 1970s, GI introduced PIC® 1650 and 1655
  - Standalone design
  - RISC with 30 instructions
  - Single ? working register (accumulator): holds the result of the last instruction
  - Many attractive features

- PIC was sold to Microchip

Microchip and the PIC Microcontrollers

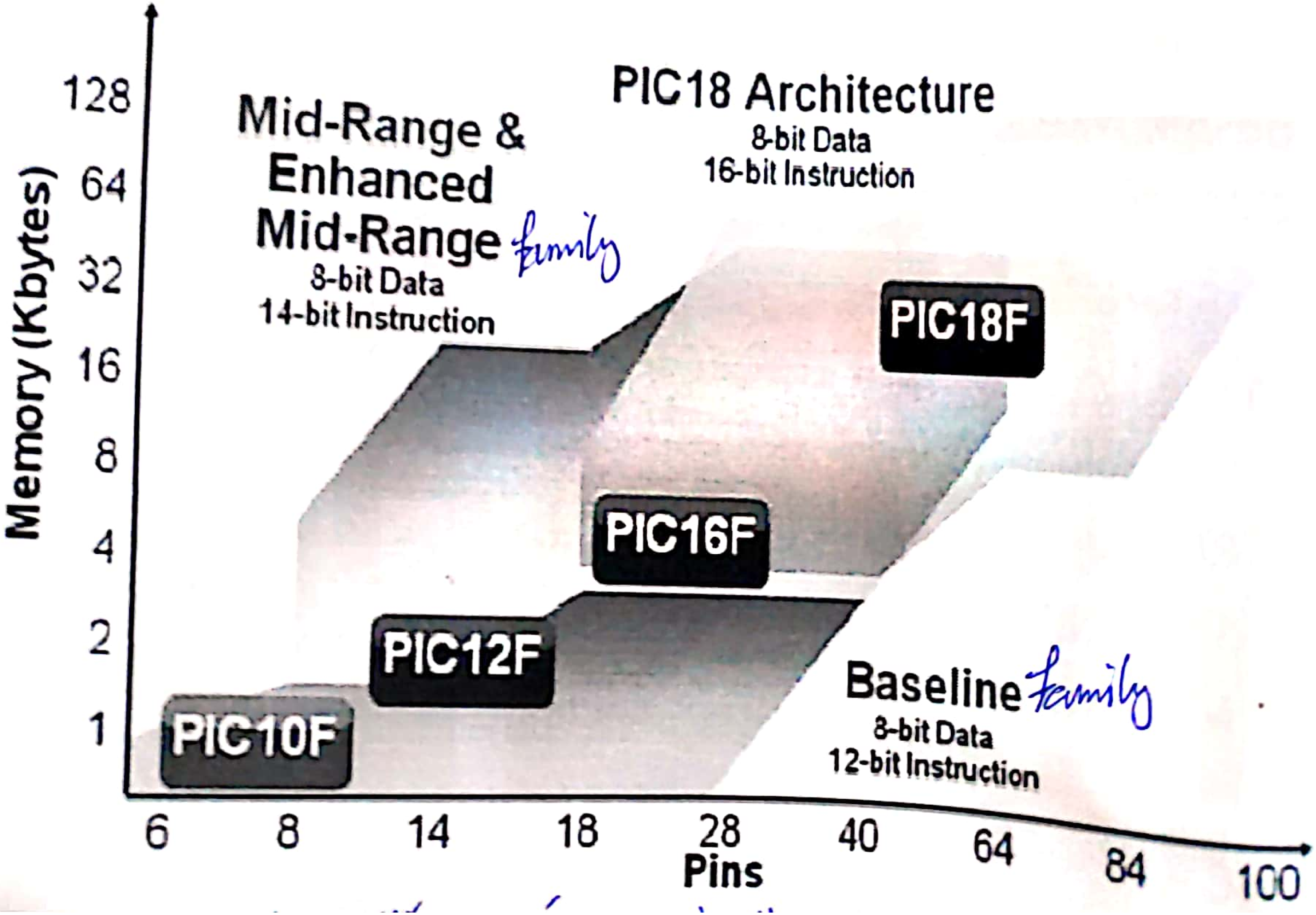
Microcontroller's families can be found in the datasheet



8-bits M.C : the size of the data memory size is 8-bits for all variables => all variables are 8-bits in size

Microchip and the PIC Microcontrollers

Families



low power

L

MCs name in the given series

don't care

series ← 16 F 87X A → advanced technology

C

CMOS

incorporates flash memory

for each MC one is with A last ⇒ advanced

Microchip and the PIC Microcontrollers

- **PIC Families**

stack: volatile memory automatically written on and read from;
→ it holds the return address (~~saved~~ for if we have an interrupt)

PIC Family	Stack Size (words)	Instruction Word Size	No. of Instructions	Interrupt Vectors
12CX/12FX	2	12- or 14-bit	33	None
16C5X/16F5X	2	12-bit	33	None
16CX/16FX	8	14-bit	35	1
17CX	16	16-bit	58	4
18CX/18FX	32	16-bit	75	2

call subroutine → push PC ~~4~~ of the next instr to stack
⇒ PC = address of 1st instr of subroutine

- Example: the 16C84 was the first of its kind built using CMOS technology. It was later reissued as 16F84A incorporating flash memory and other technological features

N levels of stack: N nested calls

nested call: do a call before returning from Previ
call

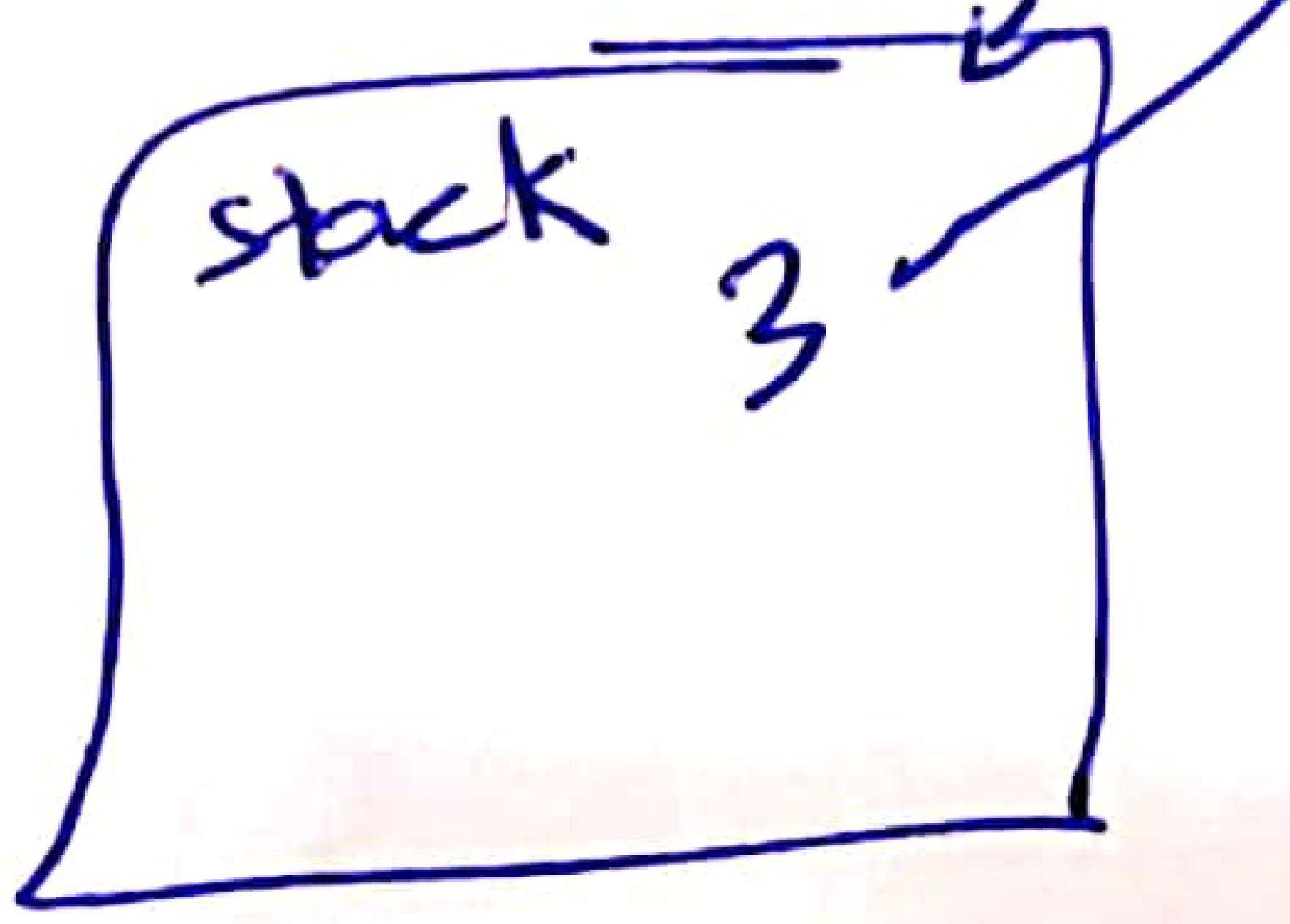
Microchip and the PIC Microcontrollers

- **PIC Characteristics**
  - Low-cost
  - Self-contained
  - 8-bit
  - Harvard architecture
  - RISC
  - Pipelined
  - Single accumulator (the working or W register)
  - Fixed reset and interrupt vectors ⇒ can't be changed.

0 → instr 1
1 → instr 2
2 → call ~~1~~ sub1 PC=100
3 → instr 3
4 → call Sub2
5 → call Sub3

100 sub1
105 return PC=106 initially
200 sub2 then pop
205 return
300 sub3
305 return

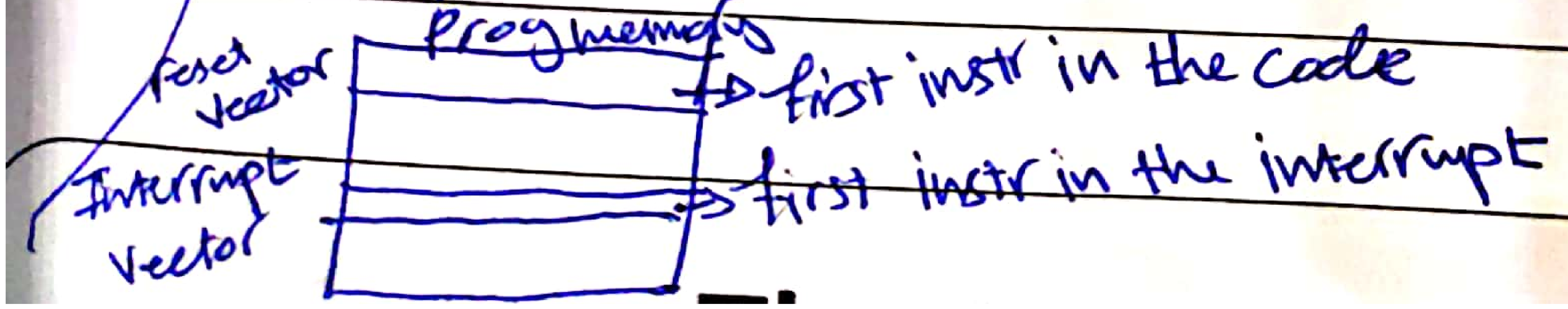
PC=3



we need at least 2 levels of stack if we want to do nested function calls

Reset Vector: the address that is automatically loaded in the uC on power up

Interrupt Vector: // // // // // // // // // // //

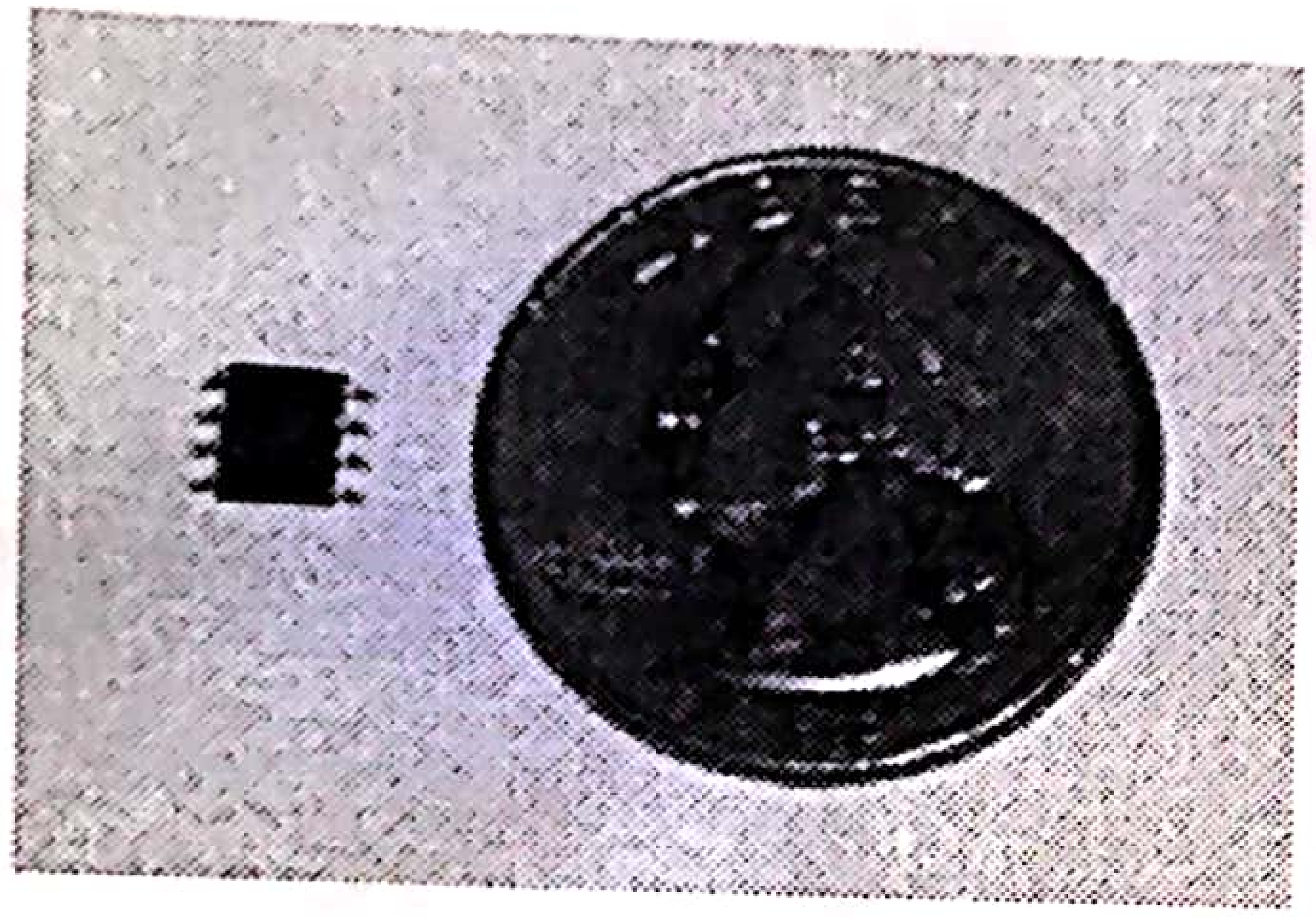


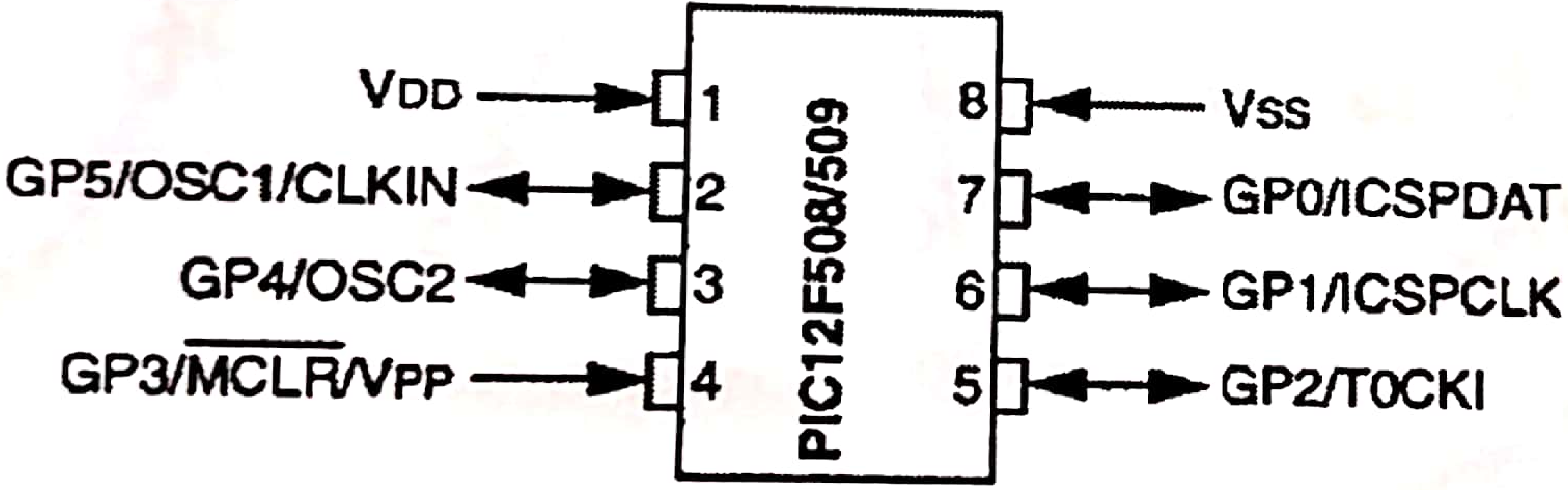
→ first instr in the code
→ first instr in the interrupt

The PIC 12 Series

مش ح يشرحه الدكتور

- PIC 12F508/509
- The smallest and simplest PIC





Key

V_{DD}:	Power supply	V_{SS}:	Ground
V_{PP}:	Programming voltage input	MCLR:	Master clear
OSC1, OSC2:	Oscillator pins	CLKIN:	External clock input
GP0 to GP5:	General-Purpose input/output pins (bidirectional except GP3)		
CSPDAT:	In-Circuit Serial Programming™ data pin.		
CSPCLK:	In-Circuit Serial Programming™ clock pin.		

mid range family : 8-bits M.C , RISC, Harvard

EEPROM stores permenent variables for holding settings

for I/O max 13-b

Some members of the PIC 16 Series family

mid range family

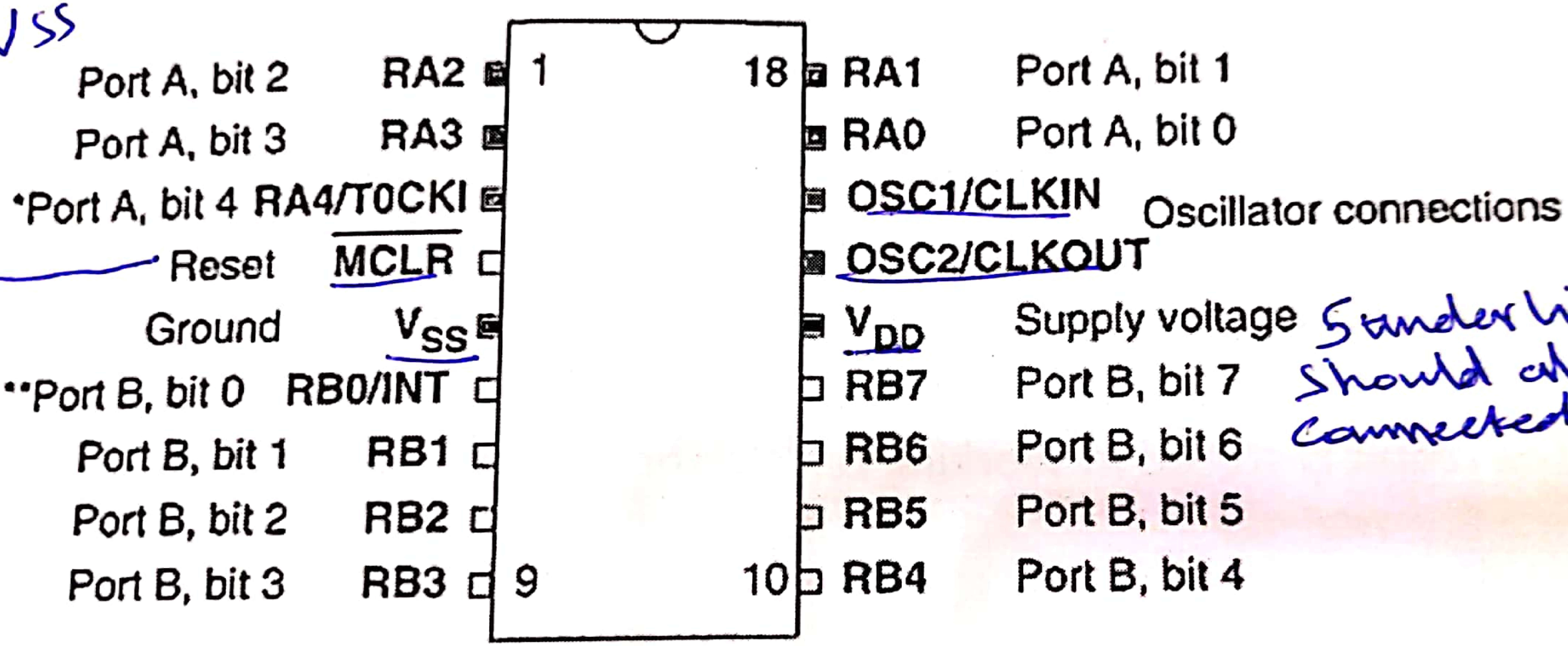
Device number	No. of pins*	Clock speed	Memory (K = Kbytes, i.e. 1024 bytes)	Peripherals/special feature
16F84A	18 not all for I/O	DC to 20 MHz	1K program memory. Code perme 68 bytes RAM. 64 bytes EEPROM	1 8-bit timer 1 5-bit parallel port port A 1 8-bit parallel port port B
16LF84A	As above	As above	As above	As above, with extended supply voltage range
16F84A-04	As above	DC to 4 MHz	As above	As above
16F873A	28	DC to 20 MHz	4K program memory	3 parallel

An Architecture Overview of the 16F84A

→ when = ∅ ⇒ reset. (PC ← reset vector)

VDD

VSS



: for clock

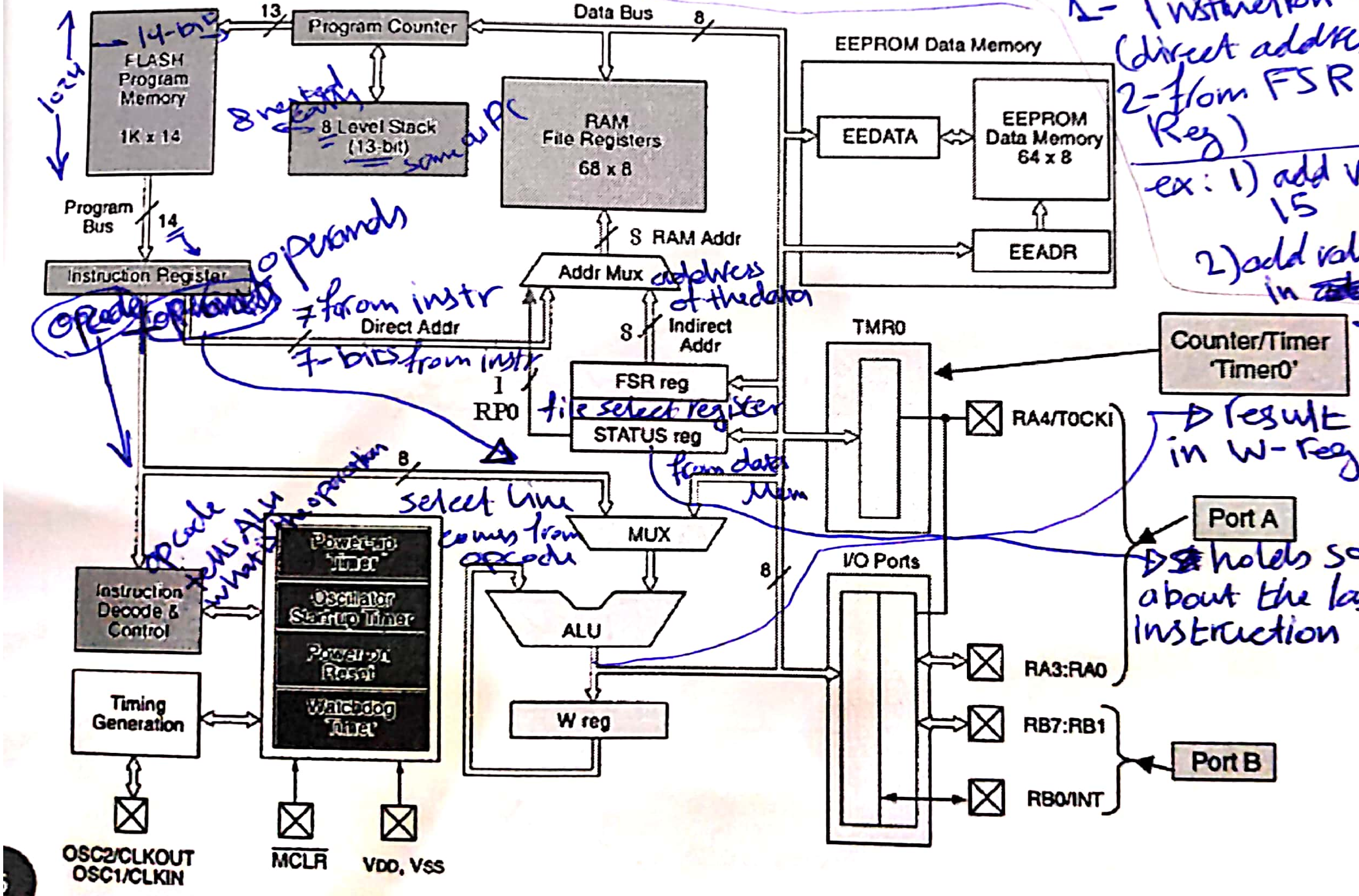
S underlined pin should always be connected

*also counter/timer clock input

**also external interrupt input

- 18 Pins / DC to 20MHz / 1K program Memory/ 68 Bytes of RAM / 64 Bytes of EEPROM / 1 8-bit Timer / 1 5-bit Parallel Port / 1 8-bit Parallel Port

An Architecture Overview of the 16F84A



⊳ from DM: address is either in
1- instruction itself (direct address)
2- from FSR (file select Reg)

ex: 1) add value at addr 15
2) add value stored in ~~address~~ FSR

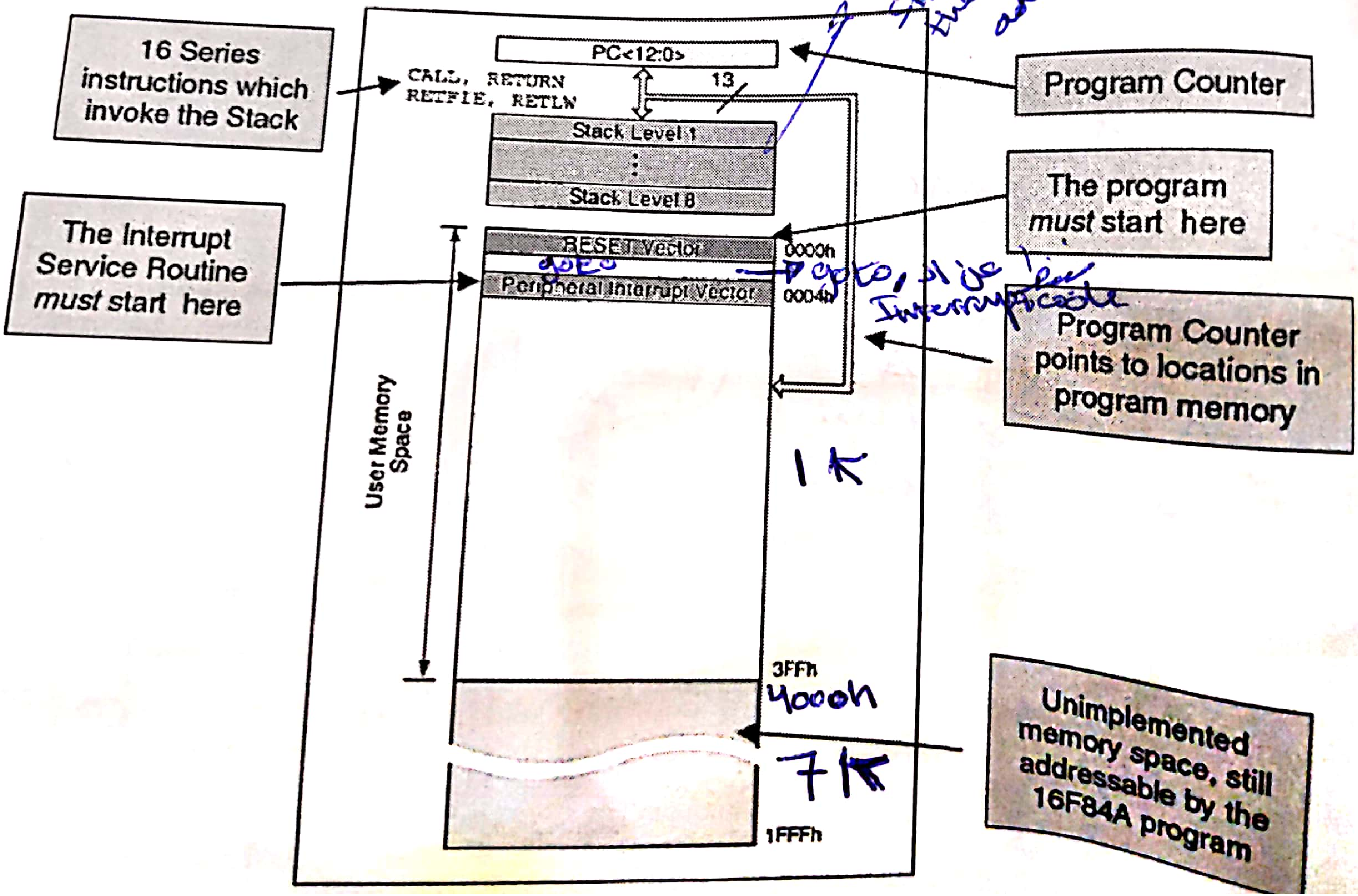
⊳ result is stored in W-reg or in DM

⊳ ~~S~~ holds some informa about the last executed instruction

any instr that requires two variables, one of them comes from W-Reg & the second is either from 1) instr (Literal), 2) from Data Memory

The PIC 16F84A Memory Organization

- *Program Memory and Related Units*



16 Series instructions which invoke the Stack

The Interrupt Service Routine *must* start here

Program Counter

The program *must* start here

Program Counter points to locations in program memory

Unimplemented memory space, still addressable by the 16F84A program

The PIC 16F84A Memory Organization

- ***The Configuration Word*** holds some configuration instructi about the prog.
  - A special part of the program memory
  - Allows the user to configure different features of the microcontroller at the time of program download and is not accessible within the program or while it is running

→ to modify we have to redownload the code

R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
bit13													bit0

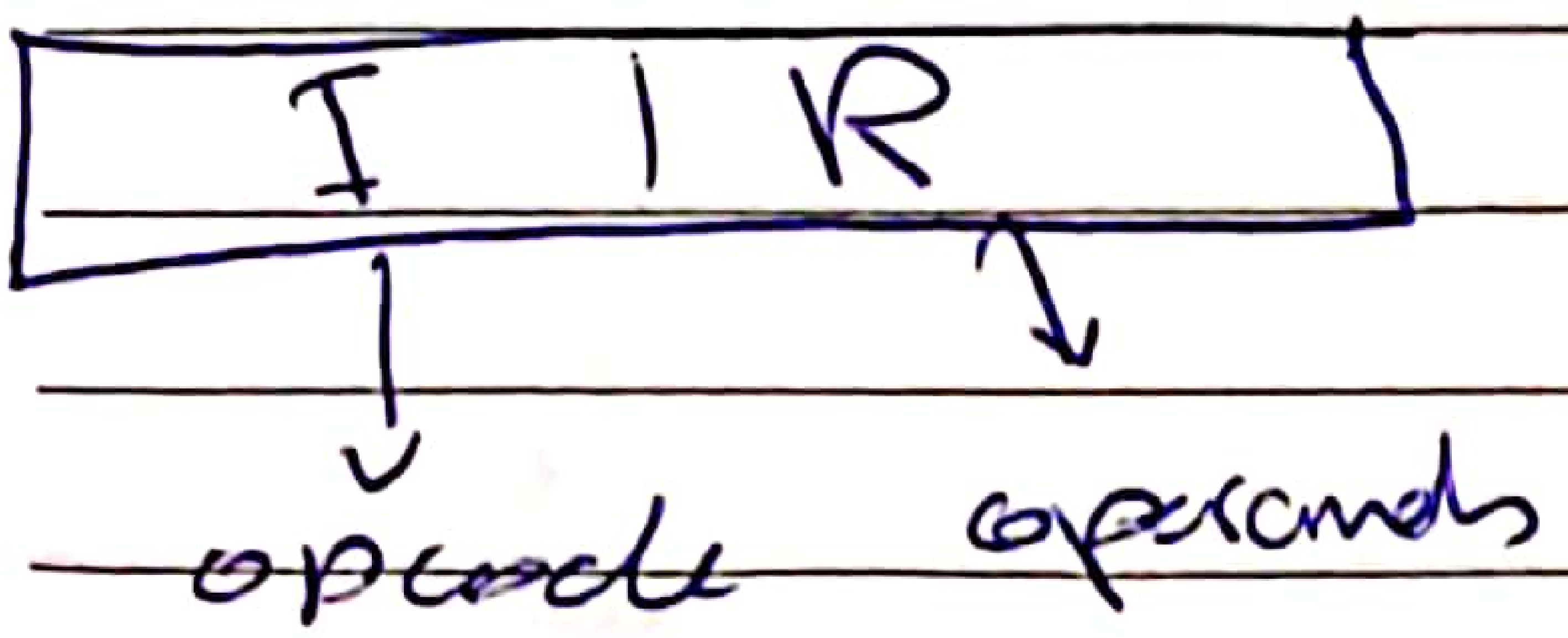
bit 13-4 **CP: Code Protection bit** if = 0 then your code is protected
1 = Code protection disabled
0 = All program memory is code protected

bit 3 **$\overline{\text{PWRTE}}$: Power-up Timer Enable bit** : if equals 0, on power up, keep M.C in reset mode for a while
1 = Power-up Timer is disabled
0 = Power-up Timer is enabled

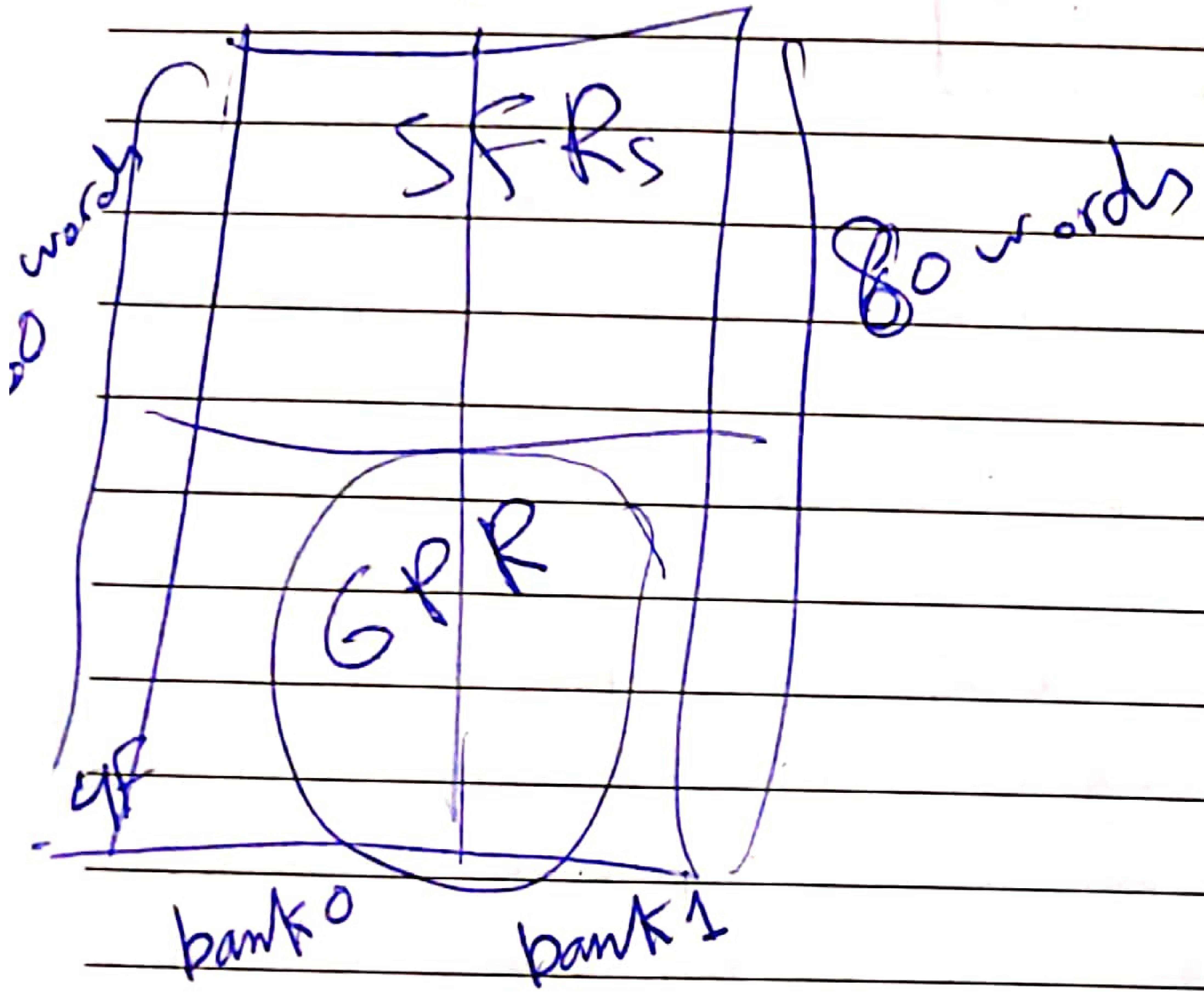
bit 2 **WDTE: Watchdog Timer Enable bit** : it resets M.C on Crash
1 = WDT enabled
0 = WDT disabled

bit 1-0 **FOSC1:FOSC0: Oscillator Selection bits** configuration word Specifies the type of osc to be connected.
11 = RC oscillator
10 = HS oscillator high speed
01 = XT oscillator crystal
00 = LP oscillator low power

Instr memory



addressing ~~modes~~: how you name the operands



$80 + 80 = 160 \Rightarrow \lceil \log_2 160 \rceil = 8$ bits to addres any word in a memory of size 160 words

* when we divide the memory into 2 banks, each of size 80 words
* to address any word in a given bank

$\lceil \log_2 80 \rceil = 7$ bits

* we still need a bit to select the bank

to access any word in the memory

1-bit ← 1) choose the bank if not chosen before

7-bits ← 2) choose the address in the bank

the benefit ⇒ you save one bit from the address to any word in the data memory

*cost : additional steps to select the bank if not already selected.

* If the instruction containing an address to the data memory, word ⇒ the address is 7 bits & the 8th bit comes from the status register

Ex:

add 85 → should be 7 bits ⇒ compiler discards the leftmost bit

1000 0101 8

8-bits

assume that there exists an instruction called RD ad which reads the data memory word of address "ad", write pseudo code ~~the~~ to read the addresses: 57, 98, A3, 2, b, 5F.

bank 0 0101 0111 1001 1000 1010 0011 bank 1

0010 1011 0101 1111

```
RD 57    ⇒ 5th bit in status register
select bank 1
RD 98        → 1001 1000
RD A3
~~RD~~ select D0
RD 2
RD b
RD 5F
```

the default bank address is bank 0
we can overwrite the already selected bank address
by selecting the bank we want ⇒ changing the 5th bit
~~th~~ in the status register

EEPROM   example (read EEPROM from addr (15))

1) Write the address you want to Read in EEAPR

2) Select bank 1 (b1)

b1 3) Set RD bit in EECON1 // Start reading from EEPROM to EEDATA

4) Select b∅ (bank 0)

5) to read the copied word, you find it in EEDATA

RD bit set by SW (software), cleared by HW (Hardware)

To write value to the EEPROM at address 15
1) Write the value (30) in EEDATA
2) " " address (16) in EEADR
b1 3) Set WREN in EECON1 // writing in the EEPROM is enabled

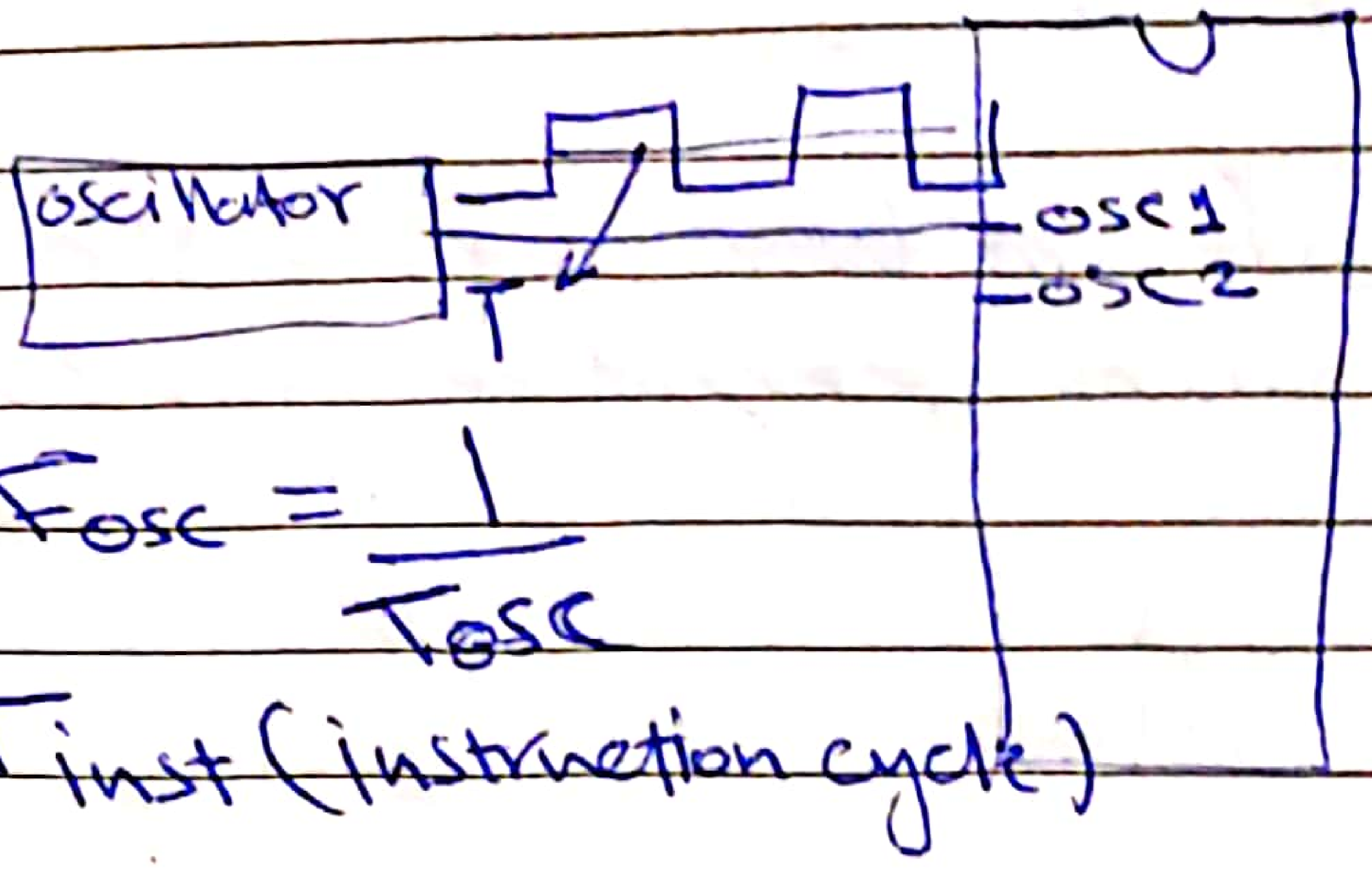
4) write 55H to EEPROM → 0101 0101 | tells the
5) " AAH " EECON2 → 1010 1010 | uC writing is intentional

6) Set WR bit ~~to~~ in EECON1 // start ~~reading~~ writing from EEAPR to EEPROM

7) when writing is over, EEIF=1 in EECON1
8) If I have any error in the writing WRERR 1 in EECON1.

Clock

Delay
→ SW ⎫
  clock
→ HW ⎭



$F_{osc} = \frac{1}{T_{osc}}$

T_{inst} (instruction cycle)

→ is the time enough to fetch or execute one instr

fetch time = execute Time

$T_{inst} = X * T_{oscillator}$

in PIC : $T_{inst} = 4 * T_{oscillator}$

Ex: assume μC1 has $T_{inst} = 10\, T_{osc}$ and μC2 has T_{inst} of $20\, T_{osc}$

μC1 works on freq 20 MHz
μC2 // // // 30 MHz

μC1 ⟹ $T_{inst} = \frac{10}{20\,MHz} = 0.5\ \mu s$

μC2 ⟹ // $= \frac{20}{30\,MHz} = 0.666\ \mu s$

one instruction to be fully executed
Fetch + execute

μC1 = 0.5 + 0.5 = 1 μs

μC2 → 0.666 + 0.666 = 1.3 μs
μC2 better

for each instruction to be fully executed, needed time is $2T_{instr}$

ex. if we have a prog that is composed of instrs. what is the whole execution time for this prog, $F_{osc} = 4 MHz$

$T_{osc} = \frac{1}{4MHz} = 0.25 \mu s$

$\Rightarrow T_{inst} = 4T_{osc} = 1 \mu s$

for a prog with ~~a~~ 10 instrs $\Rightarrow 10 \times \frac{2 \mu s}{instr} = 20 \mu s$
if there is no pipelining

* With pipelining the execution time goes to half approximately

* Some instrs will cause a failure in the pipeline because the fetched instruction is not ~~that~~ the one that will be executed next. In this case, the fetched instruction will be flushed, and the correct instr will be fetched

* all instrs with pipeline need 1 T_{inst} except the instr that causes the failure, takes $2T_{inst}$

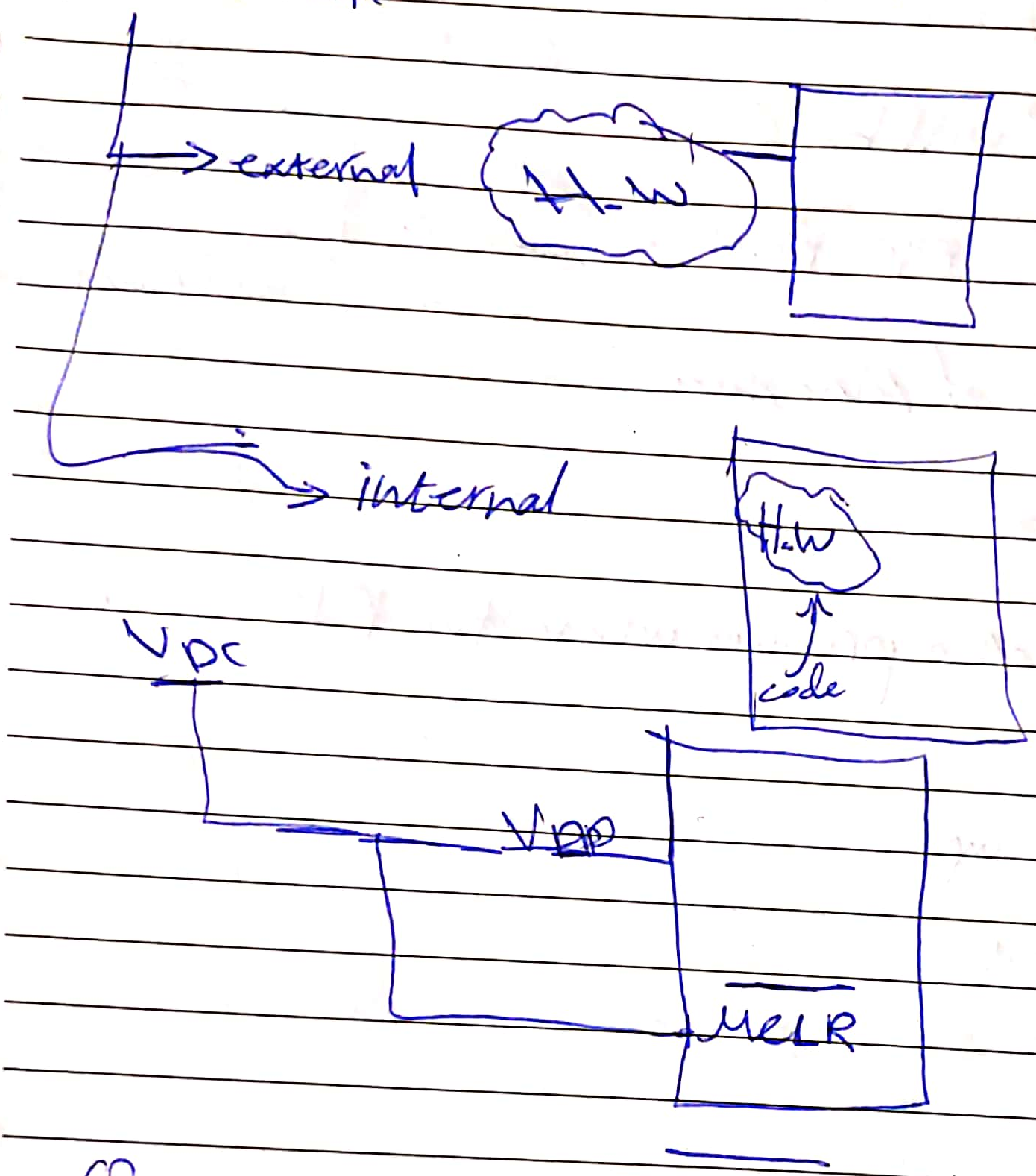
$F_{osc} = 20 MHz$

$T_{osc} = \frac{1}{20m} = 50 ns$

$T_{inst} = 50 ns$

Power up

on Power up, we need to keep the PIC in reset mode for a while



On power up ⇒ Reset Pins will stay low untill the capacitor is charged, which needs time relative ~~constant~~ $\tau = RC$

* Rs is ~~a~~ used to protect the PIC from high voltage

* free wheeling diode is used to make the reset process faster (Capacitor will discharge through it)

* when does the reset happen
S=1 & R=0

if S=1 ⇒ R will be 0

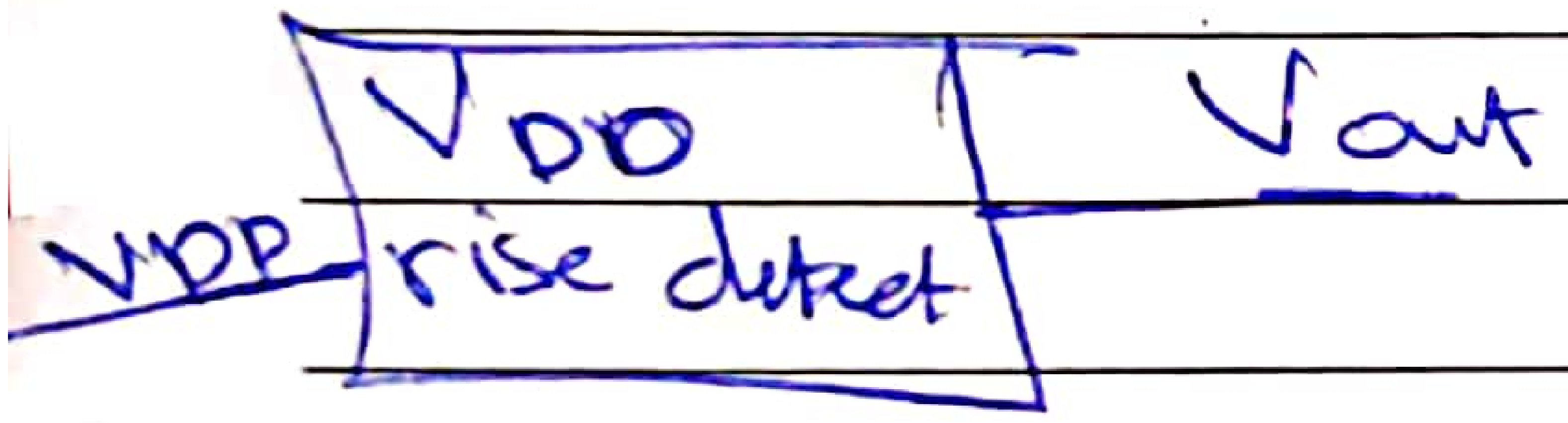
	S	R	Q	$\overline{Q}$
	0	0	Q	$\overline{Q}$
	0	1	0	1
PIC Reset ←	1	0	1	0
	1	1	undefined	

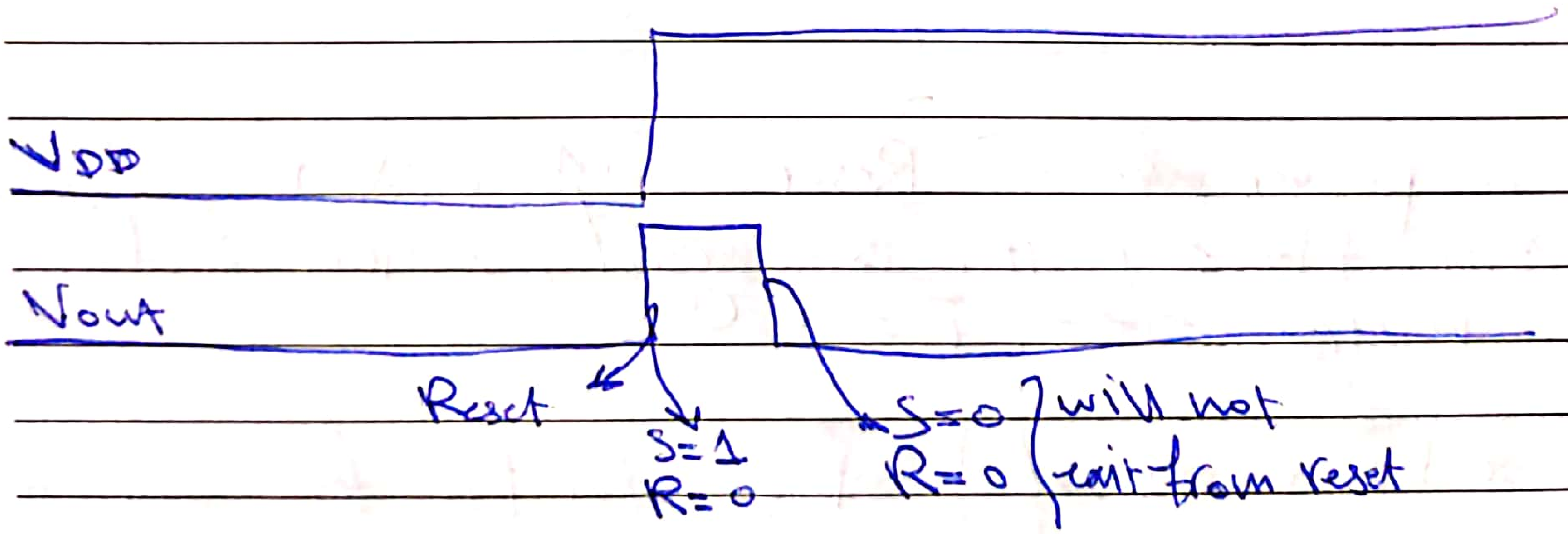
S=1 in any of these cases

1) $\overline{MCLR}=0$

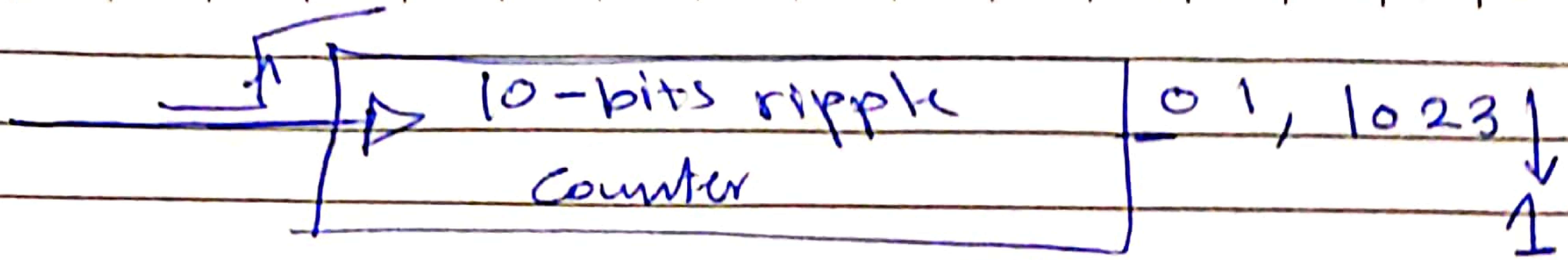
2) $\overline{WDT}$ detects a problem while the PIC is not in sleep mode

3) On power up





so when do we exit from reset mode
when R=1



after 1024 cycles of the driving clock, output = 1 / 72 ms

If Enable PWRT is 1 we have to wait 1024 cycles

If enable PWRT is enabled → ~~one~~ enable bit in the configuration word

1024 cycles of internal RC OSC + 1024 cycles of external RC OSC

if enable OSC is enabled

it is by default enabled if you choose OSC of types XT, LP

Data memory 2)
1) Vertically A(SFR B) General Registers

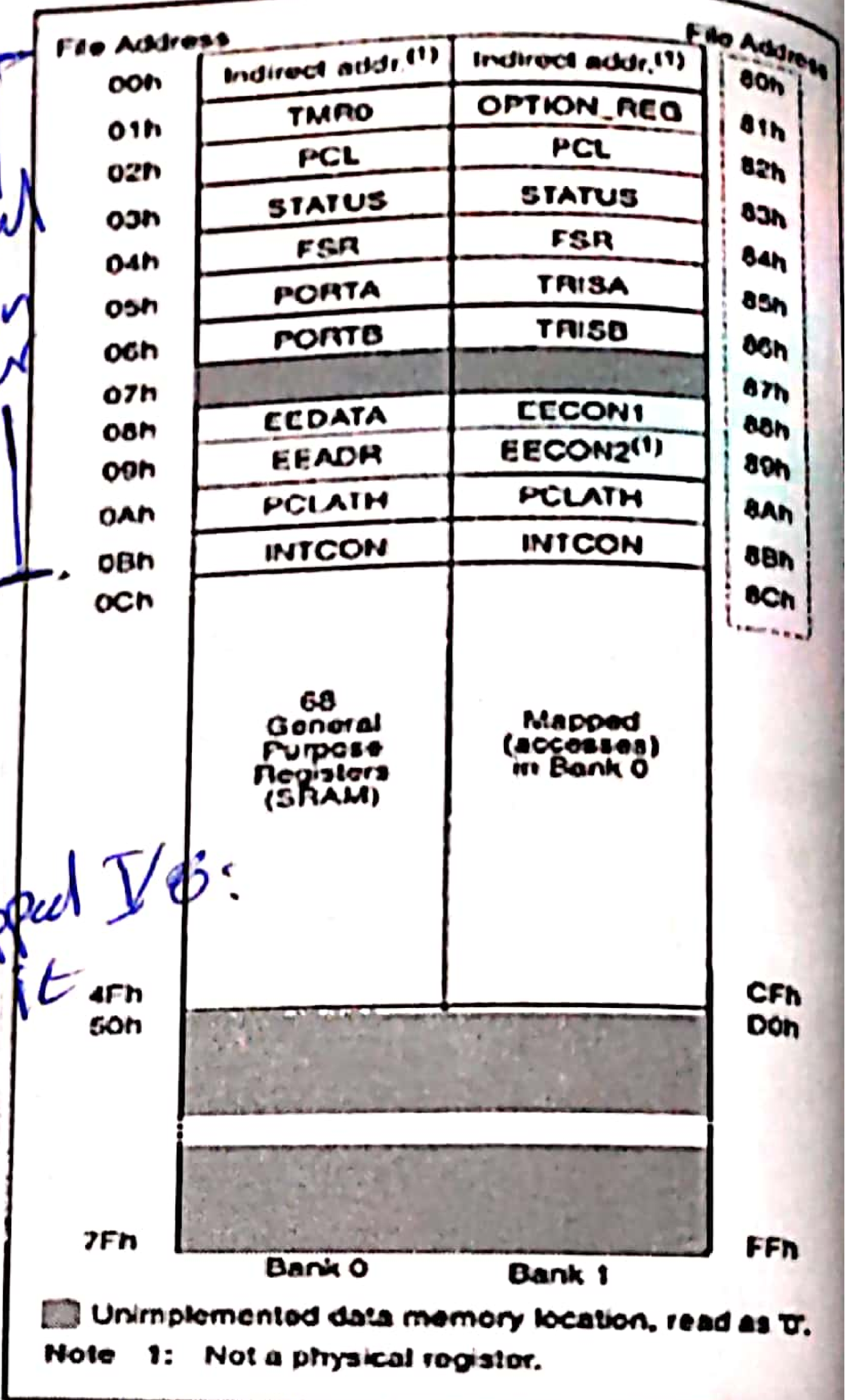
The PIC 16F84A Memory Organization

Data Memory and Special Function Registers (SFRs)

- *SRAM (volatile)*
- *Banked addressing*

special function Register

- ***Special Function Registers SFRs***
  each of these has a specific func
  - *Locations 01H-0BH in bank 0 and 81H-8BH in bank 1*
  - *Used to communicate with I/O and control the microcontroller operation*
  - *Some of them hold I/O data,* memory mapped I/O:
  → every I/O device has a memory mapped to it
- ***General Purpose Registers***
  - *Addresses 0CH – 4FH (68 Bytes)*
  - *Used for storing general data*



File Address			File Address
00h	Indirect addr.[1]	Indirect addr.[1]	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	▨	▨	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2[1]	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68 General Purpose Registers (SRAM)	Mapped (accesses) in Bank 0	8Ch
4Fh			CFh
50h	▨	▨	D0h
7Fh			FFh
	Bank 0	Bank 1	

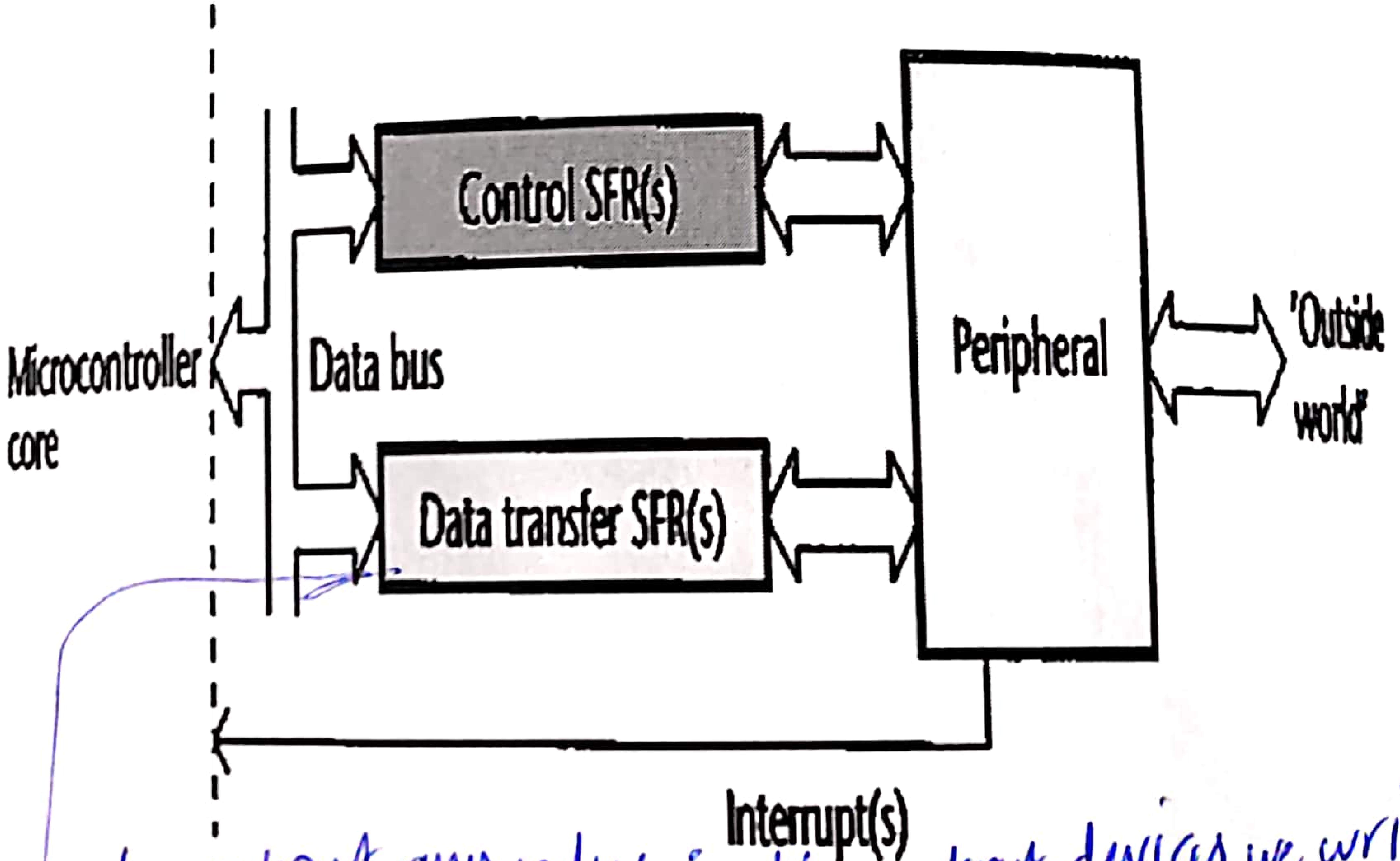
▨ Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

→ to program the I/O devices we need to know what to write in the SFRs mapped to the I/O device

The PIC 16F84A Memory Organization

- *Special Function Registers (SFRs) interacting with peripherals*



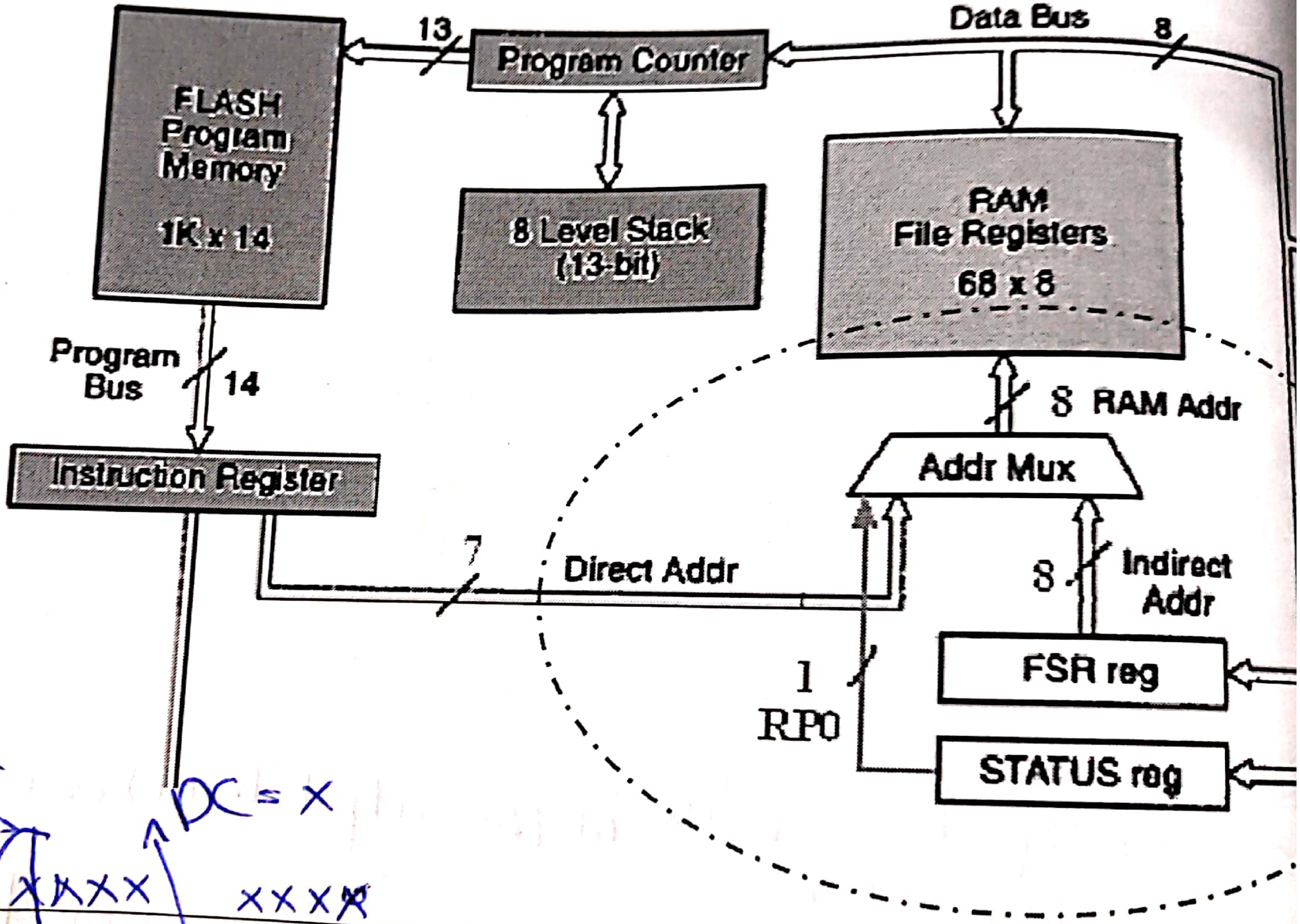
→ to output any value in the output devices we write it in the data register

→ In case of input devices, the data register holds the entered data

The PIC 16F84A Memory Organization

The PIC 16F84A Memory Organization

- ***Data Memory Addressing***



C = X

DC = X

XXXX XXXX

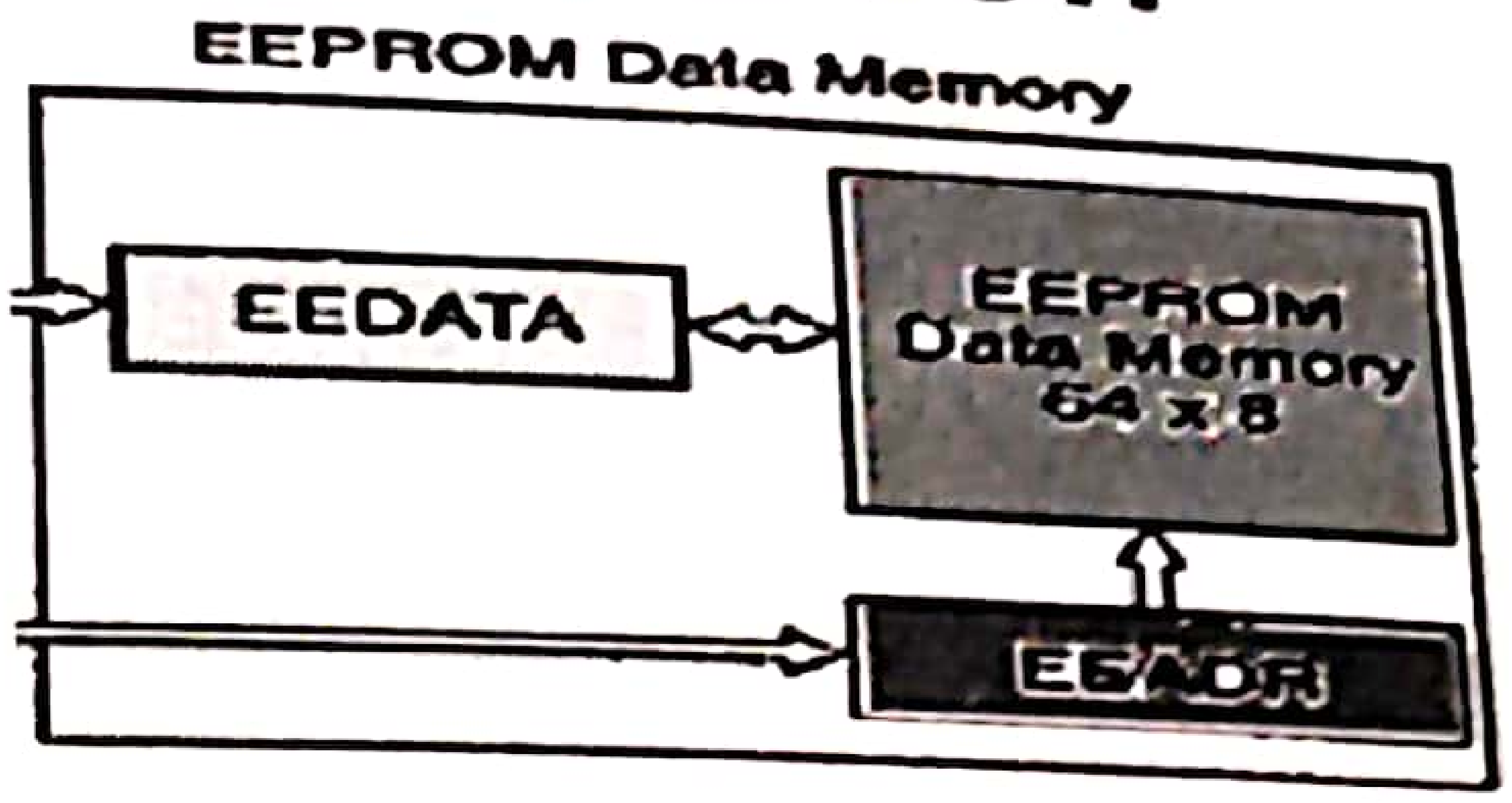
XXXX XXXX

XXXX XXXX

f = φ ⇒ Z = 1

The work with the EEPROM is done through SFRs exactly as dealing with I/O s

The PIC 16F84A Memory Organization



- *Data Related*
  - *EEPROM Data Memory*
    - *64 bytes Non-volatile*
    - *10 000 000 erase/write cycles*
    - *Used to store data that is likely to be needed for long term*
    - *Operation is controlled through EEDATA (08H), EEADR (09H), EECON1 (88H), and EECON2 (89H) SFRs*

code | 08 09 EEDATA EEADR disk | EECON1 EECON2 88 89 control

EEPROM

    - ***To read a location***
      - *store the address in EEADR and set the RD bit in EECON1*
      - *data is copied to EEDATA register*
    - ***To write to a location***
      - *data and address are placed in EEDATA and EEADR, respectively*
      - *enable writing by setting the WREN bit in EECON1 SFR*
      - *store 55H then AAH in EECON2*
      - *commit writing by enabling the WR bit*
      - *Once the write is done, the EEIF flag is set in EECON1.*

The PIC 16F84A Memory Organization

- *The EECON1 Register (88H)*

U-0	U-0	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
—	—	—	EEIF	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIF:** EEPROM Write Operation Interrupt Flag bit

1 = The write operation completed (must be cleared in software)

0 = The write operation is not complete or has not been started

set by hardware
cleared by software

bit 3 **WRERR:** EEPROM Error Flag bit

1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ Reset or any WDT Reset during normal operation)

0 = The write operation completed

bit 2 **WREN:** EEPROM Write Enable bit

1 = Allows write cycles

0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit

1 = Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.

0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit

1 = Initiates an EEPROM read, RD is cleared in hardware. The RD bit can only be set (not cleared) in software.

0 = Does not initiate an EEPROM read

Chapter 4

destination bit D

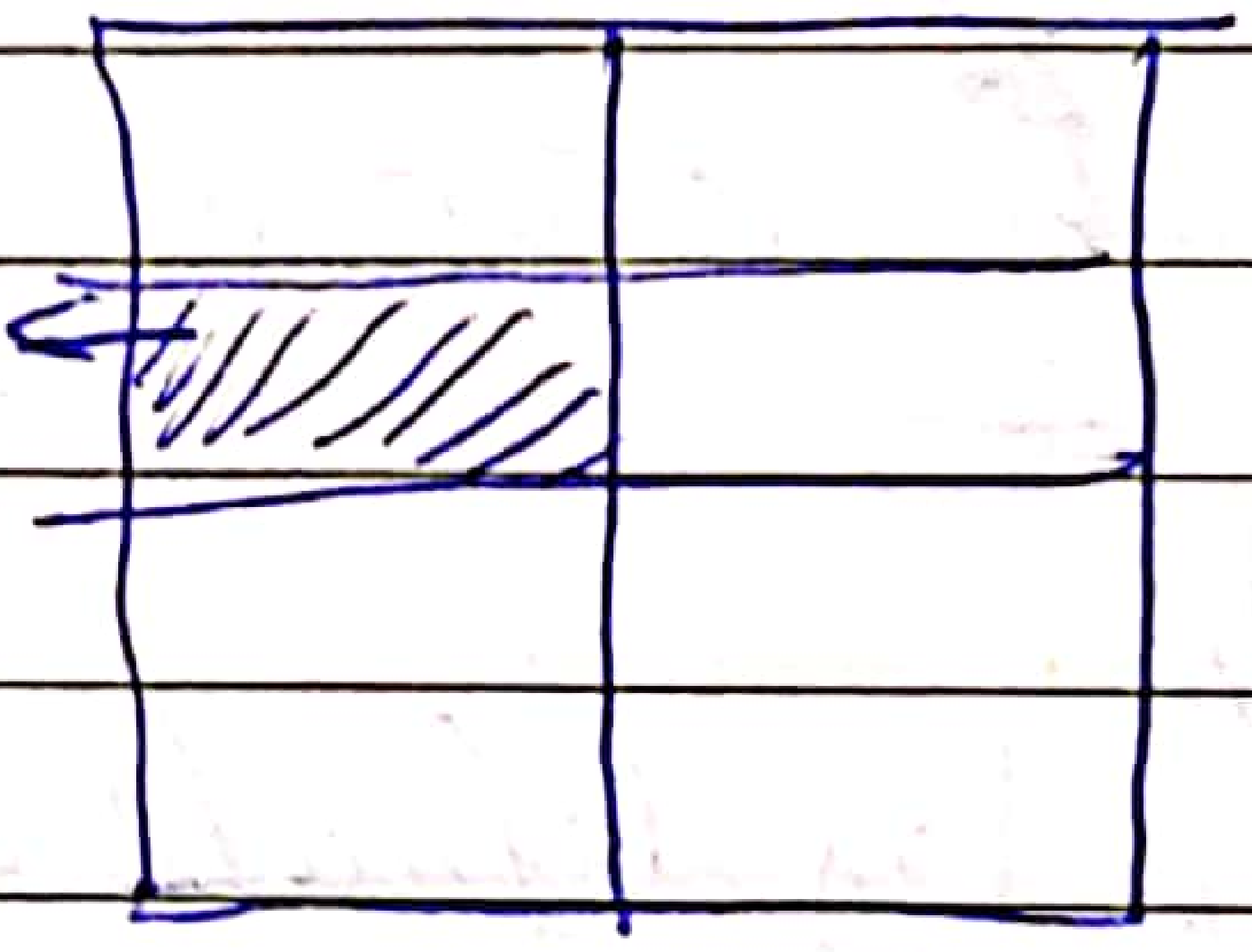
d=0 ⇒ result stored in W register
d=1 ⇒ " " " data memory

opcode	operands

Types of instructions

1) Byte oriented file register → works on data memory

works on one word/Byte
f (7-bits) address of data memory
d (1-bit)
op f,d



2) Bit oriented file register → works on data memory

op f, b

works on one bit
f (7-bits), b (3-bit) (bit location)

3. Literal ~~add~~ operations : the value is in the instruction → op K
K (8-bits) / destination is w-reg

4) Control operations : they change path of execution
call : K (an address for the instruction)
Goto (11-bits)
→ op K

Arithmetic instructions

Examples

Given that the data memory and the w-Reg have the following initial state, what are the final states after executing the following code

15	1
16	2 1
17	3

W = 4

~~Address~~

AddWF 15, ~~1~~ 0 , 4+1 ⇒ W=5
ADDWf 16, 0 , 5+2 ⇒ W=7
ADDLW 3 → 3+7 = A ⇒ W=A
ADDWF 17 , 1 → A+3 ⇒ data(17) = D

IncF 15 , 0 → W = 2
DECF 16 , 1
COMF 17 , 1 , address 17
↓ invert address 17 = F2

D

0000 1101

1111 0010

F 2

A - B = A + 2's complement of B

A = 5 , B = 7

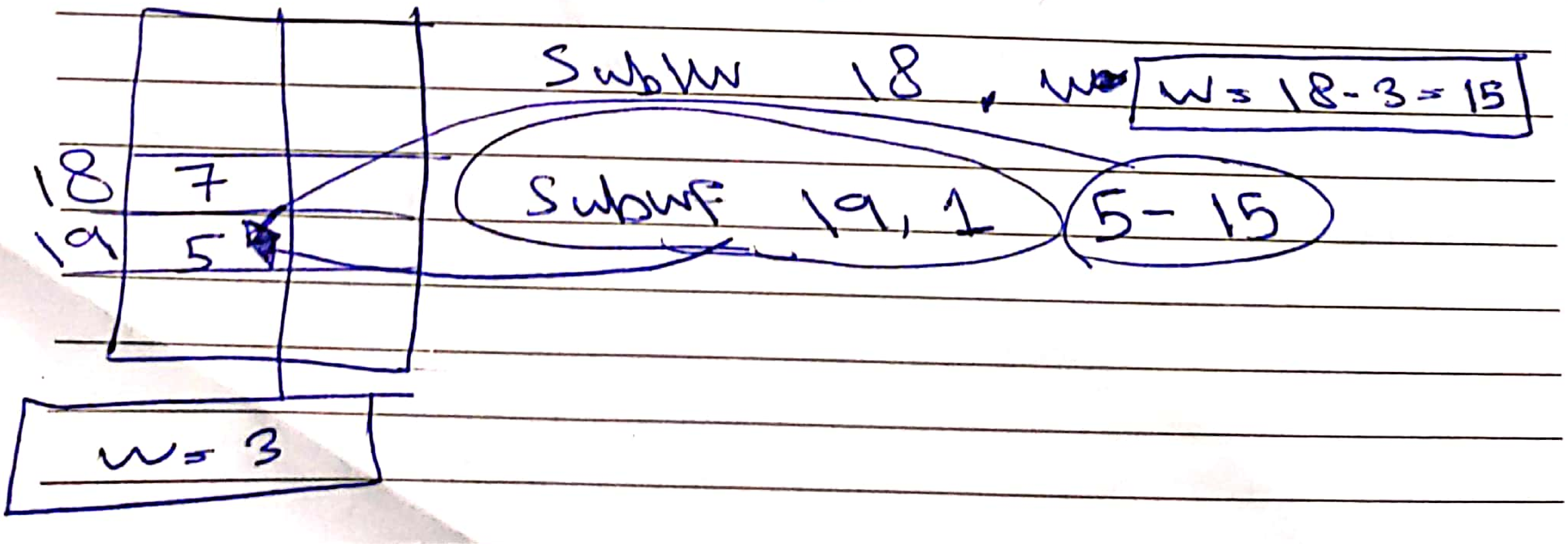
```
A - B      C=1
           ↖   0000   0101
            \  1111   1001
             ‾‾‾‾‾‾‾‾‾‾‾‾‾‾
               1111   1110

B - A      C=1
           ↖   0000   0111
            \  1111   1011   +
             ‾‾‾‾‾‾‾‾‾‾‾‾‾‾
               0000   0010
```

C = 0 result is negative
C = 1 // // positive

we always subtract the working register

SUBLW ~~K~~ [W] = K - [W]
SUBWF [F] - [W]

Sub exercise



18	7
19	5

W = 3

SUBLW 18 , W   W = 18 - 3 = 15

SUBWF 19, 1   5 - 15

logical instructions : we use them to do bits masking

if you want to set, clear, or complement a part of the word

$X.\emptyset = 0$ Clear $\quad X \oplus \emptyset = X$

$X.1 = X \quad X \oplus 1 = \bar{X}$ Complement

$X + \emptyset = X$

$X + 1 = 1$ Set

```
ADDWF  f,d
ADDLW   K

IORWF  F,d
IORLW   K

XORWF  F,d
XORLW   K
```

Examples.

run 1 instr to clear the least signif. 4-bits of the W-reg and keep the others

```
ANDLW  FØ
```

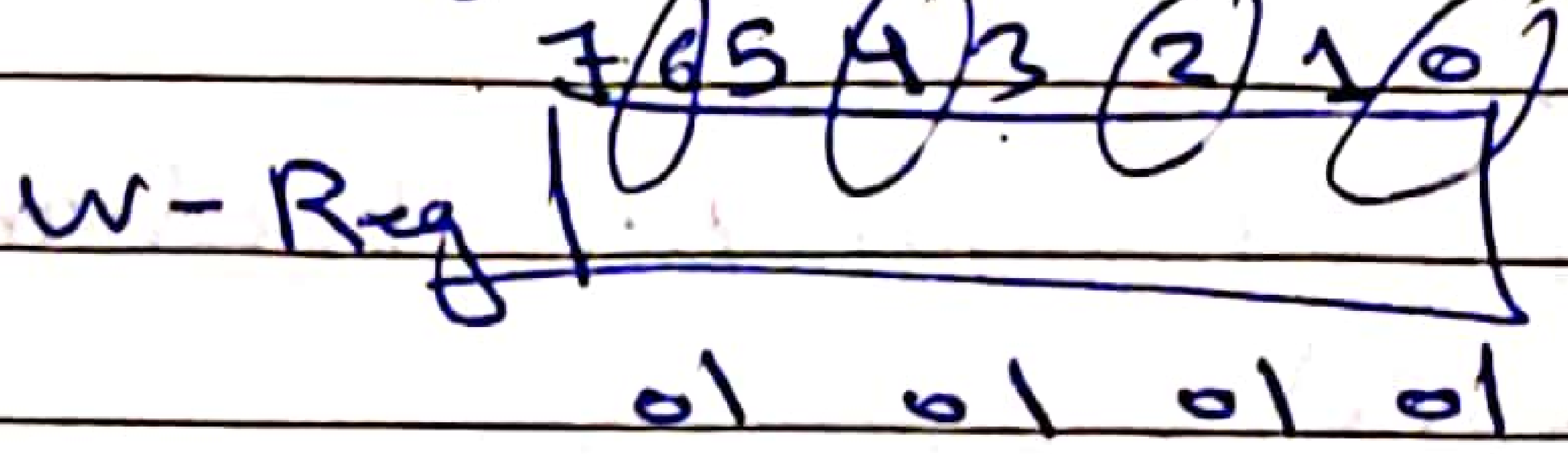
2) Set the most signif. 4-bits in the W-Reg

```
IORLW  F0
```

Scanned with CamScanner

3) Complement the bits in W-Reg with even positions

XORLW 55



4) Set the most signif. 4-bits in the data memory word with address 20

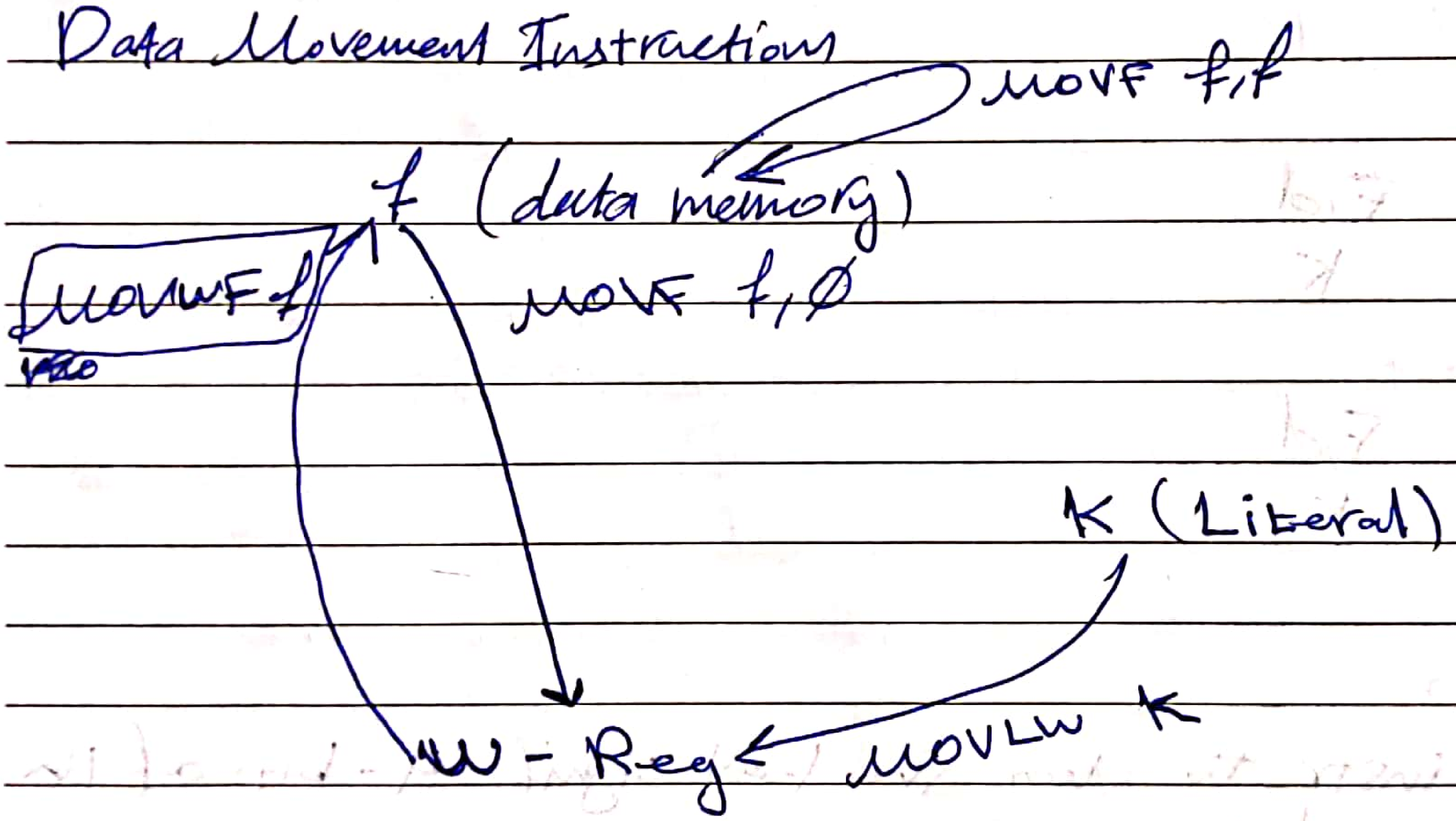
~~WRITE~~

~~IORWF 20,~~ IORLW F0

Write F0 in W-Reg OR ANDLW F0

IORWF 20,1 IORWF 20,1

Data Movement Instructions



Examples:

Write code to Initialize the file register with address 21 to the value 15

1) Write in W-Reg

~~2) Copy~~

MOVLW 15

2) Copy W-Reg to DM

MOV WF 21

Ex: copy value in addr 22 to address 23

1) MOVF 22,∅

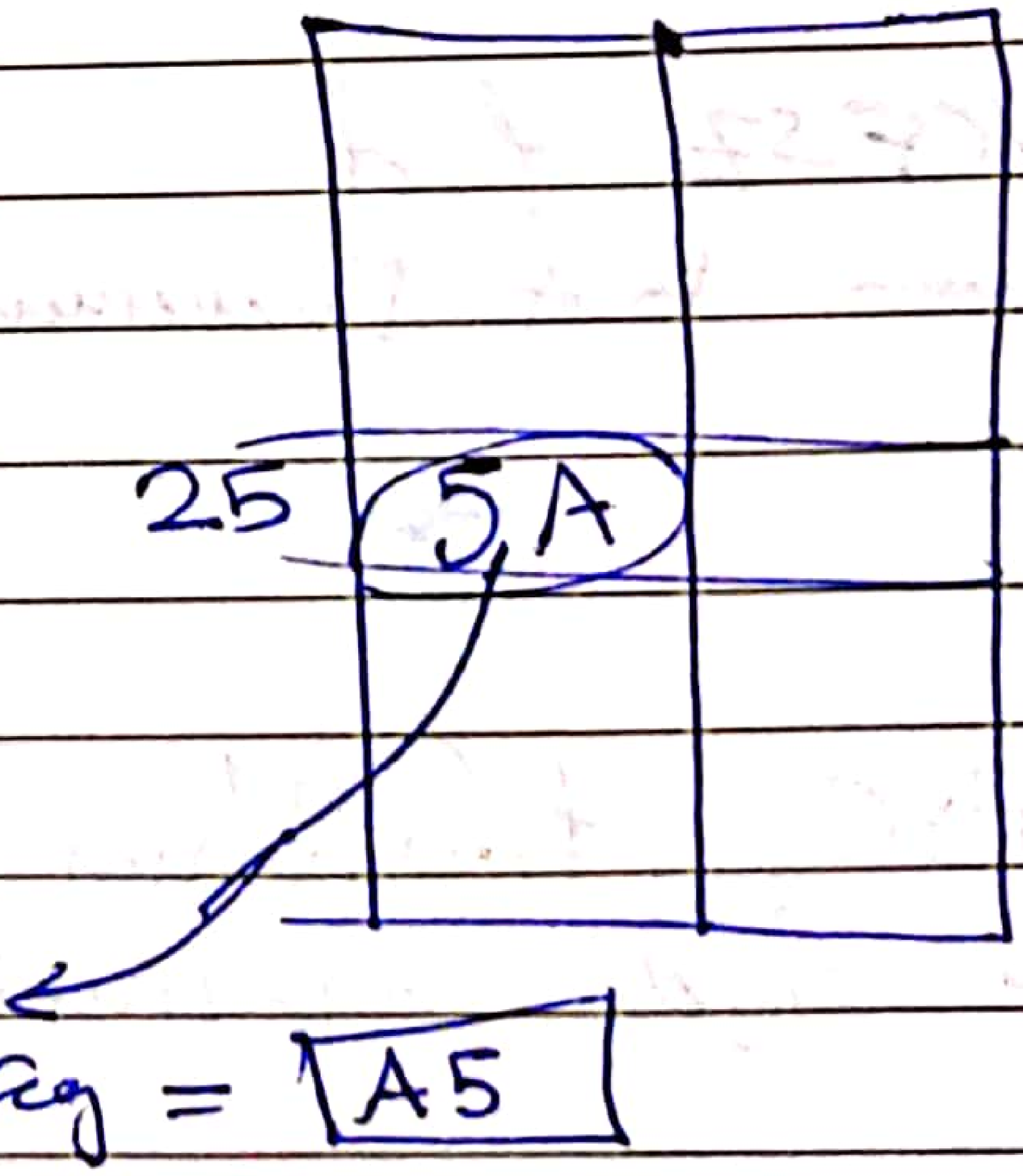
2) MOVWF 23

MOVF is used to check if the value in the bit is 0 or Not

if ([25]==0)
MOVF 25,1
code to check Z-flag

Caution!

SWAPF 25, ∅



Control instrs

Call K   Stack ← PC   you can
          PC ← K   return

PC ← K   you can't return

GOTO K

→ address of instruction (11-bits)

Return $PC \xleftarrow{POP}$ stack

retlw K } → subroutine
retfie → interrupt

Conditional Branch: GOTO if condition true / skip if condition is true → one instr

if
if else
loops

DECFSZ f,d

the condition is true if after ~~Increment~~ decrement
the value in address f is zero

INCFSZ f,d
→ same but Increment

BTFSC f,b bit position

the condition is true if bit # b in address f is Ø

BTFSS f,b
→ same but ≠ Ø

On skip these instrs they take 2 T instrs, Otherwise
1 Tinst

e.g

```
MOVF    23, 1            = if ([23,] = ∅)
BTFSS   STATUS, Z            [23] ++
INCF    23, 1
```

[23] = 0 ⇒ Z = 1

```
MOVLW   5
SUBWF   24, ∅
BTFSC   STATUS, C
INCF    24, 1
INCF    25, 1
```

[24] - 5
if [24] > 5 ⇒ C = 1 ⇒ ~~sk~~ skp
if [

In C if [24] > 5
[24]++
[25]++

Ex: For (int i=15; i>0, i--)

```
{
    block of code
```

```
MOVLW   15        i = 15
MOVWF   30        [30]
loop: block of code
      DECFSZ  30, 1
      GOTO    loop
```

if = 0 ⇒ infinite loop

the skip is not dependent on d-bit

Home exercise

```
For (int i=0 , i < 50 , i++)
        block of code
```

do it yourself

Misc instr.

BCF f, b clears bit number b in address f

bcf 03, 5

Status register → selects bank 0

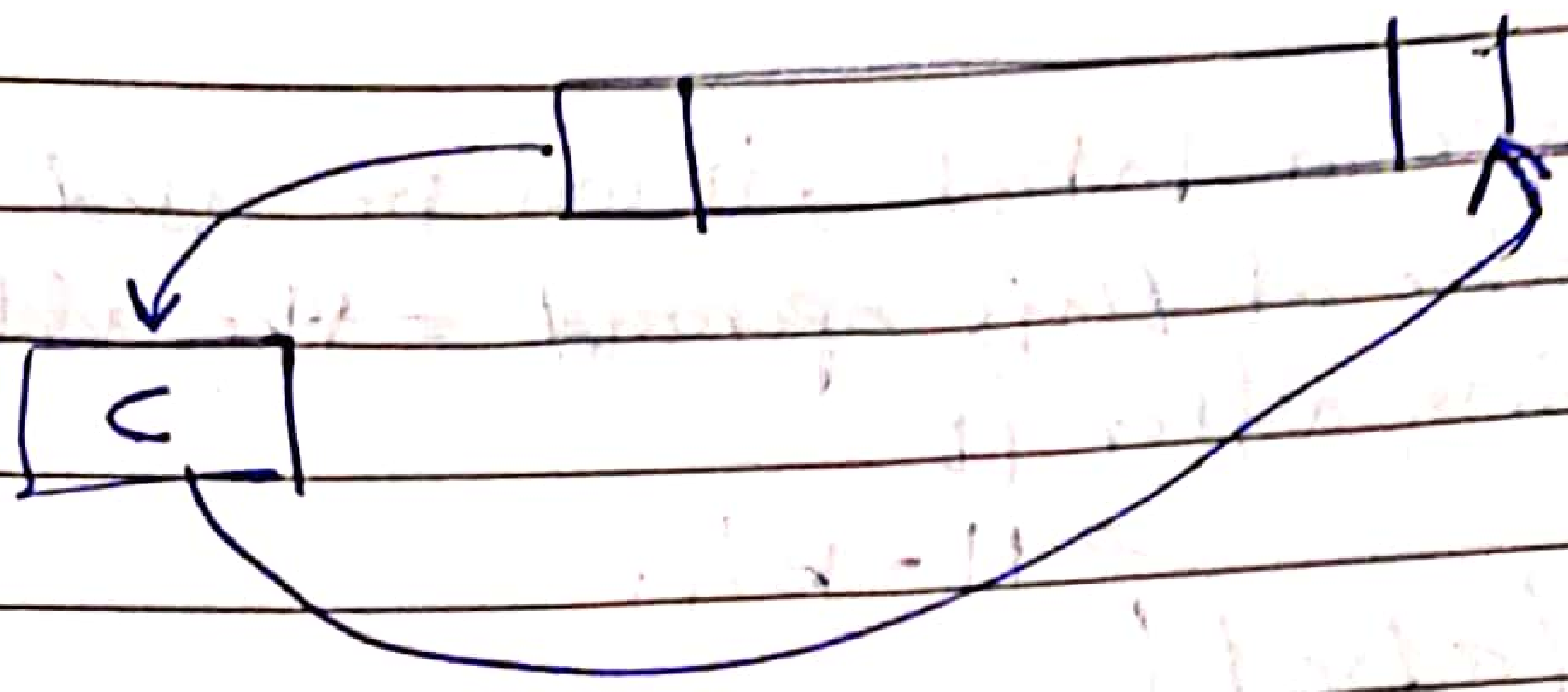
BSF f, b sets — — — —

BSF STATUS, 5 → selects b1

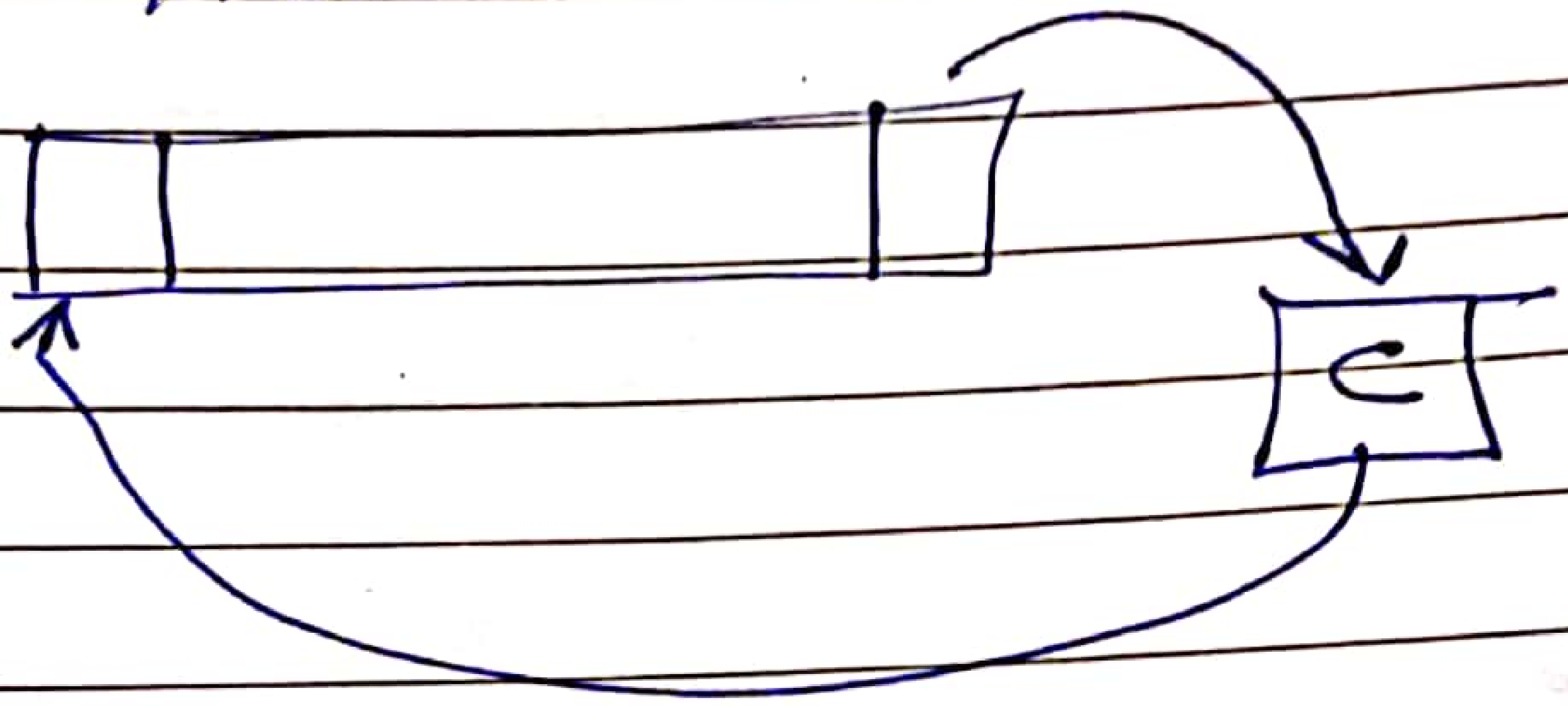
CLRWDT → WDT

Sleep → to enter sleep mode

RLF f,d



RRF f,d



6

0 0 00 0110

12 0 0 00 1100 ←

13 0 0 00 1101.

2n+1

code for multipkying value in address 25 by 4

```
BCF  STATUS, C
RLF  25, 1
BCF  STATUS, C
RLF  25, 1
```

Once you write a label, it can be used as an operand, the value of this operand = the address of the first inst coming after it.

GOTO label — 11-bits

Label
↓
zero-bits محجوزة
memory بال

Assembler details

Assembler directives : it gives some information to the assembler at compilation time, then they're discarded

```
#Include < ioStream.h>
void main() {
cout << "Hello";
```

#Include PIC 16F84LATNS

⇒ You can address registers and bits using their name

ORG 05 it tells the assembler that the next instr should be stored in 05

Instr1
instr 2
instr 3

start of code → ORG 0000
GOTO START

start of interrupt ← ORG 0004
GOTO ISR

equ: it defines a constant

var1 equ 4

MOVLW var1
MOVF var1, 0
address

BCF var1, var1
address bit number

cblock 20
const1
Const 2
Const 3

const1 equ 20
Const2 // 21
Const3 // 22

Sample program 1

```
ORG     0000
GOTO    START
ORG     0004
ISR GOTO ISR

START BCF   STATUS , RPØ
MOVF     31,Ø
ADDWF    45,Ø
ADDWF    47,Ø
MOVWF    22
GOTO     DONE
end
```

CA 5

```
if ([23] < ~~2~~7)
    [23] = [23]x4
else
    [23] = [24]-8;
```

[23]-7

[23]>7 ⇒ C=1

```
        MOVLW     7
        SUBWF     23, ∅
        BTFSS     STATUS, C
        GOTO      mul
        GOTO      sub

mul:    BCF       STATUS, C
        RLF       23, 1
        BCF       STATUS, C
        RLF       23, 1
        GOTO      next
sub:    MOVLW     8
        SUBWF     24, ∅
        MOVWF     23


next:
     MOVLW   7
     SUBWF   23,∅
     BTFSS   STATUS, C
     GOTO    MUL

Sub  MOVLW   8
     SUBWF   24,∅
     MOVWF   23
     GOTO    Next
MUL  BCF     STATUS, C
     RLF     23,1
     BCF     STATUS, C
     RLF     23,1

for (i= 20 ; i>0 ; i--)
        [22]++

Counter        EQU      23
               MOVLW    D'20'
               MOVWF    Counter

~~start~~ loop      INCF  22,1

               DECFSZ   Counter,1
               GOTO     loop
```

for (int i = 5 ; i < 30 ; i++)
    add 3 to the address 16

Counter   EQU   24
           MOVLW   5
           MOVWF   Counter

label
           MOVLW 3
           ADDWF 16,1

           INCF   Counter, 1   i ≤ 31   إذا بدي اياه
           MOVLW   D'30' ⇒   خليها هاي.
           SWBWF   Counter, 0   D'31'
           BTFS$\boxed{S}$ STATUS, Z
           GOTO   label

Tarlat

Home exercise

for (int x = 3 ; x ≤ 50 ; x+=2
    if ([23] == 7)
    {
        if ([24] is even)
            multiply [24] by 2
        else
            copy the value in address 15 to 16
    }
    else
    {
        multiply the value in address 18 by 23
    }

Conditional branching Example 1 in slides

```
[11] + [22]
 if (c==0)
  → [33]
else
  → [44]
```

~~GOTO Store 33~~
~~GOTO Store 44~~

~~Store33 movwf ~~MOVF~~ 33~~
~~GOTO ~~NEXT~~~~

MOVF 11,0
~~[illegible]~~ADDWF 22,0

BTFS$\boxed{S}$ STATUS ,C

GOTO Store33
GOTO Store44

Store33 MOVWF 33
GOTO NEXT

STORE44 MOVWF 44
GOTO NEXT

NEXT

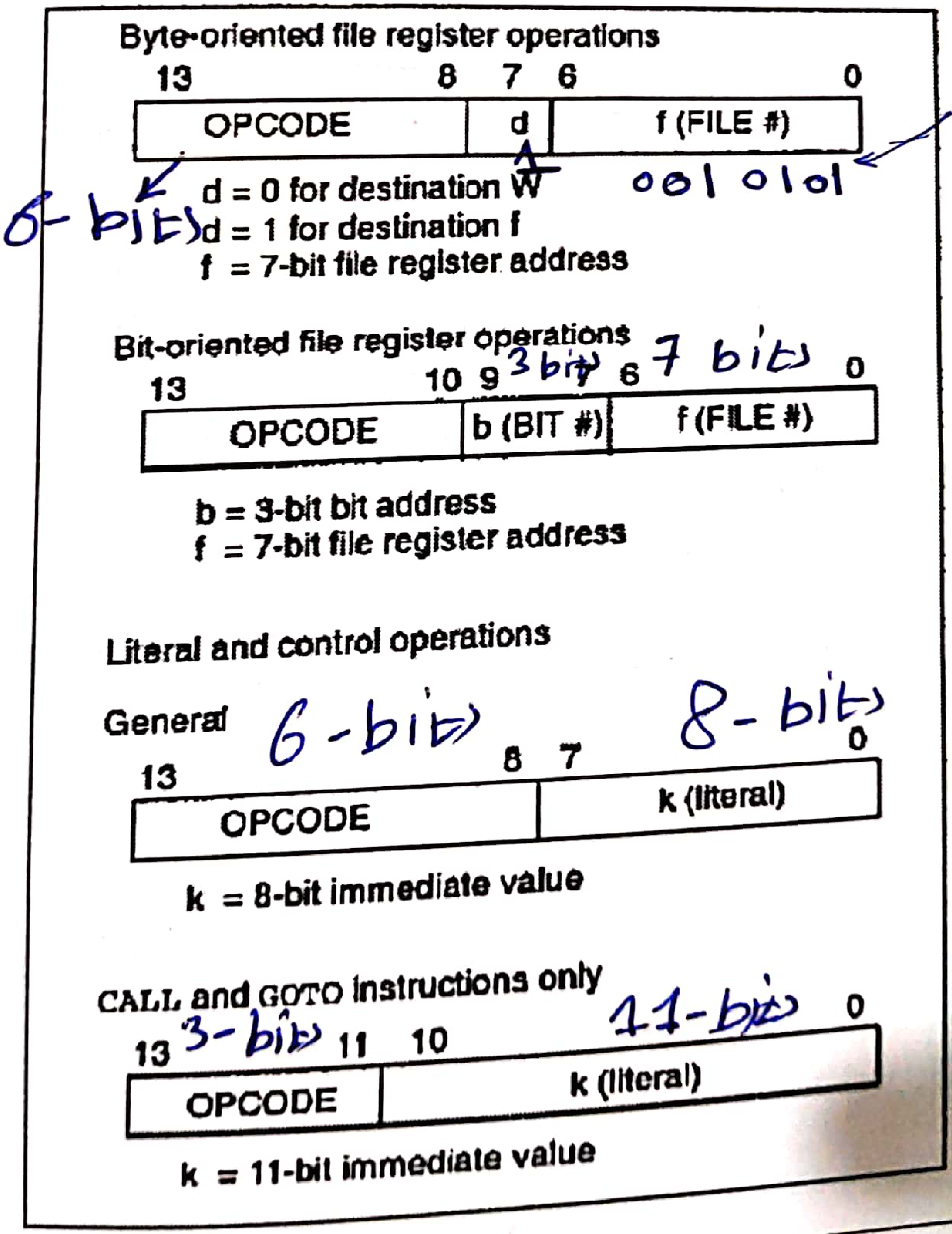
The PIC 16 Series Instruction Set Encoding

SUBWF 15, 1

BCF 03, 5

APPLW 7

GOTO 15



Byte-oriented file register operations

d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

Bit-oriented file register operations

b = 3-bit bit address
f = 7-bit file register address

Literal and control operations

General

k = 8-bit immediate value

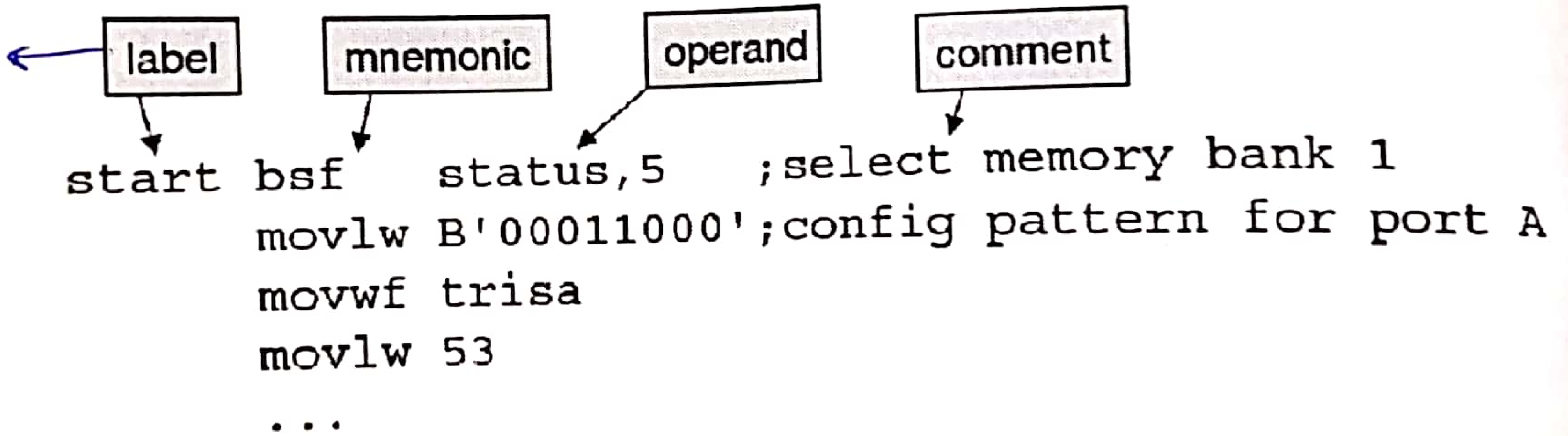
CALL and GOTO instructions only

k = 11-bit immediate value

Check Ap
for opeco
co

Assembler Details

- Any assembler line may have up to four different elemen



case sensitive

```
start bsf   status,5   ;select memory bank 1
      movlw B'00011000';config pattern for port A
      movwf trisa
      movlw 53
      ...
```

- We can specify values in different bases in assem programs

Radix	Example
Decimal	D’255’
Hexadecimal	H’8d’ or 0x8d
Octal	O’574’
Binary	B’01011100’
ASCII	‘G’ or A’G’

if the Hex value starts with a letter it should be preceded by 0x or H

default

Assembler Details

ORG 05
instr 1 05
instr 2 06
instr 3 07

~~ORG~~

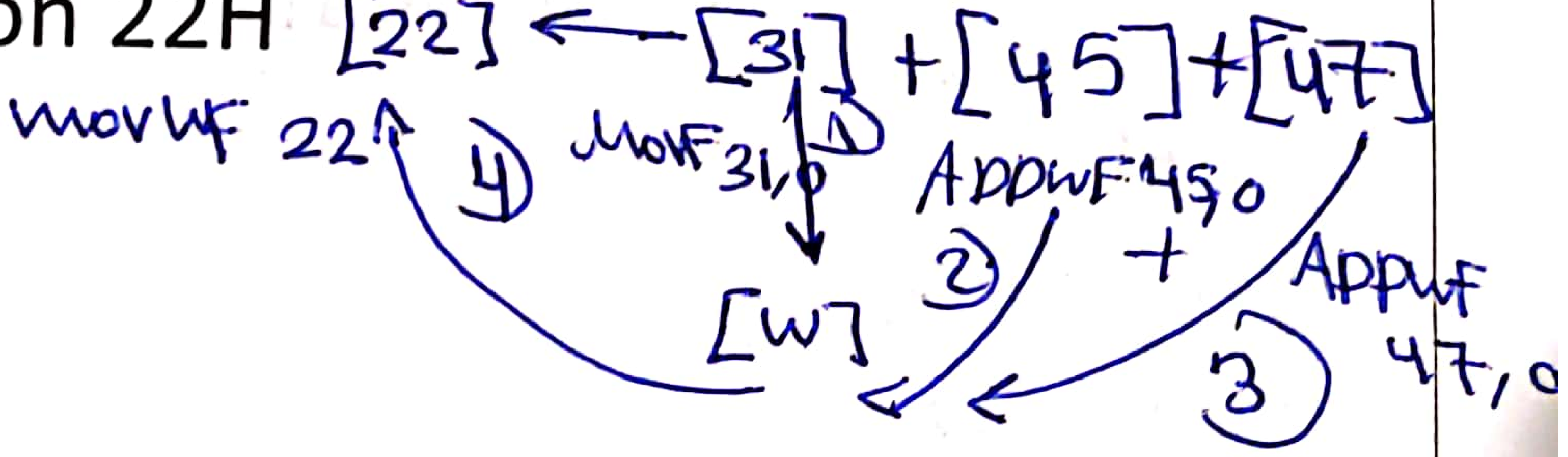
it tells the assembler that the next instr should be stored in address 05

- **Assembler directives**
  - These are assembler-specific commands to aid the proce
    of assembly programs

Assembler directive	Summary of action
org	Set program origin
equ	Define an assembly constant; this allows us to assign a value to a labe
cblock and endc	Define a block of variables
end	End program block
#include	Include additional source file

Sample Program 1

- Write a program to add the numbers stored in locations 31H, 45H, and 47H and store the result in location 22H



```
ORG 0000
GOTO START
ORG 0004
GOTO ISR
ISR
MOVF    31, Ø
~~ADDWF~~
ADDWF   45, Ø
ADDWF   47, Ø
Movwf   22
```

Sample Program 2

- Write a program to swap the contents of locati
  0x33 with location 0x11

[11]
[22]
[33]
1
2
2
3

```
MorF 11,Ø
MOVWF   22

MOVF 33,Ø
MOVWF 11

MOVF 22,0
MovWF 33
```

Sample Program 2

```
;************************* EQUATES ********************************
STATUS          equu    0x03
RP0             equ     5                       ; define SFRs
;************************* VECTORS ********************************
                org     0x0000                  ; reset vector
                goto    START
                org     0x0004
INVEC           goto    INVEC                   ; interrupt vector
;*********************** MAIN PROGRAM ***********************
START           bcf     STATUS , RP0            ; select bank 0
                movf    0x33 , 0                ; put first number in W
                movwf   0x22                    ; store the 1st number temporarily
                movf    0x11 ,  0               ; get 2nd number
                movwf   0x33                    ; store 2nd in place of 1st
                movf    0x22 , 0                ; get 1st number from 0x22
                movwf   0x11                    ; store 1st  in place of 2nd
DONE            goto    DONE                    ; endless loop
                end
```

size = 10 x 14 bits

if $F_{osc} = 4$ MHz

$\Rightarrow T_{osc} = 0.25\ \mu s$

$T_{inst} = 1\ \mu s$

Execution time of this program is 9 μs

Summary

- The PIC 16F84A has 35 instructions to perform different computational and control operations
- Programs can be written using different levels of abstraction
- Using assemblers simplifies the program development process
- There exist many IDE to aid writing programs and simulate their behavior before putting them into hardware

2 T_{instr}

Subroutine : starts with a label
ends with return ? Stack $\xrightarrow{Pop}$ PC
or return k
2 T_{instr}

```
movlw   k
Return
```

and it is executed when ~~calling~~ called by instr Call

Call. $\boxed{K}$ ← label

compiler calculates value of K at Compilation time.

Delay : 1) HW using timers & interrupt
2) SW using NOP

loop code that turns on a LED
250~~000~~ NOP instr (250 ms delay)

$F_{osc} = 4$ KHZ
$T_{osc} = \frac{1}{4\times10^{3}} = 250 \mu s$
$T_{instr} = 1$ ms

code that turns off LED
250 NOP
GOTO loop (بطفي و بيضوي)
250 ms delays

I can only store 1024 instructions in the program memory

```
movlw   X → 0 results in the maximum # of iterations
movwf   Counter

loop    NOP
        NOP
        NOP
        decfsz  Counter, 1
```

maximum 256 iterations if X = 0

```
MOVLW     X
MOVWF     CountL
MOVLW     Y                      ⇒ nested loop,
MOVWF     CountH                   256 x 256 iterations

loop      NOP
          NOP
          DECFSZ   CountL, 1
          GOTO     loop
          DECFSZ   CountH, 1
          GOTO     loop
```

```
MOV 2W        X
MOVWF         CountH
Call          delaySub
DeCFSZ        CountH, 1
GOTO          loop

delaySub      X
movwf         CountL

loop2 :       NOP
              NOP
              dECFSZ   CountL, 1
              GOTO     loop

              return
```

Delay calculation :

Single loop

1) Initialization
2) all instructions except the last
3) last iteration

Based on slides example

if ~~this~~ the code is a subroutine $5\,ms + 2\times 5\,\mu s + 2\times 5\,\mu s$

$\Downarrow$ $\boxed{Call}$ $\Downarrow$ $\boxed{return}$

```
        . Call  label
label   movlw   D'200'
        movwf   Counter

~~Loop~~ :  ~~[illegible]~~
loop    nop
        nop
        decfsz  Counter, 1
        goto    loop
```

modify this subroutine code to give exactly 4 ms delay

$4\times10^{-3} = 5\times10^{-6} * \# \text{ of instr}$

$\hookrightarrow = 800$

$800 = \overset{Call}{2} + (\overset{movlw}{1} + \overset{movwf}{1}) + (X-1) * (\overset{nop}{1} + \overset{nop}{1} + \overset{decfsz}{1} + \overset{goto}{2}) + (\overset{nop}{1} + \overset{nop}{1} + \overset{decfsz}{2} + \overset{return}{2})$

$800 = 2 + 2 + (X-1) * 5 + 6$

$800 = 10 + 5X - 5$

$X = \frac{795}{5} = \underline{\underline{159}}$ iterations

Example : write subroutine to give exactly 7ms delay including the call instruction assuming $\boxed{F_{osc} = 400\text{ kHz}}$

$$T_{inst} = \frac{4}{400\text{kHz}} = 10\ \mu s$$

$$7\ ms = 10\ \mu s * \#\text{ of } T_{inst}$$

$$\Rightarrow\ = 700\ T_{inst}$$

```
delay7ms    movlw   'x'
            movwF   Counter
            nop
            nop
            nop
loop        nop



            DECFSZ  Counter, 1
            GoTo    loop

            return
```

$$700 = \overset{\text{Call}}{2} * 2 + (X-1) * \overset{\text{last iteration}}{4 + 5}$$

~~X~~

$$X = 173.75$$

if we put movlw D'173'

$\#$of instr = $4x + 5 + 4 = 697$, we need 700 to get 7ms

So we put 3 more nop before the loop

Nested loop : Initialization + first external iteration + last external iteration + all internal except last + last internal

all external iterations except first and last

example at slide 23

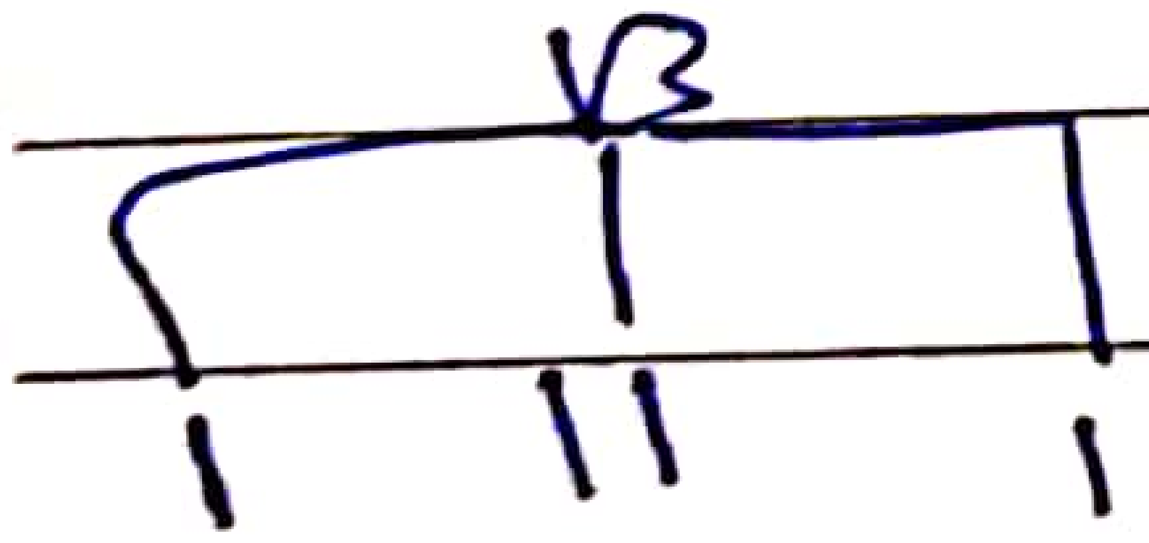
$\overline{T}_{inst} = \frac{4}{F_{osc}} = 1 \mu s$

$\Rightarrow$ delay $= T_{inst} * \# \text{ of } T_{inst}$

$\# \text{ of } T_{inst} = \frac{10 * 10^{-3}}{1 \times 10^{-6}} = 10{,}000$

$10{,}000 = 2^{Call}$

Count H



Ex: write code to clear data memory from address 0X10 to 0X4F (64) locations

```
CLRF f

CLRF   0X10
CLRF   0X11
  ⋮

      MOVLW    0X10
      MOVWF    Counter

loop          ~~CLRF Counter~~     we need Indirect Addr

              INCF Counter, 1
              MOVLW    0X50
              SUBWF    Counter, 0
              BTFSZ    STATUS, Z
              GOTO     loop
```

Indirect addressing
1) write the address in FSR
2) replace the operand f with 00 or INDF

e.g write code to Increment the value in address 14 using direct & indirect addressing modes direct

```
INCF    14,1
```

Indirect

```
MOVLW   14
MOVWF   FSR
INCF    INDF,1
```

14

Example Indirect Addre Solution

```
        MOVLW   D'64'
        MOVWF   Counter
        MOVLW   0x10
        MOVWF   FSR
```

loop~~loop~~ CLRF INDF & 00 are not a physical register

```
        INCF    FSR,1


        DECFSZ  Counter,1
        GOTO    loop
```

How to do Bank selection

direct : STATUS(5)

indirect FSR (7)

Ex: write code to read the value in address 0x95 to the w-reg using direct & Indirect

1001 0101

b1

Cautis

Direct

```
BSF     STATUS, 5   1
MovF    0x15    001 0101
```

Indirect

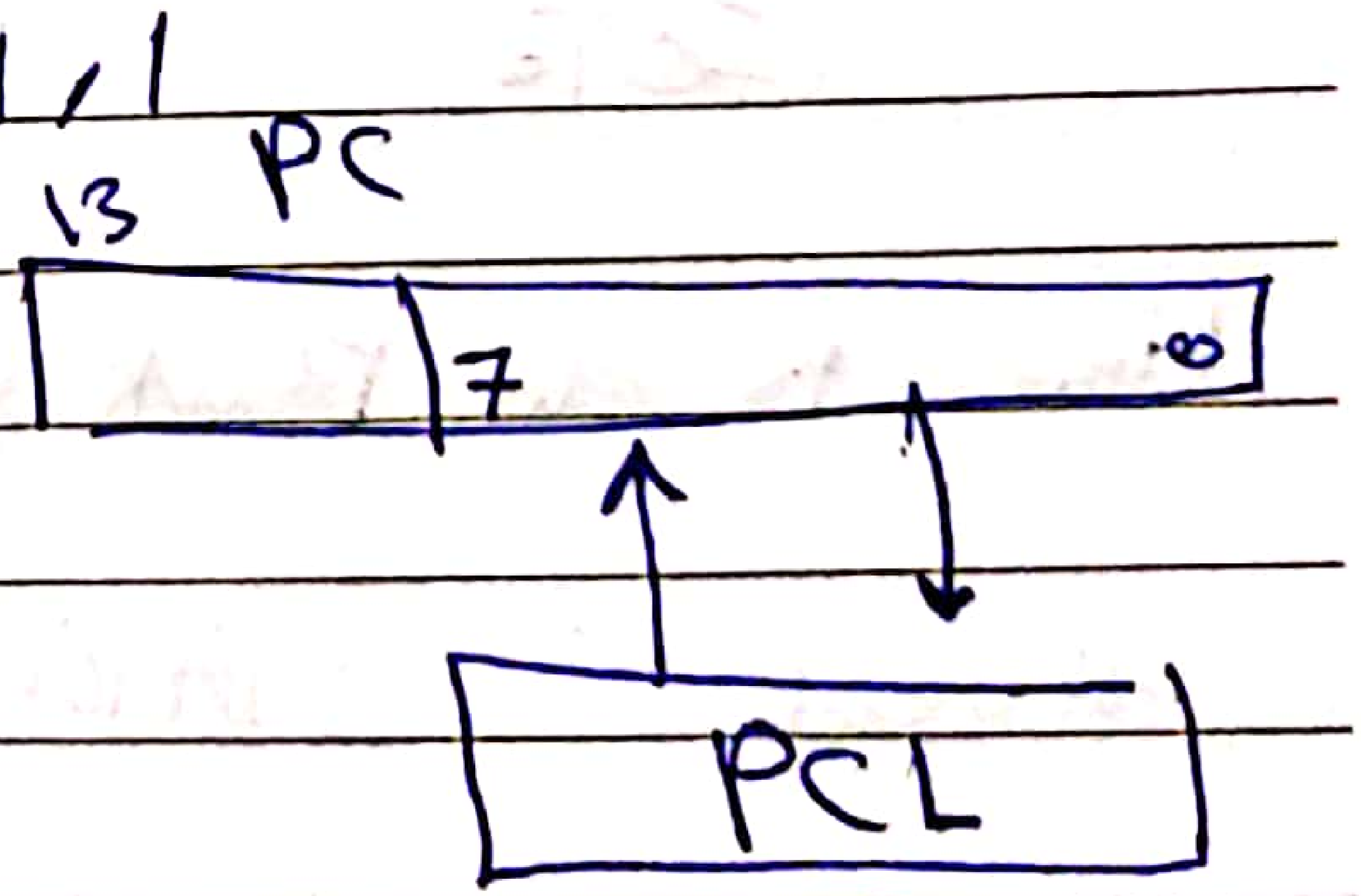
```
b1 BSF    FSR, 7
   MoVLW  0x95
   MovWF  FSR
   MovF   INDF, ∅
```

lookup table

in C

```
int square [6] = {0, 1, 4, 9, 16, 25}

    Cout << squares [3] ;
```

```
squares      ADDWF ∅ ~~Rc~~ PCl, 1

             retlw   ∅
             retlw   1
             retlw   4
             retlw   9
             retlw  D'16'
             retlw  D'25'
```



```
             MovLW   4
             Call    Squares
```

we use program memory to save lookup tables ; it's bigger, and easier for retrieval.

Example - Continued

```
                                              ClRw
                                  multiply    MovF    31, 1
                                              BTFSC STATUS, Z
multiply    clrw                              ~~return~~
Repeat      addwf  0x30, 0   ; repeated addition
            decfsz 0x31, 1   ; counter loop  : ADDWF   30, ø
            goto   repeat                      DECFSZ  31, 1
            return           أسلع انه ما تضرب   GoTo    loop
                                   ب 0          return

            end
```

```
            MovF      31, ø
multiply    MovWF     Temp              إذا بدناش
            CLR W                        نغير عليهم  0x30
Repeat :    ADDWF     30, ø                         0x31
            DECFSZ    Temp, 1
            ~~return~~    ~~loop~~
            GoTo      repeat
            the       return
```

Generating Time Delays

- In many applications, it is required to delay the execution of some block of code; i.e. a time delay!
- In most microcontrollers this can be done by
  - Software
  - Hardware (Timers)
- To generate time delay using software, *<u>let the microcontroller execute non useful instructions</u>* for certain number of times!
- If we know the clock frequency and the cycles to execute each instruction we can generate different delays

$$Delay = \#cycles \text{ x } clock\ cycle\ time$$

$$= \underbrace{\boxed{\#\text{cycles}}}_{\text{trace the code}} \text{x } \boxed{4 / \text{F}_{osc}}\ \text{Tinst}$$

Generating Time Delays

- **Example 5:** Determine the time required to execute the following code. Assume the clock frequency is 800KHz. Fosc

$T_{inst} = \frac{4}{800\,kHz} = 5\,\mu s$

Delay = Tinst · # of cycles

$= 5\mu s \times [(\overset{movlw}{1}+\overset{movwf}{1}) + 199 \times (\overset{nop}{1}+\overset{nop}{1}+\overset{dec}{1}+\overset{goto}{2}) + (\overset{nop}{1}+\overset{nop}{1}+\overset{dec}{2})]$

$\cong 5\,ms$

loop

```
        movlw     D'200' ; initialize counter
        movwf     COUNTER
del     nop              ; main loop for delay
        nop
        decfsz    COUNTER, F
        goto      del
```

- What if this code to be used as a subroutine??!!

Literal addressing: to deal with values directly, no memory involved

Direct & Indirect
dealing with data memory

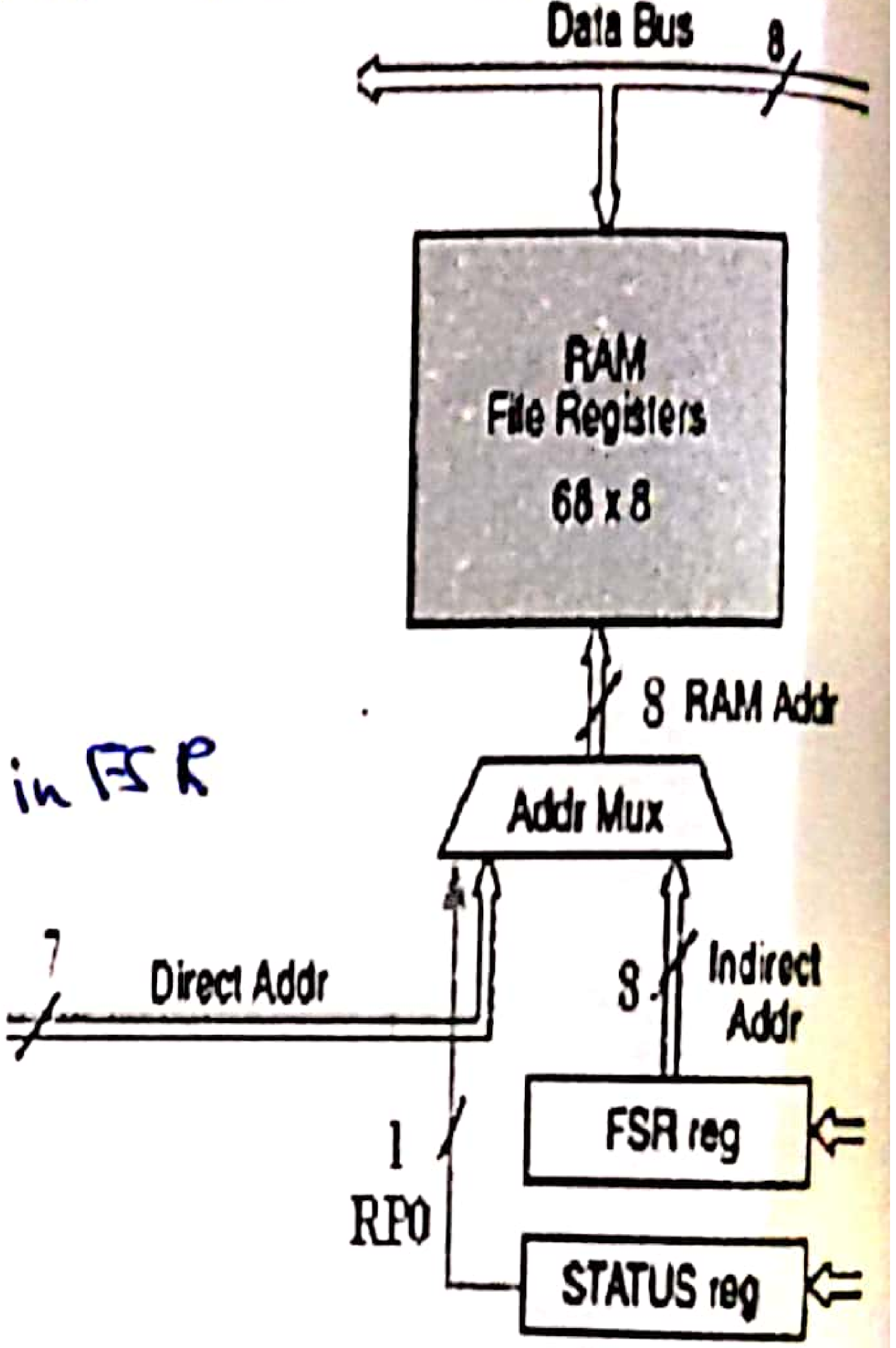
Working with Data

Direct addressing: address of the value is in the instruction itself

Indirect Addressing

used for manipulating addresses

- Direct addressing is capable of accessing single bytes of data
- Working with list of values using direct addressing is inconvenient since the address is part of the instruction
- Instead, we can use indirect addressing where
  - The File Select Register FSR register acts as a pointer to data location. address of the value is in FSR
  - The FSR can be incremented or decremented to change the address
- The value stored in FSR is used to address the memory whenever the INDF (0x00) register is accessed in an instruction
- This forces the CPU to use the FSR register to address memory


Data Bus
8
RAM
File Registers
68 x 8
8 RAM Addr
Addr Mux
7
Direct Addr
8
Indirect Addr
1
RP0
FSR reg
STATUS reg