basic input/output

embedded system functionality = **processing** + **storage** + **communication (interfacing)**

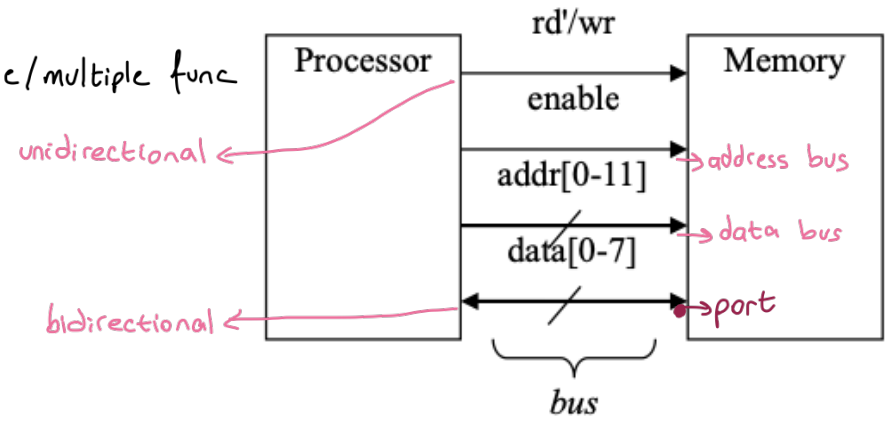
- data transformation
- using processors

- data retention (holding)
- using memory

- data transfer btw processors and mem
- using I\O ports and busses

conducting = iletmek

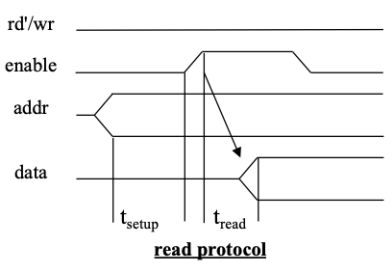
bus = single/multiple func

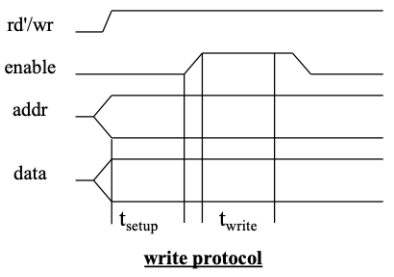


- single line can be multiple wires

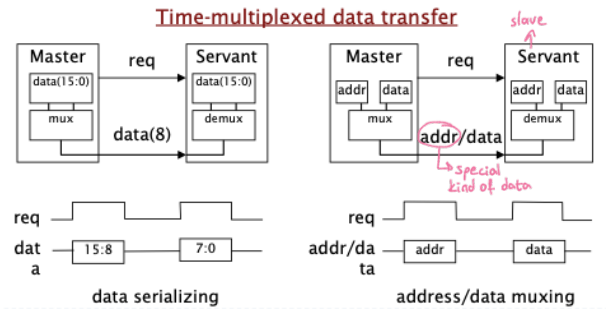
port = connects bus to processor or memory
- group of pins which can be assecced simultaneously (parallel)
- register inside a µC connected by wires to the pins of a µC
- single function wires = 12-wire address port

timing diagrams
- control signal = low or high
  - ↳ assert means active, which can be low or high
- data signal = not valid or valid

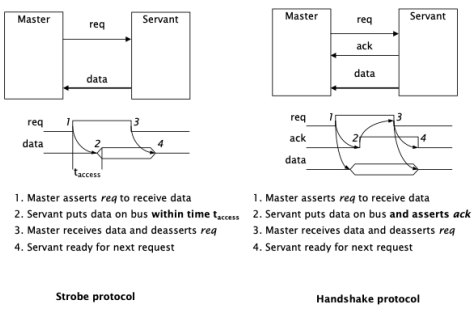


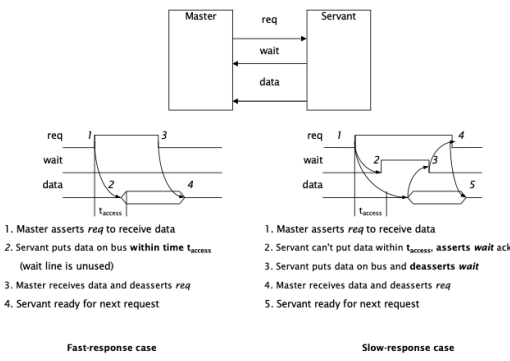


protocol = a set of rules and guideliness for communicating data



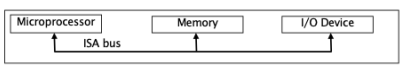
- control methods





interfacing: I/O addressing = processor communicates with other devices using some of its pins
- ↳ **port based/parallel i/o** = for each port there is a register → faster but too much
- ↳ **bus based i/o** = single bus → slower

ISA bus control = processor talks to both memory and peripherals using the same bus →



- ↳ **memory mapped I/O** = peripheral registers occupy addresses in same address space as memory (lower 32K memory, upper 32K peripherals)
  - ↳ no special instructions are needed ↳ accessing I/O devices like they are memory → in PIC
- ↳ **standard I/O** = additional pin on bus indicates whether a memory or peripheral access (when pin(M/IO) is 0 → all 64K mem, 1 → 64K perip)
  - ↳ requires special instructions (in/out) to move data between peripheral registers and memory ↳ in intel
  - ↳ no loss of memory to peripherals

PIC I/O = I/O ports are like RAM locations with wires leading from bits to the pins of the microchip
- ↳ memory mapped
- A, B, C, D, E, (F, H, J) = 8-bit, G = 6-bit bidirectional ports, they can be input or output
- each bit of ports has a direction bit, some can be inputs while others are outputs

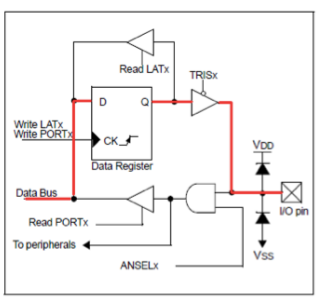
↳ **TRISxReg** = configuration controller, controls the direction (I or O) of PORTx, **1 bit input**, **0 bit output** → in bank15 → defalt value is 0

↳ **PORTx Reg** = pin I/O access, **WRITE** → changes output val (data latch), **READ** → read from external pin → it is not a physical flip-flop / latch

↳ **LATx Reg** = port data latch (mandal), both Read and Write accesses the val being output (data latch), used for read/modify/write operations

TRIS = 0 (output, write)

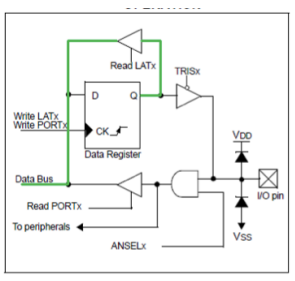
* you can write to latch or port (would be identical)



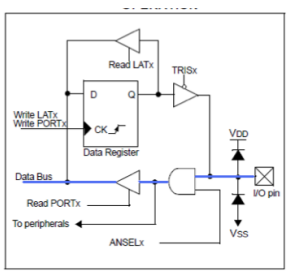
* if you write to PORT in input mode, only latch is updated
- writing to LATCH, reading from the PORT to avoid read/modify/write problem

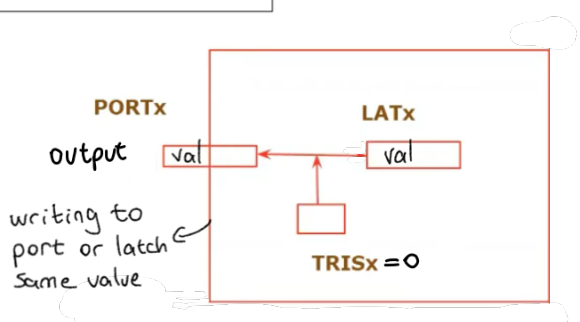
TRIS = 1 (input, read)

* reads last written val from latch

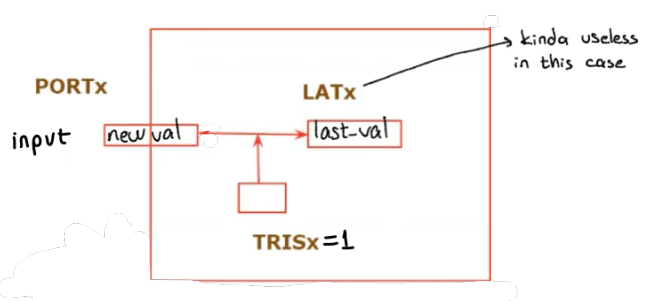


* reads pin val if read from port





LATA	0	0	0	0	0	0	0	0
TRISA	0	0	1	0	0	1	0	1
PORTA	0	0	?	0	0	?	0	?
DIRECTION	OUT	OUT	IN	OUT	OUT	IN	OUT	IN

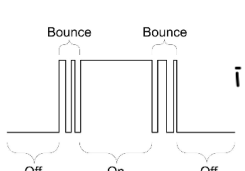


basic I/O programming = infinite loop that never exists → round-robin / cyclic executive
- ↳ event driven

In C:
```
while (1) {
    task_1();
    ...
    task_n();
}
```

In assembly:
```
LOOP:
    call task_1
    ...
    call task_n
    bra LOOP
```

button bouncing =



in hardware transitions are not smooth, check 'on' signal after a while