# Microcomputer Based System Design

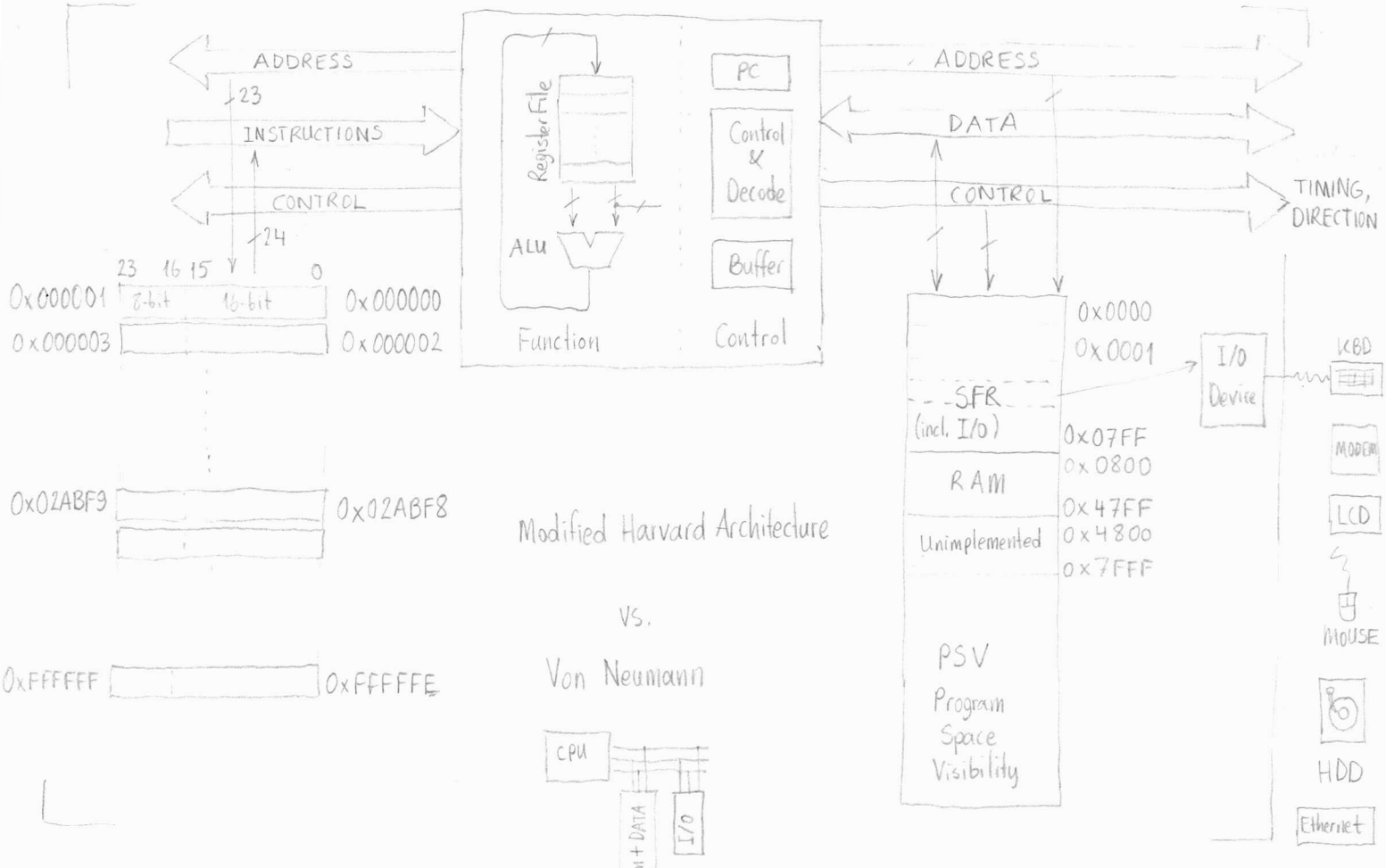
Microcontroller : A small computer on a single integrated circuit consisting internally of a relatively simple CPU, clock, timers, I/O ports and memory. (Found in cellphones, MP3 players, TV remote)

(def'n from Wikipedia)
(Merriam Webster entry 1971)

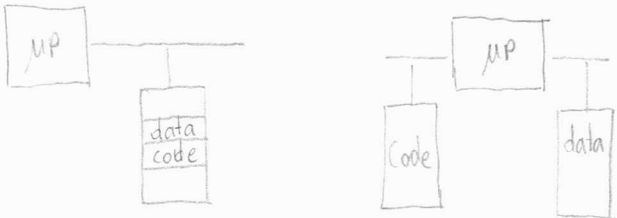
Discrete components on single PCB → Microcomputer (eg., PC)

* CPU: Part that carries out (i.e., executes) program instructions and controls peripheral devices.
* Memory: ① Holds program instructions
  ② Stores and/or provides data generated and/or required by programs.
* I/O Devices: Coordinate the flow of data into and/or out of the microcontroller.

CPU is composed of
- a control unit to sequence processing and decode program instructions
- an arithmetic/logic unit to carry out operations
- registers and accumulators
- an address buffer to hold the address of the next instruction (instr. pointer or prog. count) IP PC
- input/output buffers to hold data flowing to/from processor (cache, instr. queue)

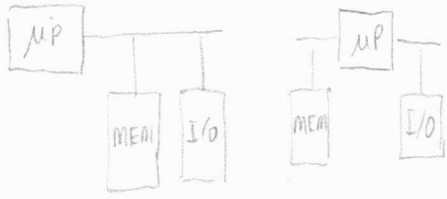


Various architectures (all have ① µP
② Memory for data & instructions
③ I/O )




Von Neumann vs. Harvard

Variations :

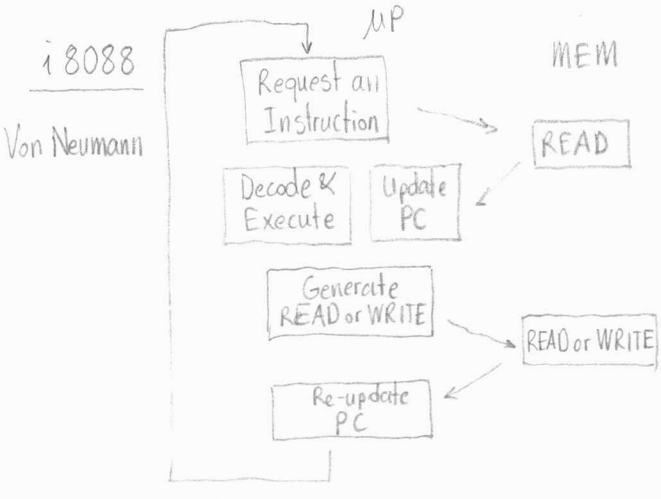



Modified Harvard : Program memory visible in data memory (as in PIC24)

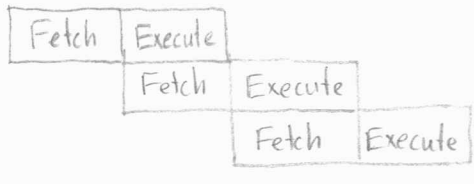
Other examples of microcontrollers : 386EX, AMD Geode, SGS Thompson STPC, Analog Devices ADuC (8051, Arm), TI MSP 430

## Sequencing of Processor-Memory Interactions

- PROGRAM COUNTER holds address of next instruction to be executed
- The INSTRUCTION is fetched and decoded
- The decoded instruction is executed
- Execution might generate memory/I/O READ/WRITE
- PROGRAM COUNTER modified (normally +1, or +(instruction length) or JUMP)



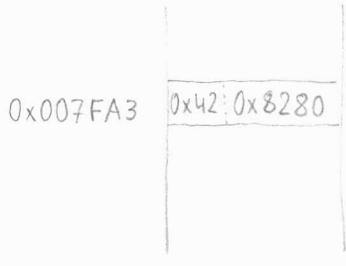

PIC24
Harvard




Pipelined Architecture

Ex PC = 0x007FA2

0100 0010 1000 0010 1000 0000

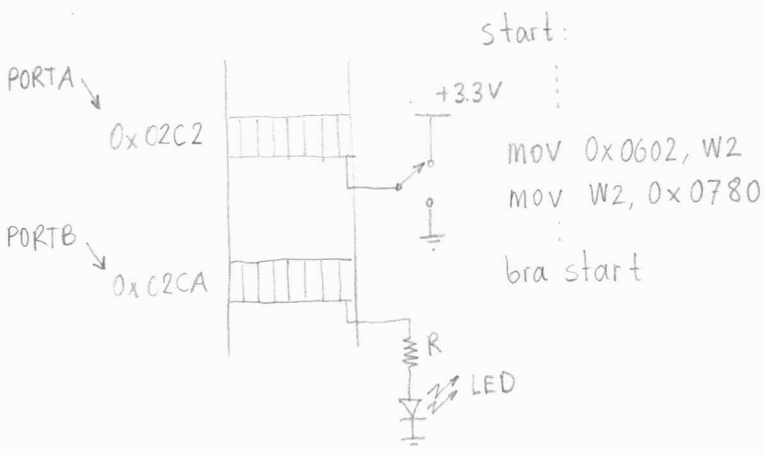


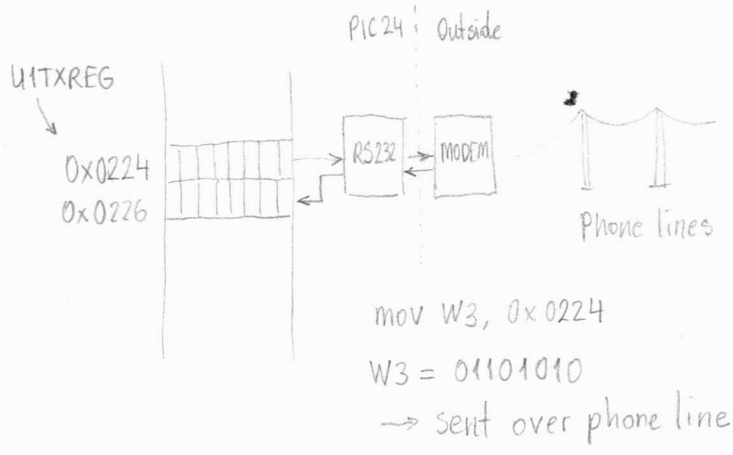

0x007FA2 → Fetch → 0x428280 → Decode → add W5, W4, W1

Execute →

| | Before | After |
|---|---|---|
| W1 | 0x0700 | 0xBCF7 |
| W4 | 0x1C02 | 0x1C02 |
| W5 | 0xA0F5 | 0xA0F5 |

# I/O Interface




## MEMORY

* Fixed response
* Generic (identical cells)
* Easy to interface

## I/O

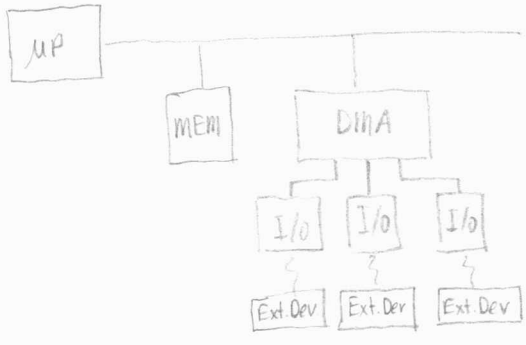
* Interfaces to external device
* NOT synchronized (events at any time)
* More dedicated interfacing

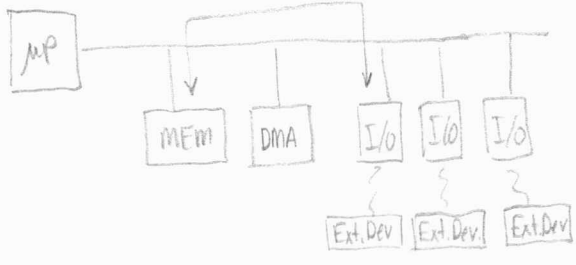
Two synchronization schemes

① Polling (Periodic status checking)
② Interrupts

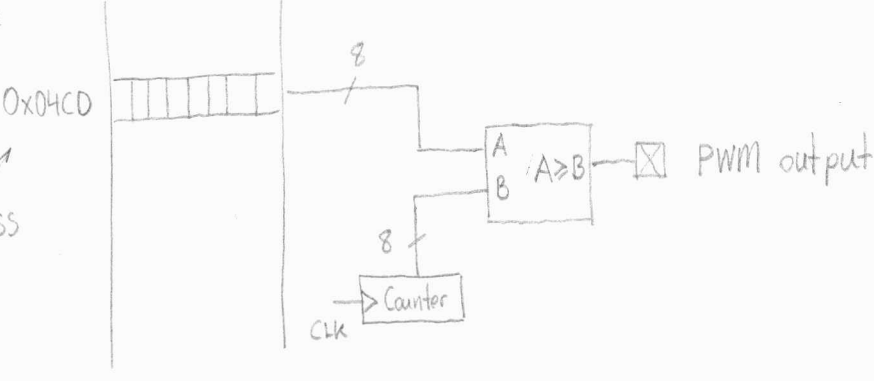
## Other Features (not to be discussed)

- DMA (direct memory access)
- Cache




* Another I/O Port Example (which requires a lot of additional circuitry)



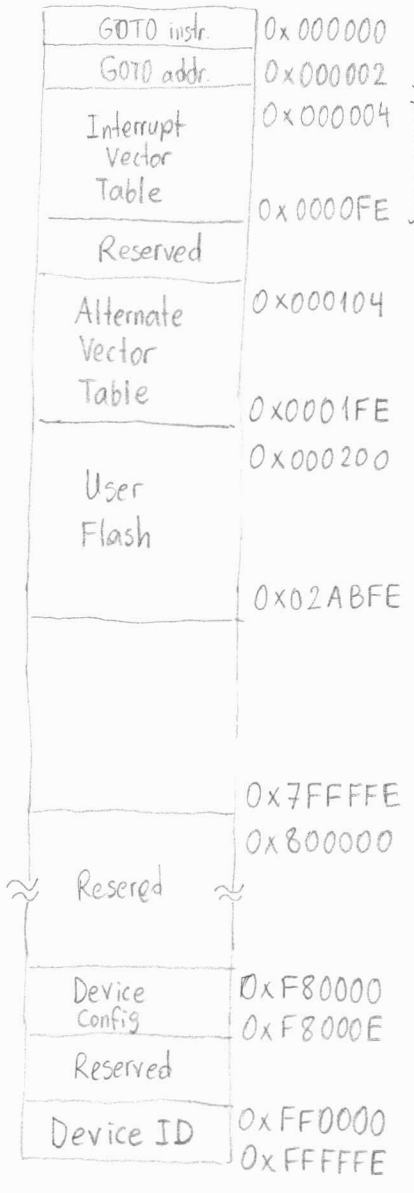
Program memory: 256K or 87552 instructions (PIC24FJ256GB110 datasheet p11)

↓

262656 bytes = 256,5 kBytes

87552 instructions → last instruction address is 87552x2-1 and 87552x2-2

87552x2-2 = 175102 = 0x02ABFE




GOTO is a 2 instruction word long instruction

0xFE - 0x04 = 0xFA = 250

→ 250/2 = 125 different interrupt sources possible (all of them not used)

PC → 24 bits with LSB=0 (always)

Max. PC (without going into the reserved area) is 0x800000 - 2

⇒ 0x400000 instructions ⇒ 0x400000 x 3 = 0xC00000 bytes = 12582912 bytes

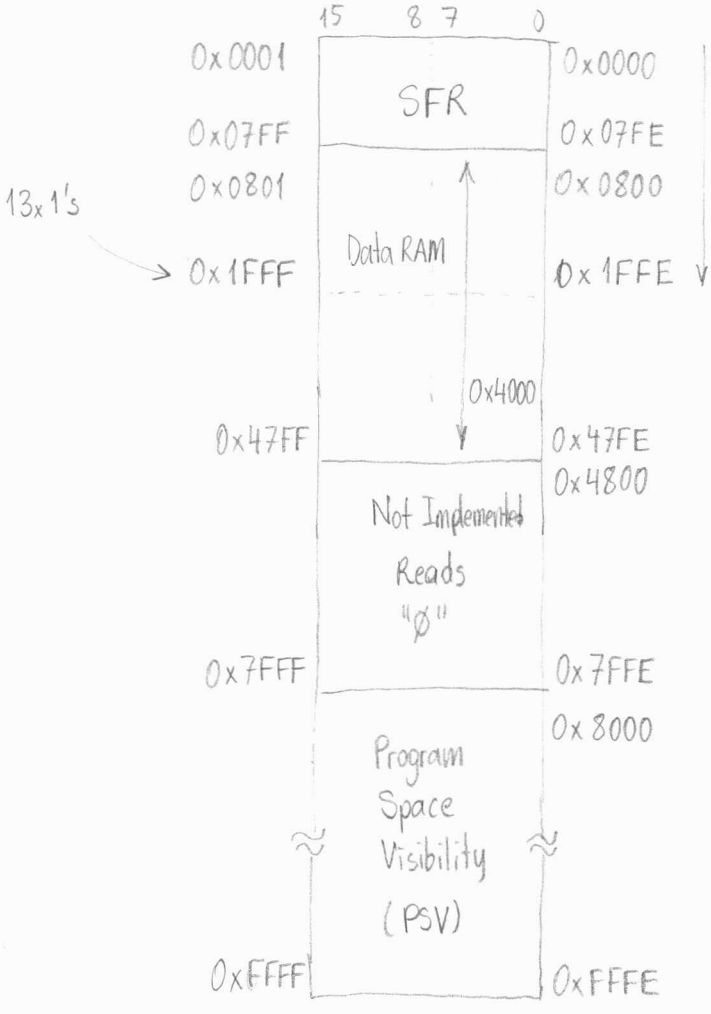
= 12288 kBytes

= 12 MBytes

max. # of bytes the program memory can have.

## Data Memory:

16384 bytes = 0x4000

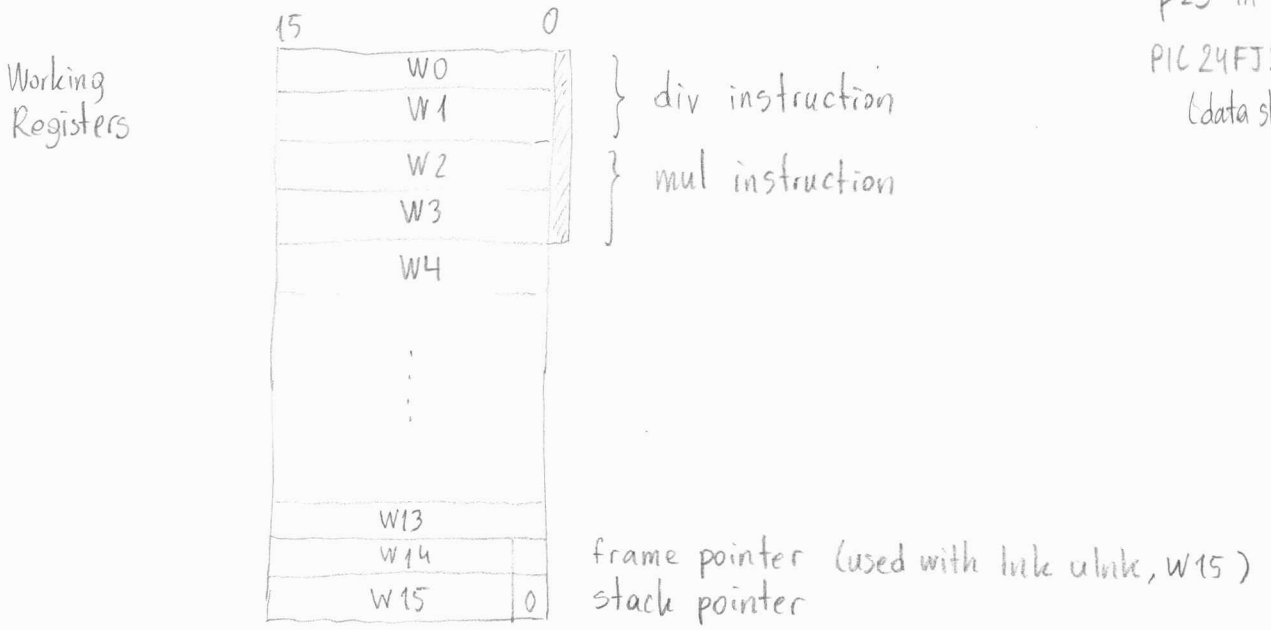


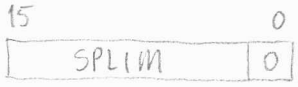

near data:

MOV W1, 0x0CF2 type of addressing with absolute address

# CPU Registers

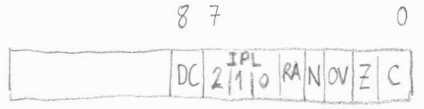
p29 in the PIC 24FJ256GB110 manual (data sheet).

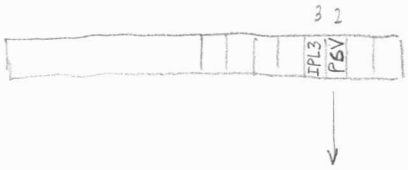









C: carry
Z: zero
OV: overflow → carry from #14 to #15 sign bit change
N: negative
RA: repeat loop active
IPL → interrupt priority level
DC → Half carry/borrow bit (DAW.B)

Check DS39703A Section 2. CPU.

DC → used in conjuction with the daw.b instruction:

Packed BCD addition:

```
    W0 = 0x0018
+   W1 = 0x0009
-------------------
    W2 = 0x0021   incorrect BCD (DC=1 after addition)

daw.b W2
+ 0x0006
-------------------
    0x0027   correct BCD result
```

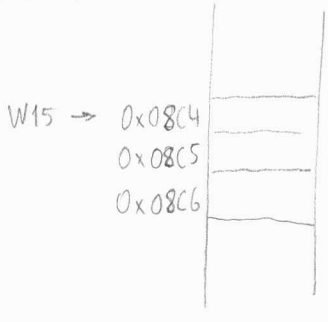
Also,

```
    W0 = 0x0012
+   W1 = 0x0009
-------------------
    W2 = 0x001B   DC=0 but DAW.B works! as B>9 invalid BCD

daw.b W2
+ 0x0006
-------------------
    0x0021   correct BCD result
```

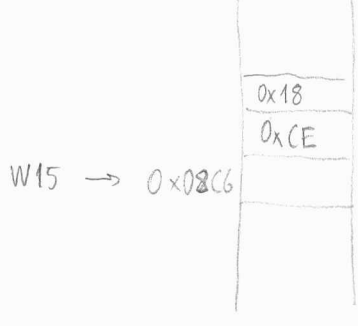
## Stack Maintenance

W0 = 0xCE18
W15 = 0x08C4




push W0 →
≡ mov W0, [W15++]

W0 = 0xCE18
W15 = 0x08C6




pop W0 →
≡ mov [--W15], W0

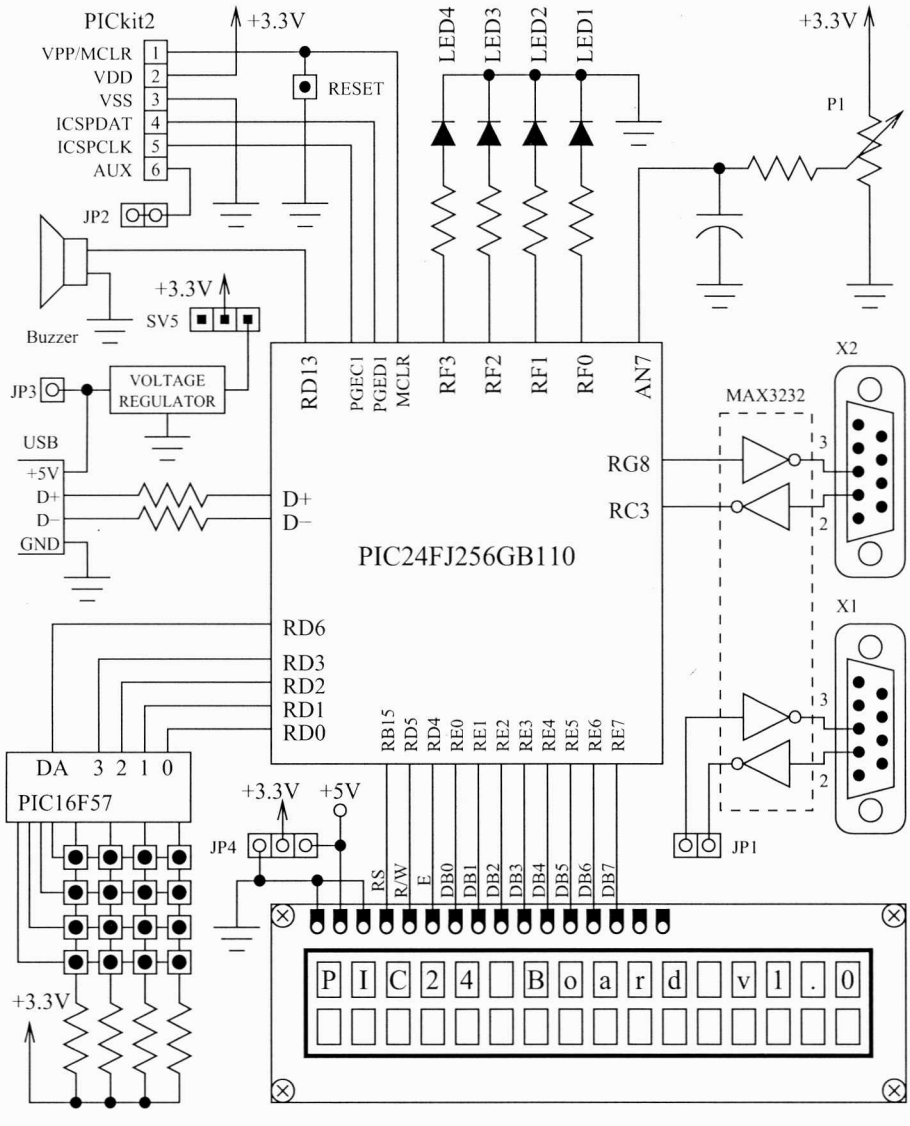
1. CPU...pdf
p8

start:

```
btss  PORTD, #6
bra   start
mov   PORTD, W0
mov   #0x000F, W2
and   W0, W2, W0
mov   W0, PORTF
bra   start
```

not to change other PORTF bits:

```
mov PORTF, W1
mov #0xFFF0, W2
and W1, W2, W1
or  W0, W1, W0
```




From now show

dsPIC Programmer's Reference

but before show one subcircuit specific manual such as 14.Timers...pdf

with (1) Schematics
(2) Registers

Also emphasize bset T1CON, #TON
type of bit manipulation instructions.

## Types of PIC24F Instructions

① Data Transfer → mov, (means "copy") — push, pop

② Arithmetic/Logic → add, addc, and, asr, com, sl

asr: ↓ arithmetic shift right (preserves sign); com: ↓ complement; sl: ↑ shift left (logic = arithmetic)

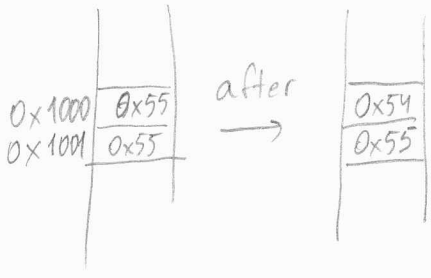
logical → lsr

## Data Addressing Modes (dsPIC Programmers guide, section 4.1)

| | Address Range | |
|---|---|---|
| i File Register | 0x0000 - 0x1FFF | (1st 8 kBytes) |
| ~~ii~~ Register Direct | 0x0000 - 0x001F | (Working register array W0:W15) |
| ~~iii~~ Register Indirect | 0x0000 - 0x~~FFFF~~ | |
| ~~iv~~ Immediate | n/a | |

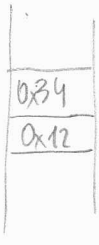
File Register → Near RAM 1st 8kByte (13 bit literal)

i) dec 0x1000




EXCEPTION mov instruction can get a 16 bit literal

mov 0x27FE, W0 (0x27FE: ↓ even)




after W0 ← 0x1234

→ for other examples see manual

③ Branching
- bra
- btsc, btss
- goto
- ~~call~~

④ Control transfer
- call
- ~~#####~~

⑤ Processor Control
- reset
- clrwdt

| Arithmetic | Logic |
| --- | --- |
| add | and |
| addc | asr arithmetic shift right |
| dec | |
| dec2 | bclr, bset } bit set/clear |
| div.s | |
| div.u | |
| | cp, cpø } compare / compare with ø |
| inc | |
| inc2 | |
| [illegible] | ior inclusive or |
| | |
| mul.s | xor |
| mul.u | |
| | |
| neg | |
| | |
| sub | |
| subb | |

① Data Transfer dsPIC 33 Programmers Reference

* mov {.b} f, WREG f = 0 : 0x1FFF (near data)

mov f, WREG f → wØ
mov f f → f (STATUS affected)

* mov.{b} WREG, f wØ → f

* mov f, Wn } f = 16 bit
mov Wn, f }

* mov lit16, Wn
mov.b lit8, Wn

-2 : - 0000 0010
1111 1101
+ 1
1111 1110
SL ↓
1111 1100 → -4
0000 0011
1
0000 0100

② → Examples from manual SL p299

Sign is preserved.
Sign change indicates overflow. Hence no ASL or LSL

Example 1 sl.b 0x909

Data at 0x0908 : 0x 9439 → 0x 48 39 ?

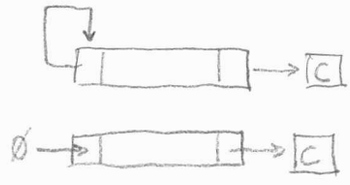
1001 0100 → 0010 1000

Ex 2 0x4065 → 0x80CA

0100 0000 0110 0101 → 1000 0000 1100 1010

ASR p98

LSR p210




④ bra label → 16 bit argument moves by Slit16 +32768 +3767 p 105

(PC+2) + 2x Slit16 → PC
nop → instruction register

bra Wn → (PC+2) + 2xWn → PC

p 106

## call Expr ← label → resolved to a 23 bit program address

Types of Instructions (4)

∴ call instruction has two words (instruction + 23 bit literal)

Operation:

(PC) + 4 → PC
(PC<15:0>) → (TOS)
(W15) + 2 → W15
(PC<23:16>) → TOS
(W15) + 2 → W15
lit 23 → PC
NOP → instruction register (23 bit literal won't be executed)

## call Wn → example on next page

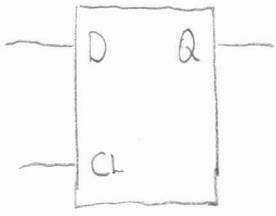
(PC) + 2 → PC
(PC<15:0>) → TOS
(W15) + 2 → W15
(PC<23:16>) → TOS
(W15) + 2 → W15
0 → PC<22:16>
(Wn<15:0>) → PC<15:1>

← bit 0 ignored (is always 0)

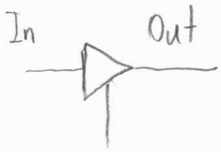
similarly bra → 1 instruction word
goto → 2 instruction word

bra C, label1 → suppose label1 is located more than 32767 instruction words away →

bra NC, label_new
goto label1
label_new:



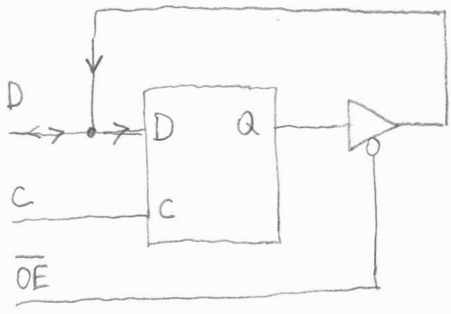

## Level triggered D-latch (1 bit storage)



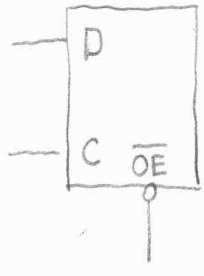

tri-state buffer

OE
Output Enable

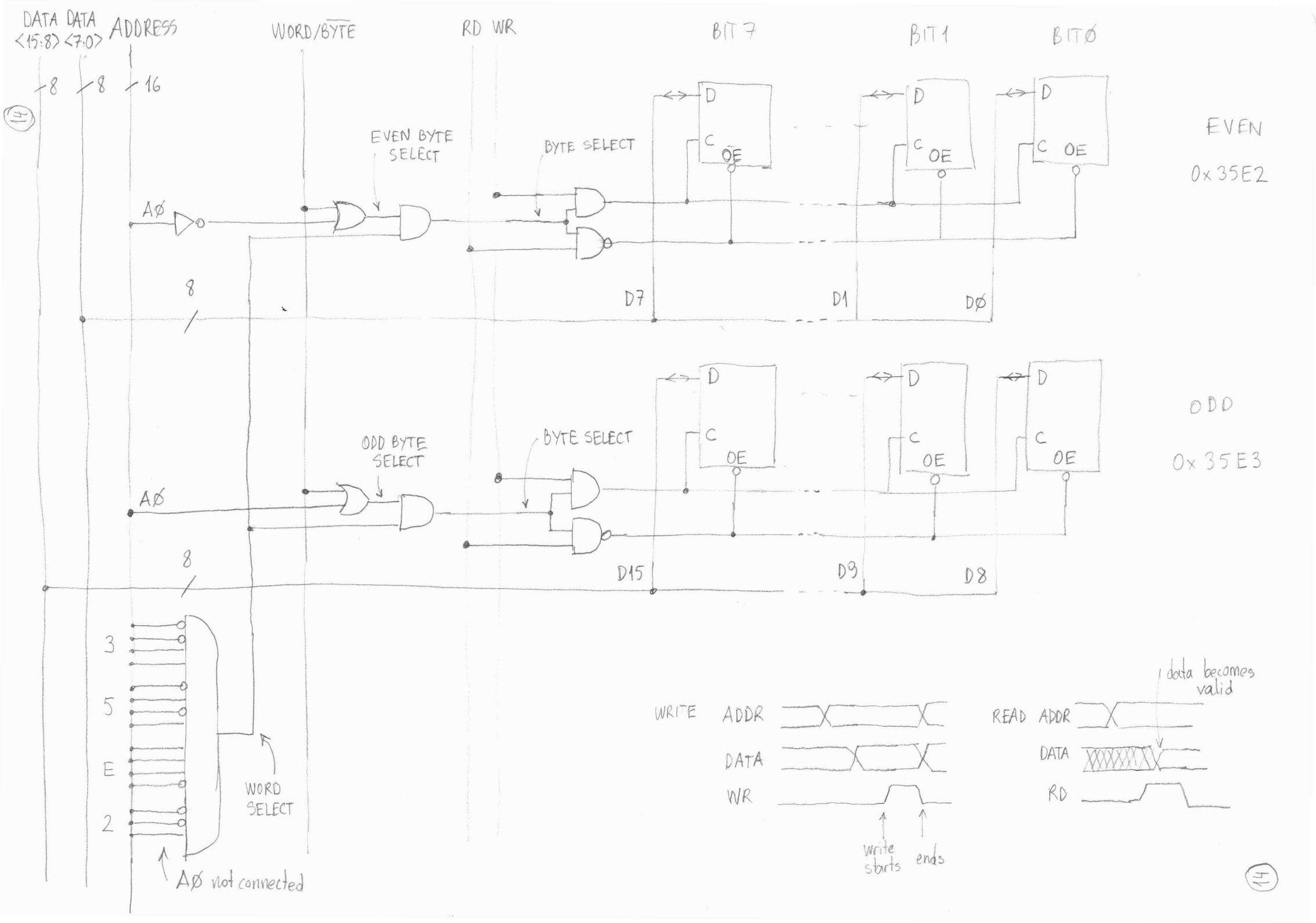
| OE | In | Out |
|---|---|---|
| H | L | L |
| H | H | H |
| L | X | High-Z |


≡




Bi-directional level triggered D-latch
with output enable

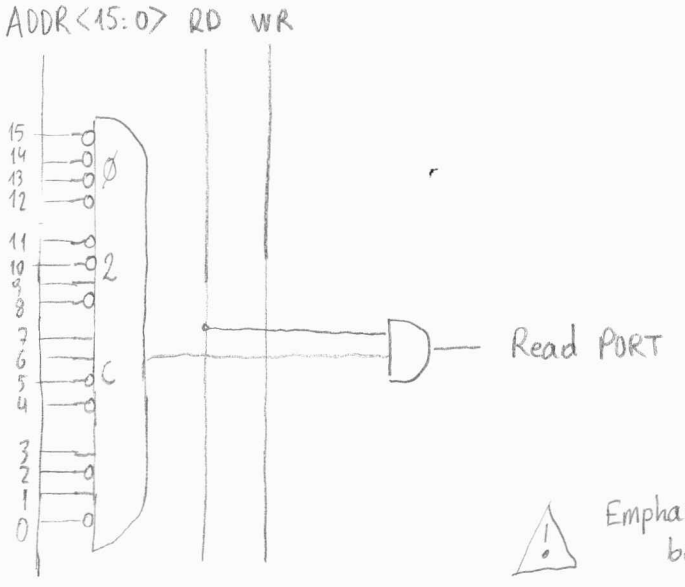

DATA DATA ADDRESS
<15:8> <7:0>
8 8 16
WORD/BYTE
RD WR
BIT 7
BIT 1
BIT Ø
EVEN BYTE SELECT
BYTE SELECT
AØ
8
D7
D1
DØ
EVEN
0x35E2
ODD BYTE SELECT
BYTE SELECT
AØ
8
D15
D9
D8
ODD
0x35E3
3
5
E
2
WORD SELECT
AØ not connected
WRITE
ADDR
DATA
WR
write starts
ends
READ ADDR
DATA
RD
data becomes valid

## PORTB Register Map

TRISB 0x02C8
PORTB 0x02CA
LATB 0x02CC
ODCB 0x02CE

Document to open:

Section 12. I/O ports with Peripheral Pin Select (PPS) (39711b)




⚠ Emphasize difference between

bset PORTB, #2

and

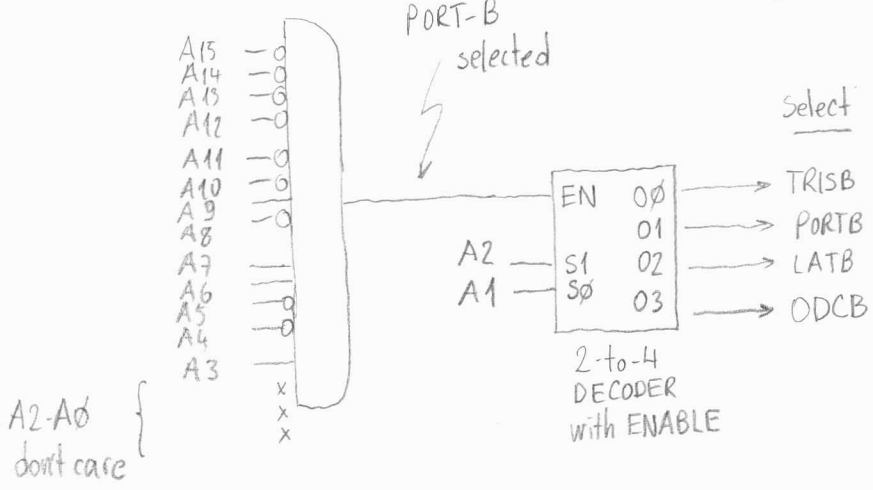
bset LATB, #2

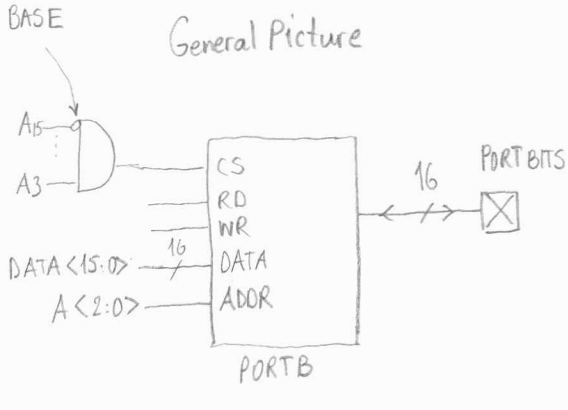
(consecutive writes to port bits may result in incorrect settings)

| | | 3 2 1 0 |
|---|---|---|
| ODCA 0x02C6 | 0000 0010 1100 | 0110 |
| TRISB 0x02C8 | 0000 0010 1100 | 1000 |
| PORTB 0x02CA | 0000 0010 1100 | 1010 |
| LATB 0x02CC | 0000 0010 1100 | 1100 |
| ODCB 0x02CE | 0000 0010 1100 | 1110 |
| TRISC 0x02DØ | 0000 0010 1101 | 0000 |

Base address of PORT-B is 0x02C8

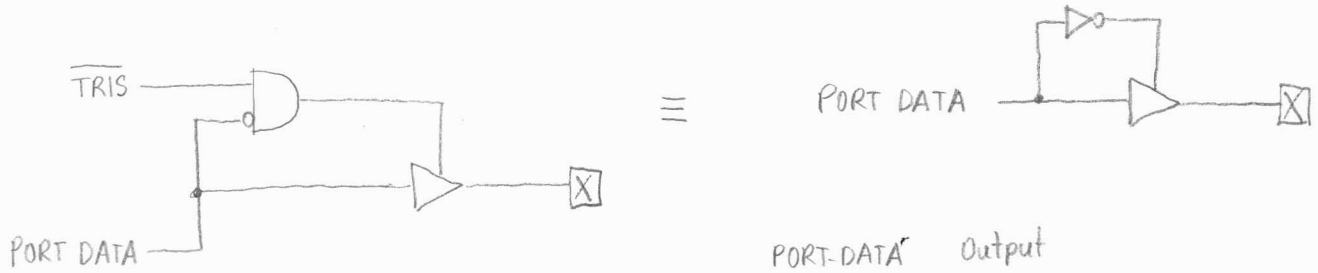
TRISB = BASEB + Ø
PORTB = BASEB + 2
LATB = BASEB + 4
ODCB = BASEB + 6



## ODC Open-Drain Control

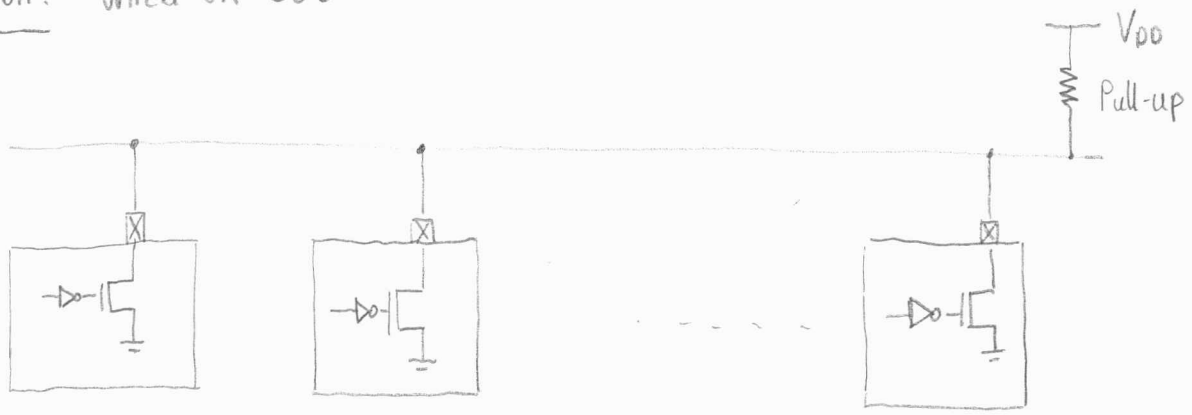
TRIS = Ø → PORT-bit used as OUTPUT




| PORT-DATA | Output |
|---|---|
| L | L |
| H | High-Z |

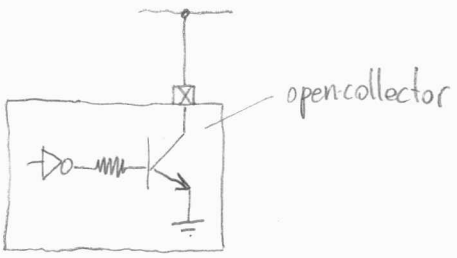
All of the above is equivalent to the following Circuit:


## Application: Wired-OR BUS




Advantages

① Wired-OR: OR function implemented without additional components.
② # of devices on BUS is flexible (simply tie the output of any additional device to BUS)
③ Supply voltage might be different for individual bus members (but requires HV compliant inputs)
④ Mixed IC technology

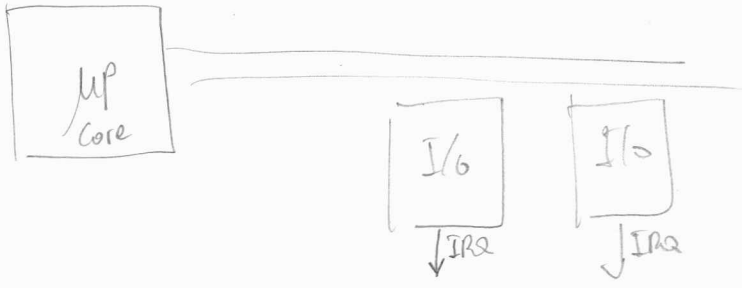



EXAMPLES

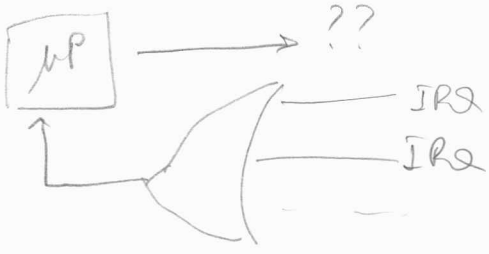
① $I^2C$ (Inter-IC) bus
② CAN bus (automotive)

31/03/2010

# Interrupt Scheme

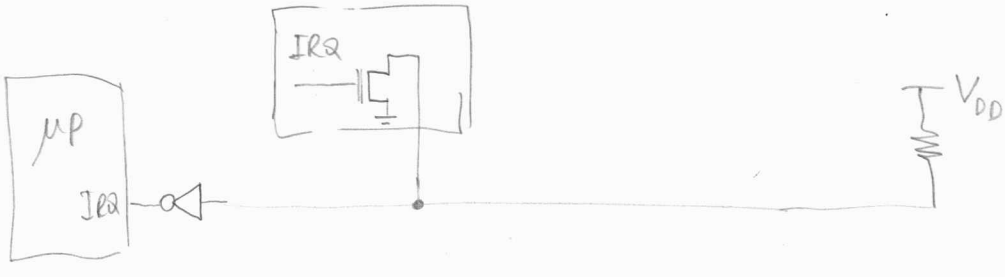



① Polling -

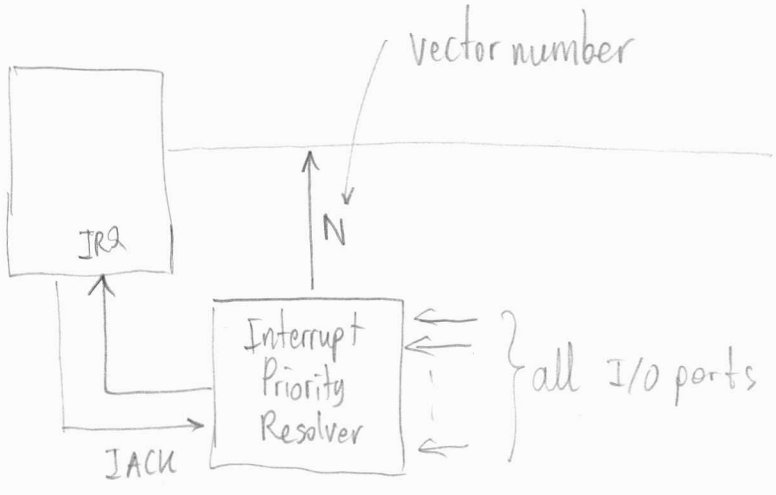



Priority : - Fixed
- Rotating
- Move served device to bottom of list.

Wired-~~OR~~




②



# Interrupt Set-up

On Port :
1. Set a proper value for interrupt priority
3. Enable port interrupt by setting the ENABLE bit
2. Clear the interrupt status flag bit

In your program
1. Declare the ISR name as global ← check the appropriate .gld file for names or page 160 of the MPLAB Assembler Users Guide
2. Set up the ISR

In the ISR
1. push used registers
2. do necessary action
3. clear the interrupt status flag bit (otherwise the interrupt source is blocked)
4. pop registers
5. execute "retfie"

When interrupt arrives
1. Current instruction's execution is finished
2. Priorities are checked
3. PC and bits of SR pushed
4. PC ← address at IVT (or AIVT)

When "retfie" executed
1. PC ← PC from stack
2. SR ← SR from stack