

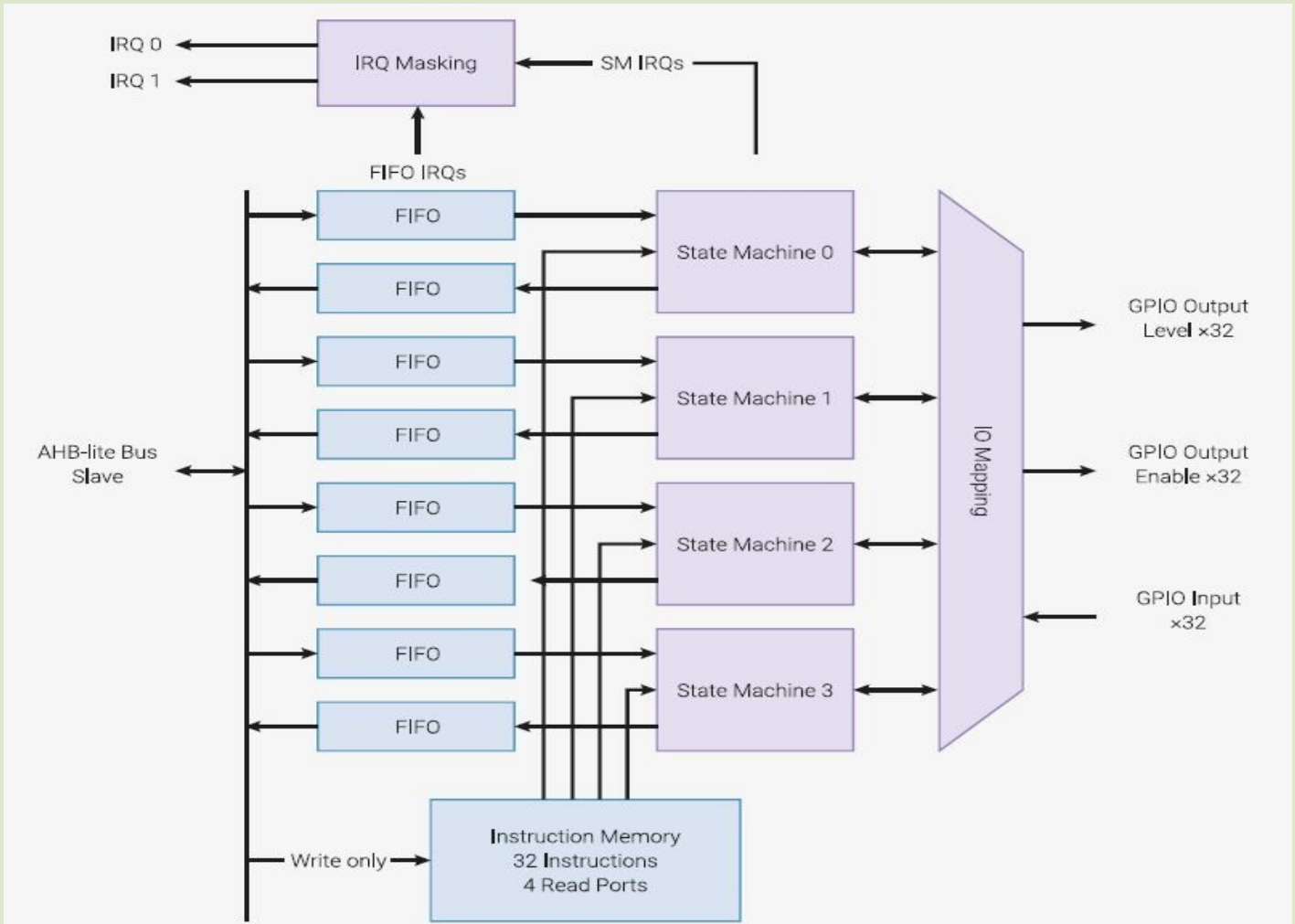
# PIO overview



**Autumn FIG leaf with RP2040**  
**In it two PIO's with**  
**each four state machines**

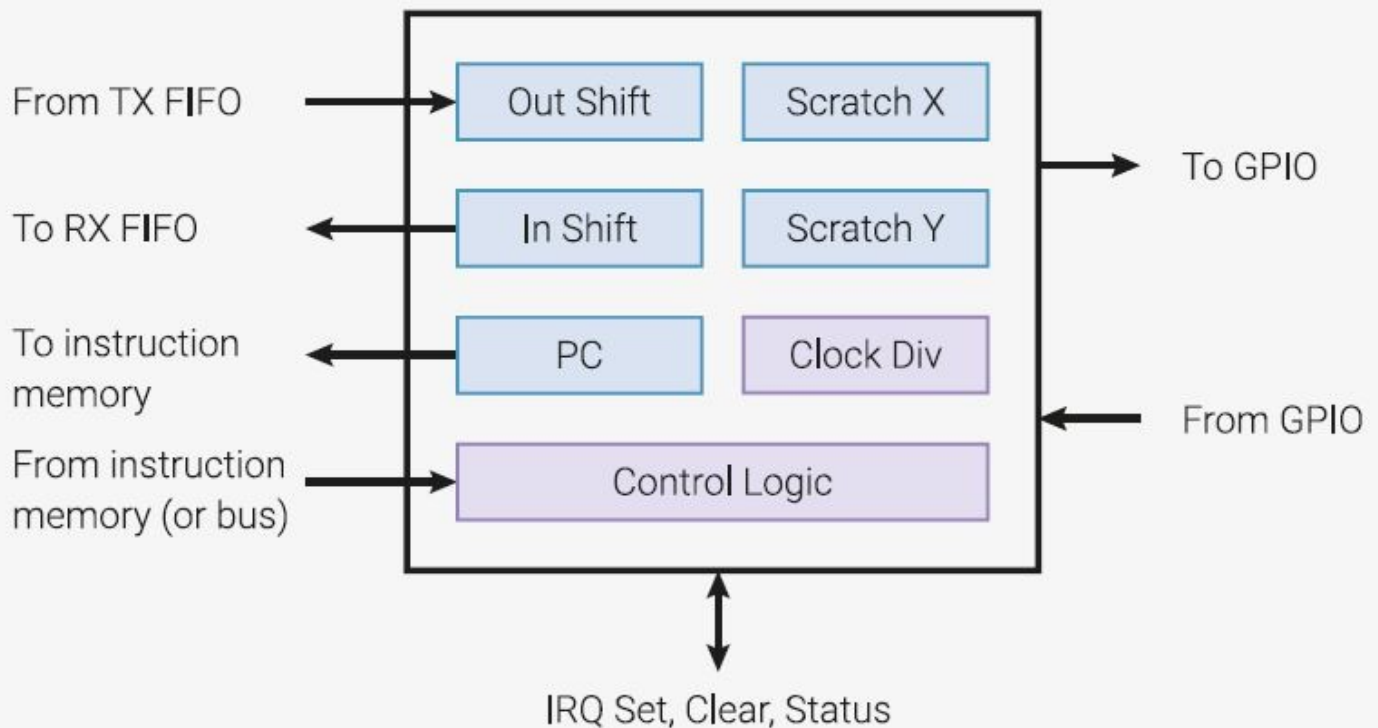
# PIO introduction

## Components of a PIO



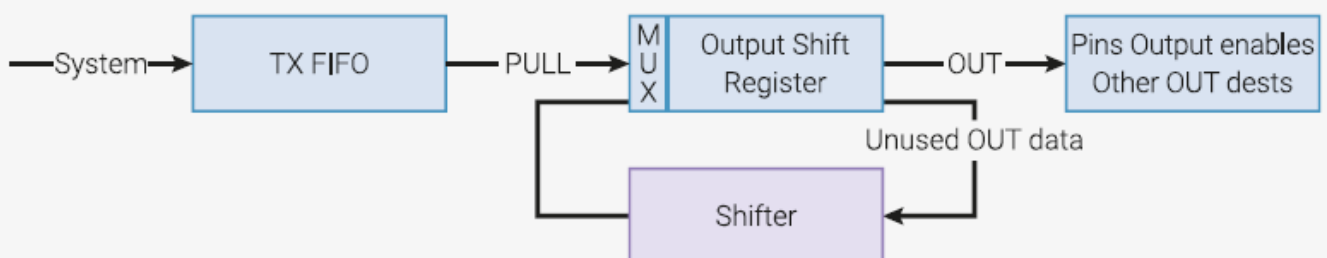
- **Four programmable state machines**
- **Each two queues (FIFO's) of 4 cells deep**
- **Shared program memory for 32 instructions**
- **Interrupt capability**
- **Eight opcodes**
- **A set of general control registers**
- **A state machine also has 6 dedicated registers**
  - **Clock frequency, Shift register control**
  - **Pin controls & Direct execution of opcodes, etc.**

## State machine structure



- **Two scratch registers (X & Y)**
- **Shift register in & out with queue**
- **Instruction pointer (PC)**
- **Clock divider**

## Output Shift Register (OSR)



- **Input comes from queue (FIFO)**
- **OSR output to PIN or scratch register (X or Y)**
- **May be coupled to input shift register (ISR)**
- **Can execute OSR data as opcode**

# The PIO assembler

## The SET opcode highlighted

### 3.4.10. SET

#### 3.4.10.1. Encoding

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SET	1	1	1	Delay/side-set					Destination			Data				

- **Five bits data (0 tot 31)**
- **Destination in 3 bits (X of Y register, PIN, etc.)**
- **Delay max. 5 bits (0 to 31 clock ticks)**
- **Side-Set are optional control's bits, these do steal bits from the delay!**

## First PIO program in Forth

```
clean-pio decimal \ Empty code space mirror

0 0 {pio          \ Use state machine-0 on PIO-0
  2000 =freq      \ On 2000 Hz frequency
  25 2 =set-pins  \ GPIO 25 & 26 for SET

  3 pindirs set,  \ Both pins are outputs
  begin, again,   \ Wait loop
wrap-target
  7 [] 3 pins set, \ LED on, pin 25 & 26 on
  7 [] 31 y set,   \ Max. delay using Y
  begin,
    7 [] 0 pins set, \ LED & pin off
    7 [] y--? until, \ Wait longer
wrap

  0 =exec          \ Start code at address 0
pio}
```

### PSEE

```
0: E083 set  pindirs 3
1: 0001 jmp  to: 1
2: E703 set  pins 3    [7]
3: E75F set  y 1F      [7]
4: E700 set  pins 0    [7]
5: 0784 jmp  y-- to: 4 [7]
```

## The program is controlled by Forth:

### hex

```
: FLASH      2 0 exec ;      \ Jump to address 2, start flasher
: LED-OFF    1 0 exec  E000 0 exec ; \ Pin 25 & 26 off, jump to wait loop
: LED-ON     1 0 exec  E003 0 exec ; \ Pin 25 & 26 on, jump to wait loop
```

## UART example:

```
clean-pio decimal \ Empty code space mirror
```

```
v: pio also definitions
```

```
: =BAUD ( b sm -- ) 8 * =freq ;
```

```
v: previous
```

```
\ With optional Side-set
```

```
0 0 {pio \ Use state machine-0 on PIO-0
```

```
\ 9600 =baud \ 9600
```

```
\ 38400 =baud \ 38k4
```

```
115200 =baud \ 115k2
```

```
26 1 =side-pins opt \ GPIO 26 for optional SIDE
```

```
26 1 =out-pins \ GPIO 26 for OUT & SET
```

```
26 1 =set-pins
```

```
1 pindirs set, \ Pin is output!
```

```
wrap-target
```

```
7 [] 1 side pull, \ Stop bit, get data byte
```

```
7 [] 0 side 7 x set, \ Start bit
```

```
begin,
```

```
1 pins out, \ Shift 8 bits out
```

```
6 [] x--? until, \ Until 8 bits are done
```

```
wrap
```

```
0 =exec \ Start SM at address 0
```

```
pio}
```

```
: Pemit ( ch -- ) \ Character to PIO UART
```

```
begin 0 tx-depth 3 < until 0 >txf ;
```

```
: PTYPE ( a u -- ) 0 ?do count pemit loop drop ;
```

```
: ABC ( -- ) s" ABC " ptype ;
```

```
: PICO ( -- ) s" RP2040 " ptype ;
```

## UART re-use:

```
clean-pio decimal \ Empty code space mirror
```

```
v: pio also definitions
```

```
: =BAUD ( b sm -- ) 8 * =freq ;
```

```
v: previous
```

```
0 0 {pio \ Use state machine-0 on PIO-0
\ 9600 =baud \ 9600
\ 38400 =baud \ 38k4
115200 =baud \ 115k2
26 1 =side-pins opt \ GPIO 26 for optional SIDE
26 1 =out-pins \ GPIO 26 for OUT & SET
26 1 =set-pins
```

```
1 pindirs set, \ Pin is output!
wrap-target
7 [] 1 side pull, \ Stop bit, get data byte
7 [] 0 side 7 x set, \ Start bit
begin,
1 pins out, \ Shift 8 bits out
6 [] x--? until, \ Until 8 bits are done
wrap
```

```
0 =exec \ Start SM-0 at address 0
pio}
```

```
: *EMIT ( ch -- ) \ Character to PIO UART-0
begin 0 tx-depth 3 < until 0 >txf ;
```

```
: *TYPE ( a u -- ) 0 ?do count *emit loop drop ;
: ABC ( -- ) s" ABC " *type ;
: RP2040 ( -- ) s" RP2040 " *type ;
```

```
1 0 {pio \ Uart output on pin 27, 38k4 baud on sm-1 & pio-0
0 clone \ Copy sm-registers from sm-0 to sm-1
38400 =baud \ 38k4
27 1 =side-pins opt \ GPIO 27 for optional SIDE
27 1 =out-pins \ GPIO 27 for OUT & SET
27 1 =set-pins
0 =exec \ Start SM-1 at address 0
pio}
```

```
: ~EMIT ( ch -- ) \ Character to PIO UART-1
begin 1 tx-depth 3 < until 1 >txf ;
```

```
: ~TYPE ( a u -- ) 0 ?do count ~emit loop drop ;
: BCD ( -- ) s" BCD " ~type ;
: PICO ( -- ) s" PICO " ~type ;
```