* Embedded : not a general purpose computer
  ↳ Real time : There is a dead line after which actions are useless
    ↳ Hardware + mechnical parts

* To understand any embedded system :
1- inputs
2- outputs
3 - user interaction
4 - Link to other systems
} ⇒ any user
5 - Hardware : MC & other data sheets
6 - Software → Have access to the code
  ↳ Take backup & enjoy
  ↳ not : ask for code
    ↳ SW driven every component in the E.S
      controlled by the SW

* Reliable : 99.9 % good , 0.1 % Bad
  almost 100 % correct results

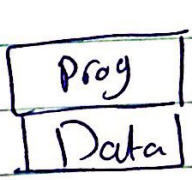
  ↳ drivers input & environment variables & temp & humedity & noise

* MC is a computer on chip :
it has the same components of the computer

1- CPU : processing, controlling, and executing the code
2- inputs / outputs : To deal with external world
3- Memory : storage
↳ Data memory → Data holds variable values

Volitile : Lost when the power is down , why?
→ Variable values have meaning during code execution on power down, the prog is executed from begining (RAM) , [Fast writting & consumes less power]

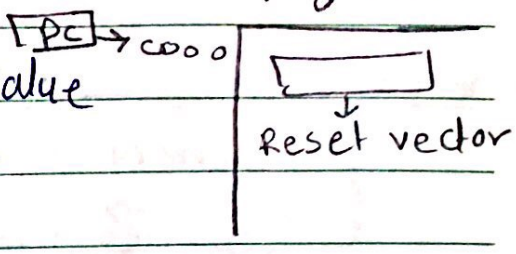
→ Program memory → holds prog (instructions) [permenant] kept on power down (ROM)



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

- if PC = 5, the next instruction that the CPU will execute is at address 5

- Reset vector is the default start value of the PC on Power up or reset



PC is auto increment

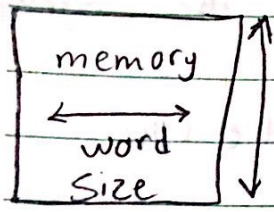
Fetch is getting the instruction from program memory into instruction regester (IR)

* Pipelining: While the instruction is executed, the next instruction is fetched

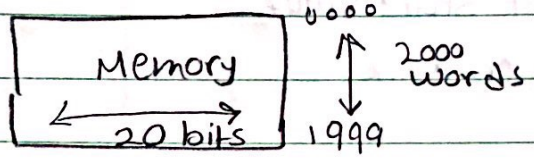
The * Von Neumann archetucture (VNA)
Less complex

The Harvard architecture (HV)
→ Parallel: Faster due to pipeling
→ variable bus sizes on each module

data W.



EX

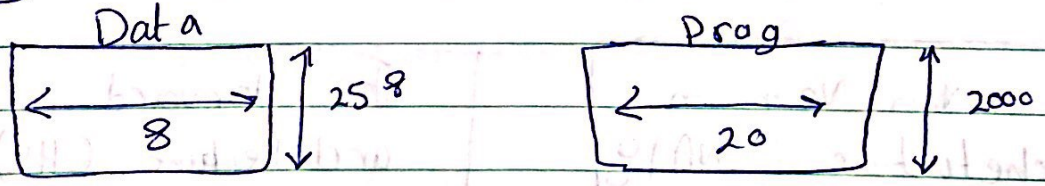


min data bus = 20 bits

min address bus = $\lceil \log_2 2000 \rceil$ = 11 bits

Size = $20 \times 2000 = 40,000$ bits = $\frac{40000}{8}$ bytes

EX



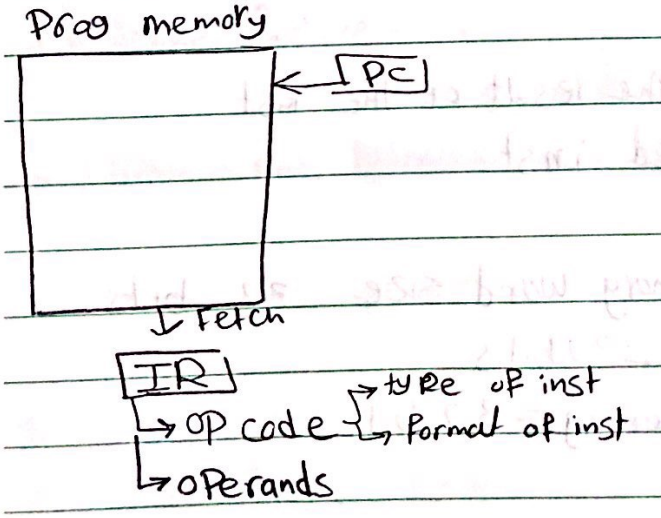
	Data memory	Prog memory
Harvard	// Data bus 8 bits; address bus // $\log_2 256$ // 8 bits	Data bus // 20; address bus // $\lceil \log_2 2000 \rceil$ // 11 bits
VNA	Data bus max	{8, 20} = 20 bits
	address bus max	{8, 11} = 11 bits

* Instruction set : Detailed descriptions for all instructions that you can use to write a prog.

* Syntax error: if the programmer did not ... to IS

- IS is like a contract between the manufactor & developer

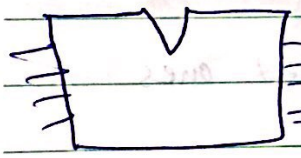
↳ RISC : only few simple instructions
fixed size insts, longer programs, faster execution
Pipelining is more efficant because all insts have same fetch and excute time ; long compilation time
↳ CISC : many instructions including complex ones
- variable size instruction
- should memorize more insts & format many addressing modes



- Reset : Restart Prog execution (PC ← reset vector)
- iterupt : code that is invoked by the HW event (eg: keyboard)

-Within the same family, all MC,s have the same core (CPU) and same I'S, But have different memory sizes and periphase

→ How do I know the family of a MC? Datasheets



Dual inline Packaging (DIP)
- Size of MC
  1- # of Pins based on # of I/Os
  2- inter Pins Spacing

working

W-Reg (Accumulator): Holds the result of the last executed inst

32 bits computer : data memory word size 32 bits
all variables are of size =32bits
data bus of the data memory = 32 bits
Variables ($0-2^{31}-1$)

8 bits M.C → Base line
↳ mid range
↳ high speed

→ Flash memory
16 F 87 X A → advanced tech
↳ Series # ↳ MC number in the given series
↓ dont care

C → CMOS

L → Lower Power

* *Stack: voltaile memory
automatically written and read
you don't have access to it, holds return address

*Pipeline Failure: when the fetch inst is not the one that will be executed
→ Flush the fetched inst and fetch a new one

if stack size = x levels,
you can do x nested calls

*Nested call: do call before return from Previous call

```
inst 1
inst 2
inst 3
call
inst 4
call
call
call
return
```

OPcode	operands	inst

* interupt vector & the address which is automatically loaded in the Pc when interrupt happens

*Possibly there are multiple interupts, the developer should write code to know the reason behind the interrupt

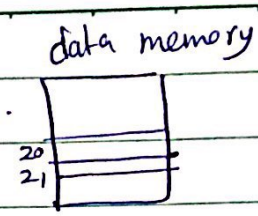
* multiple interrupt vectors, you can enforce preirties between them

508 / 509 ⇒ difference ⇒ memory size

*result of inst can be stored in W-Reg or in the data memory

* IF the inst uses 2 variables. (add, sub, and)
  1) W-Reg
  2) a- instruction it self (literal) eg: add values
     b- from the data menory
        ↳ address → instruction. (Direct)
                  ↳ Register called FSR (indirect)

12 F 508
509



add values in 20 & 21 X
move one of them to w-Reg
add the second variable

add value 5 to value in address 21X
move 5 to w-Reg
add [21]

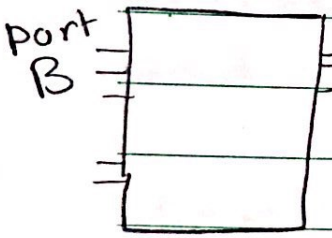
* Status reg (Flag reg): Gives information about last executed inst

* watch dog timer: it resets M.C when crach happens

* MCLR : when =0 "Reset"

CH: 2

EEPROM: holds data permanently for data that isn't changed freq.
eg: settings, Phone numbers



VDD: Power supply
Vss: Ground
$\overline{MCLR}$: External reset
OSC1, OSC2 } external OSC

I/O
8 bits timer

عدد البتات = 14
Size of address bus = 13 bits
Size of literal bits = 8 bits
level of stack = 8 ., nested calls = 8

reset vector = 0000
interrupt vector = 0004
word size in stack must equal address → (13)

* Configuration word: holds information about the Program configuration

↳ can be written or modified only at Prog download time
→ To modify it you should re-download the Program

1) Fosl II Fosϕ : defines the OSC that can be connected to the PIC

2) watch dog timer enable WDTE : enables disable WDT

3) $\overline{\text{PwRTE}}$: $\phi \Rightarrow$ on Power up, the MC stage in reset mode for a while

4) Code protector: cp = $\phi \Rightarrow$ code is protected (no one can read it)

2/10/2018

* On Power up, I want to keep the µ.C in reset mode for a while → External → ☐ - HW
↳ internal

* On Power up, the capacitor starts charging, and when it gets charged after time related to $\tau = RC$, the input to MCLR = 1 ⇒ exit from reset
- R_S to prevent high currents entering to the PIC
- Free wheeling diode to reset instantly (very fast)

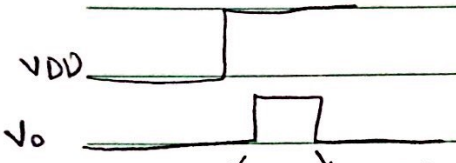
The 16F84A on-chip Reset circuit

S	R	Q	$\bar{Q}$
0	0	Q	$\bar{Q}$
0	1	0	1
1	0	1	0
1	1	unstable	

q. Reset is connected to $\bar{Q}$ and it is active low?

q. when does reset happens?
when S=1 and R=0

q. when does S=1 & R=0?
1- $\overline{MCLR} = 0$
2- WDT & not in sleep mode
3- On power up



Reset ↙ ↘ S=0, R=0 } no change, keep it in reset

* When does the PIC exit from reset mode after power is up?
when R=1

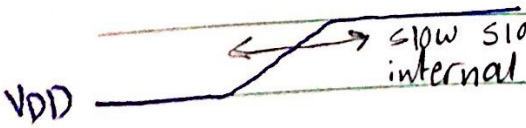
① IF enable PWRT = 1 (enabled)
we need 1024 cycles from the internal RC OSC (72 ms)
- if PWRT is disabled: the output = 1 immediatly

② After the first Counter finishes counting, the second counter starts counting, and after 1024 cycles of the external OSC ⇒ output = 1

③ IF both enables are enabled, how long the PIC stay in reset mode after Power

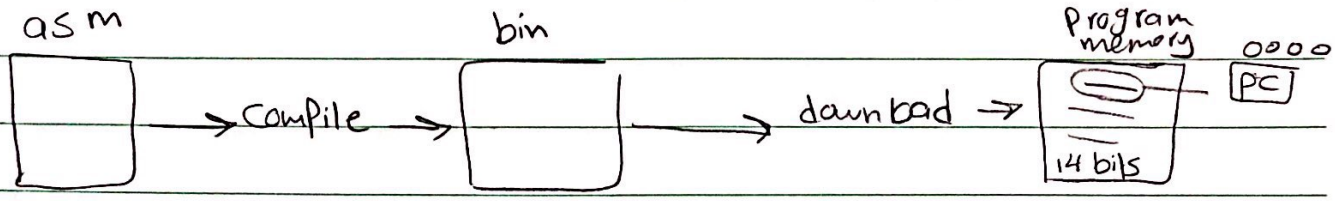
* How to enable PWRT? The configuration word has a bit
* How to enable OST? IF you choose one of the OSC types: XT, HS, or LP, then it is enabled
if you choose ~~RC~~ OSC, then it is disabled

Config Word



4/10/2018

Chapter "4"



HLL (High level language) → compile → intermediate lang (assembly) → bin
OR
Assembly → assembler → bin } → more efficent [less lines & time]

---

* The PIC 16 series instruction set

- 35 instructions
- Result is stored either in W-Reg or (d=0)
in data memory (d = 1)

- Types of instructions:
1- Byte oriented File register F(7bits), d (1bit) [op F,d]
2- Bit oriented File register R (7bits), b (3) [op F,b]
3- Literal K (8 bits) [op K]
4- Control: change path of execution K(11 bits) [op K]

instruction : | opcode | operand |

ADDLW k
ADDWF f, d

SUBLW k
SUBWF f, d

INCF f, d
DECF f, d
COMF f, d

1- OP F, d
2- OP F, d
3- OP k
4- OP k

ADDLW 2 [w] = [w] + 2
ADD 20, 1 [20] + [w]
ADWF 21, 0 [w] = [21] + [w]

7/10/2018

Logical instructions:

AND WF	k	$A \cdot \phi = \phi$	clear
AND LW	f, d	$A \cdot 1 = A$	
IOR LW	f, d	$A + \phi = A$	
IOR WF	k	$A + 1 = 1$	set
XOR LW	f, d	$A \oplus \phi = A$	
XOR WF	k	$A \oplus 1 = \bar{A}$	complement

We use the logical instructions for bits masking
Bit masking: make some bits = 0, 1, or keep the others

[Ex] Write one instruction to clear the least significant 4 bits of the W-Reg and keep the others AND LW F0

[Ex] Write a code to complement even bits of address 20
XOR WF 20, 1
(W ← 55)

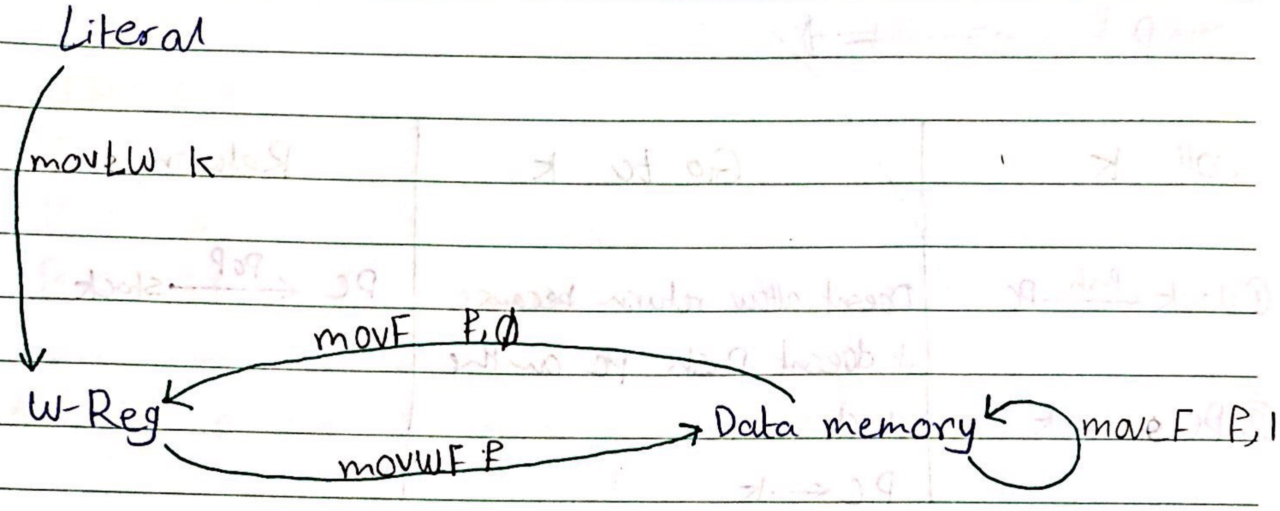
[Ex] Write one instruction to complement the bits with odd position in W-Reg
XOR LW AA

[Ex] Set most sig 4 bits of wReg IORLW F0

[Ex] Complement address 22
comF 22, 1

The flags are affected regardless the destination bits.

~~# The Flags are~~



* To initialize any ~~memo~~ data memory location with a value

1) movLW k
2) movWF F

* To copy any data memory location to another 20 → 21

1) MOVF 20,Ø
2) MOVWF 21

• مش موجود movLF (F, k) عشان مابسمح inst size (رح يصير كبير)

(assembly is case sensitive)

move 20, 1
[20] ← [20]

if you want to check if address[21] = Ø
movF 21, 2
Check 2 flag

SWAPF F, d
SWAPF 21, Ø

~~Swap f, d~~

~~Swap f, 1, 0~~

call k	Go to k	Return
① stack ← Push PC	Doesnt allow return because it doesnt Push PC on the stack	PC ← Pop stack
② PC ← k	PC ← k	

* conditional branchs Go to if condition
- In PIC skip if condition (conditional skip)

INC FSZ	F, d	increment address f and skip next inst if the result is zero regardless of d
DEC FSZ	F, d	
BTFSS	F, d	Bit # b in address F=1
BTFSC	F, d	skip the next inst

● Conditional skip:

INCFSZ DECFSZ	IF
BTFSS	IF, else
BTFSC	Loops

*Write a code to multiply the value in address 22 by 4

```
BCF   STATUS, C
RLF   22, 1
BCF   STATUS, C
RLF   22, 1
And   B'11110110
IOR   WF  B'
IOR   WF  0X01, 0
```

complement even bits

OP F, d (7bits, 1bit)

OP F, b (7bits, 3bits)

6 bits op code

4 bits op code

Literal opk ← 8 bits

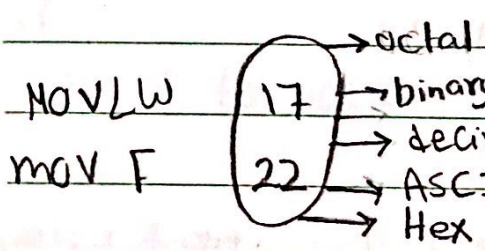
Go to opk ← 11 bits

Reg 14 bit total

$\lceil \log_2 35 \rceil = 6$

14/10/2018

* Label: Should be Placed at the begining of the line
  Case Sensitive
  Once it is defined, it can be used as operand and its value equal the address of first inst comes after it



the value starts with letter (A-F), it should be preceded by either 0X or H

Assembler Directives: Gives the assembler some information at compilation time, and then they are discarded

---

#include : it opens a file and you can use anything inside it
#include PIC16F84A INC : you can use all regs and flags by their names

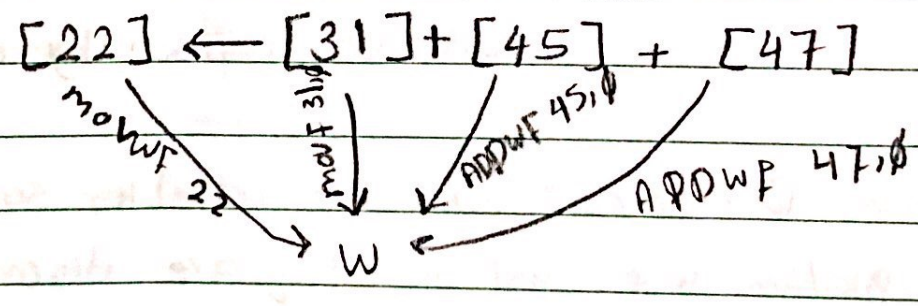
```
BSF STATUS, RP0
ORG  5 : it tells the assembler that inst1 after being compiled,
       inst1 it must be stored at address 5
       inst2
ORG 0000 inst3
start
ORG 0004
ISR
```

3) EQU : it defines a const

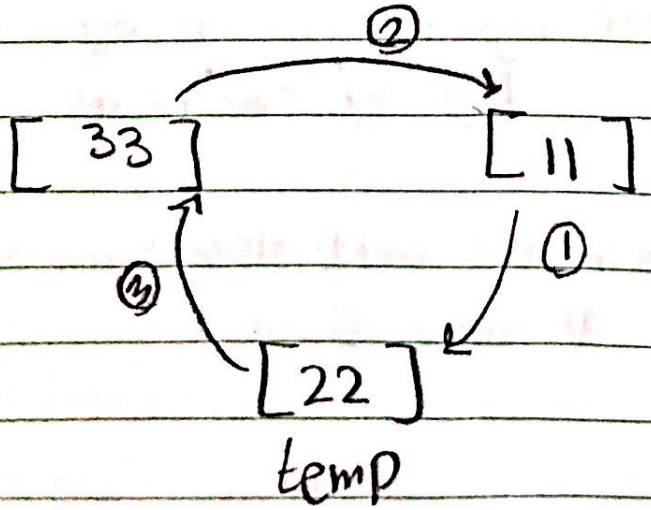
```
STATUS EQU 03
RP0  EQU 5
BSF  STATUS, RP0
```

```
Clock  20
   Var1  =  Var1  EQU 20
   Var2  =  Var2  EQU 21
   Var2     Var3  EQU 22
endk
```

end : it tells the assembler not to compile anything beyond this point



Sample Program 2 (swap)




```
MOVF  11,0
MOVF  22
```

16/10/2018

* conditional skip

BTFSS f,b

BTFSC f, b

INCFSZ f,d

DECFSZ f,d

• Example:

if

add 5 to address 227

else

add 9 to address 22

```
Goto    ADD5        movLW   5
Goto    ADD4        SUBWF   22,0    [22]-5
Add5    movLW 5     BTFSS(C) STATUS,C  [22]>5⇒C=1
        AddWF 22,1  GoTo    ADD5
        Goto  next  movLW   9
add9    movLW 9     ADDWF   22,1
        addwF 22,1  GoTo    NEXT
Next                ADD5    MOVLW   5
                    ADDwF   22,1
                    NEXT
```

---

* Exercise for Home: if [21] = 0)
add 2 to [22]
if ([22] > 5)
add 5 to [22]
else
add 7 to [21]

- For (i= 15; i>0 ; i--)
  sub 3 From address [21]

```
Counter   EQU   25
MOVLW     D'15'
MOVWF     counter

Label :   MOVLW   3
          SUBWF   21,1

          DECFSZ  Counter,1
          GOTO    Label
```

For (i=0; i <15 ; i++;)

```
Counter   EQV.    25
movLw     0
movwF     Counter
movLW     3
SUBWF     21,1
INCF      Counter,1
MOV LW    D'15'
SUB WF    Counter,0
BTFSS     STATUS,2
GOTO      Label
```

- Exercise : For (i=7; i ≤ 18 , i++)

Slides: Conditional branching. Example 1:

[0x11] + [0x22]

if = 0

33

else

44

```
        movf    11, 0
        addwf   22, 0
        btfs    5   status, c
        Goto    Label 1
        Goto    Label 2
Label 1 movwf   33
        Go To   NExT
Label 2 movwf   44
NEXT
```