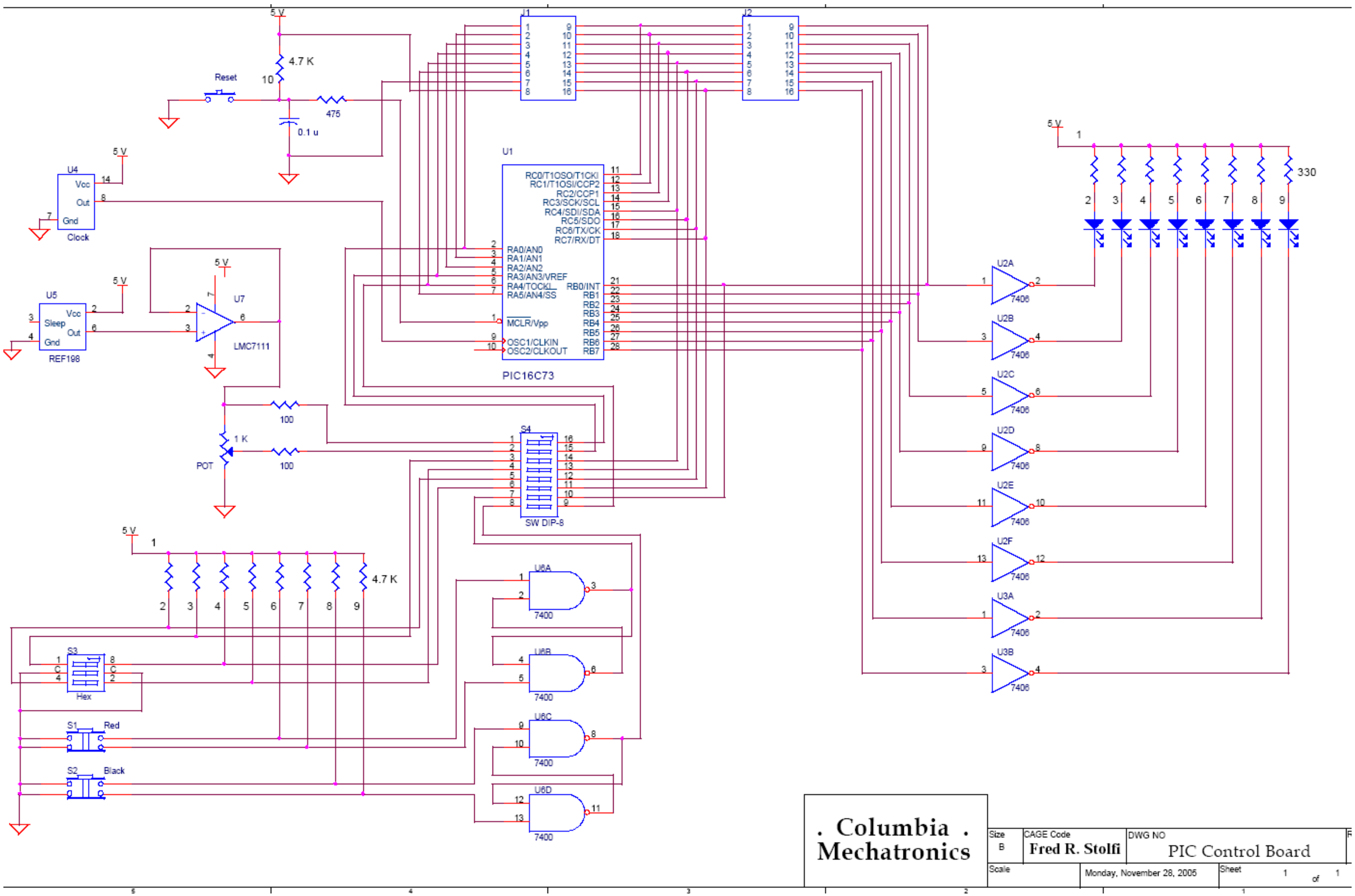
DC Motors

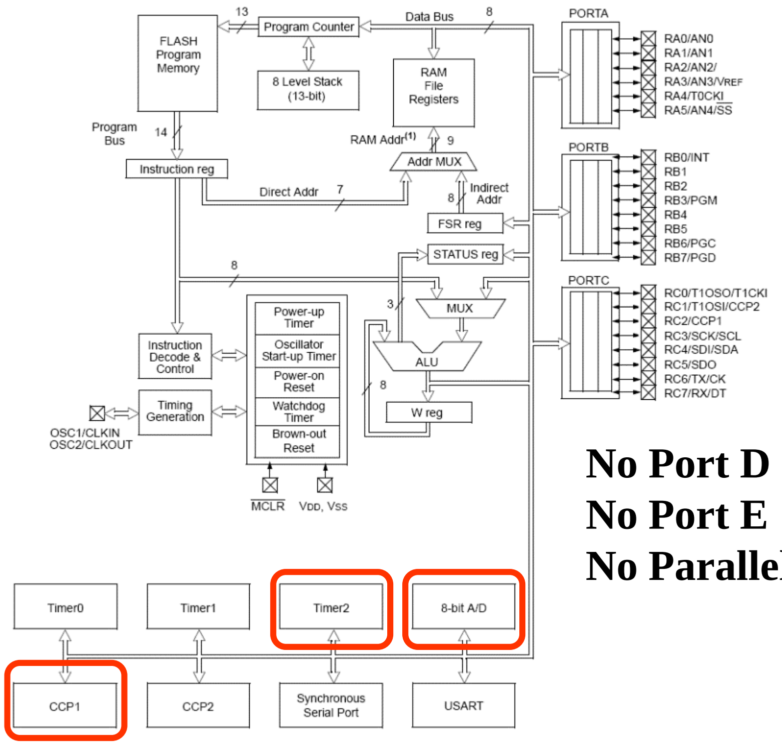
DC Motor Case Study

Interrupts & Pulse Width Modulation

Mechatronics Control Board



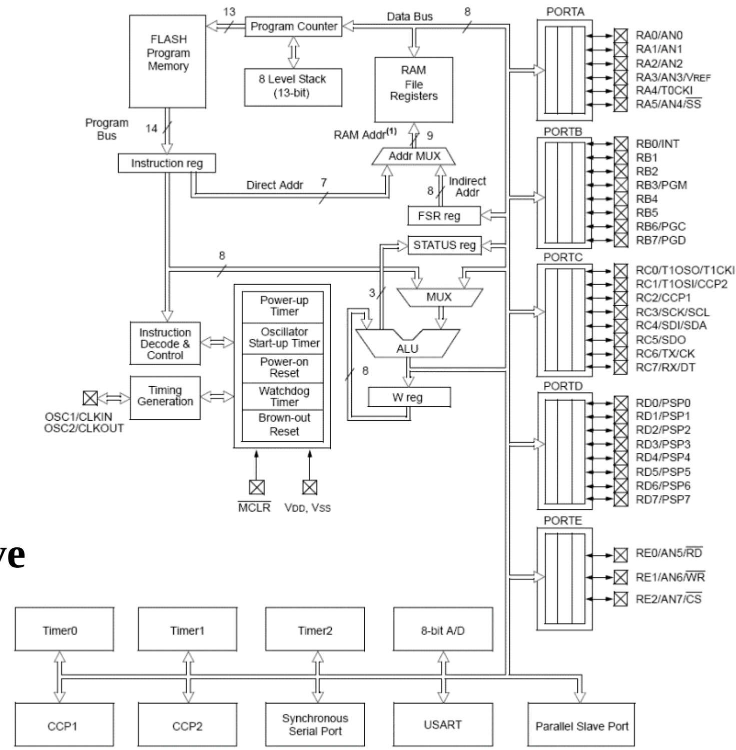
MicroChip PIC16F73 Microcomputer



No Port D

No Port E

No Parallel / Slave



PIC16F73

PIC16F74

Aspects of Embedded Programming

There are several aspects to programming an embedded MicroChip microcontroller which makes it different from other programming.

- Program and Data are in separate memory locations
- Input / Output pins have multiple uses which can be changed within a program (by manipulating special function registers)
- Limited memory addressing
  - Program memory is split into pages
  - Data memory is split into banks
- Data memory is memory mapped so the same instructions used for internal memory are used for input / output
- Program execution cannot halt – the processor must always be doing something
- **Use of interrupts**

Program Counter & Stack

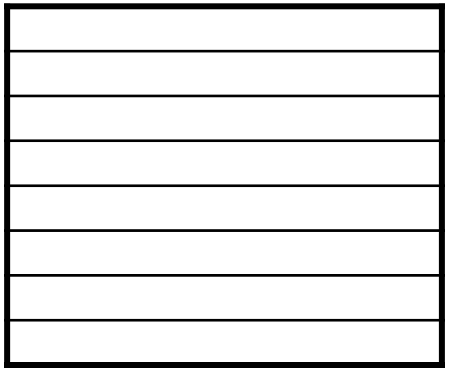
Program Memory

000h	***Reset Vector***
001h	
002h	
003h	
004h	***Interrupt Vector***
005h	
006h	goto Loop
007h	
008h	call Timer
009h	
00Ah	
00Bh	Loop addlw 06h
00Ch	
00Dh	Timer andwf PORTA,F
00Eh	
00Fh	return
010h	

Loop	equ	00Bh
Timer	equ	00Fh

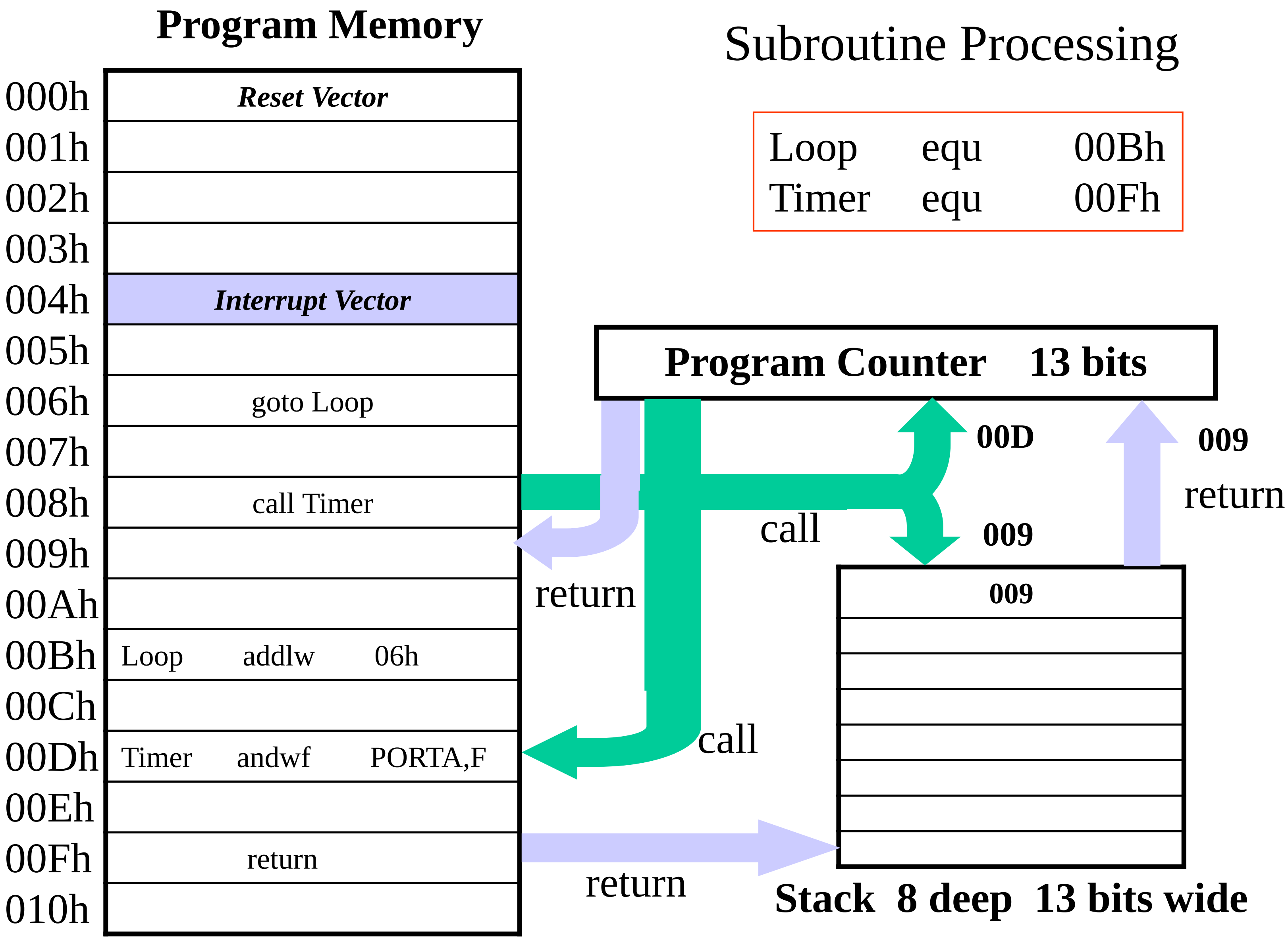
00B

Program Counter 13 bits



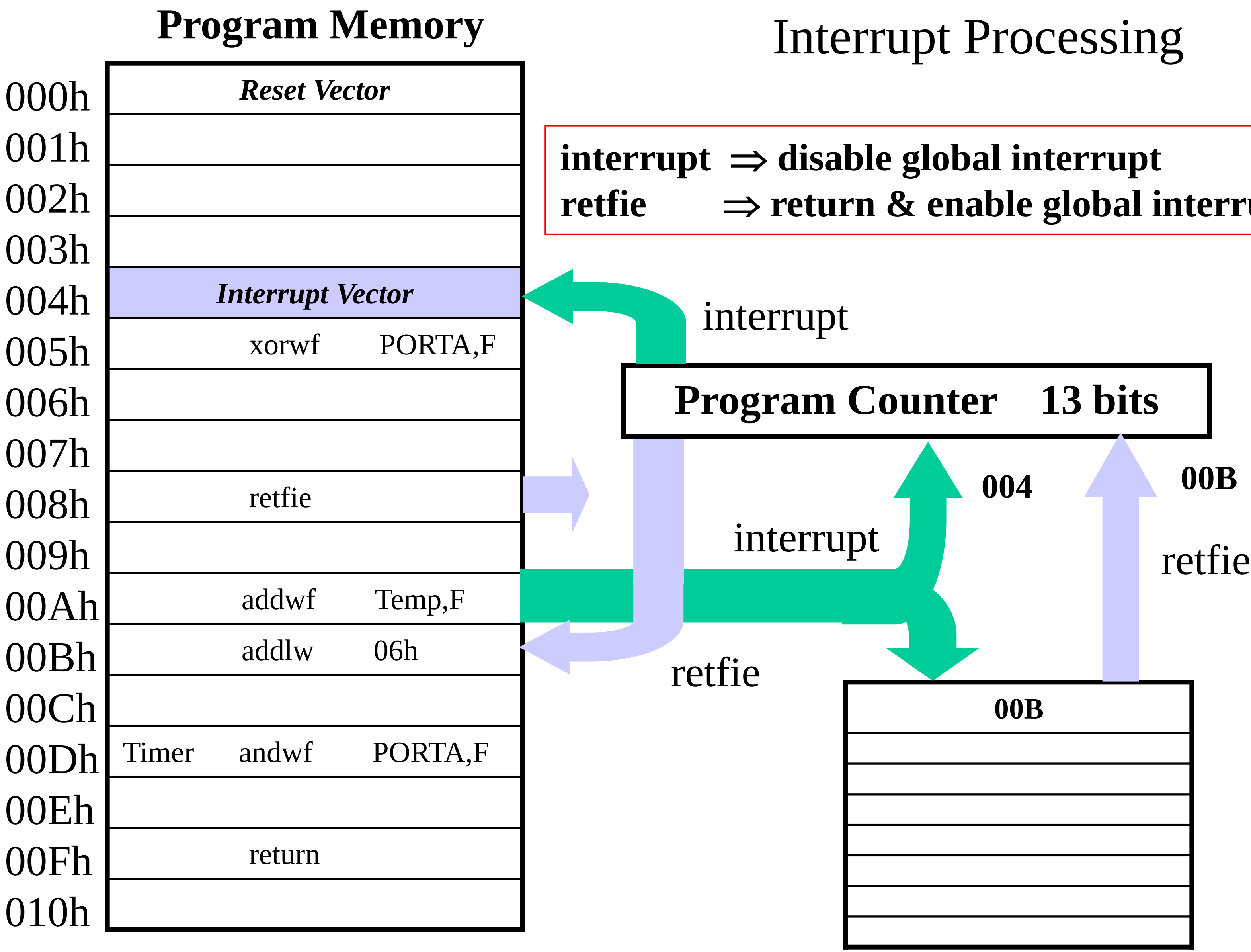
Stack 8 deep 13 bits wide

Subroutine Processing



Interrupt Processing

interrupt ⇒ disable global interrupt
retfie ⇒ return & enable global interrupt



Issues with Interupts

- There may be some code that you are executing that you do not want to be interrupted from - - you have to disable interrupts (easiest using global interrupt enable bit (GIE)).
- In Assembler, you have to do context saving (at least W register and STATUS register).
- Interrupt code starts in program memory location 004h.

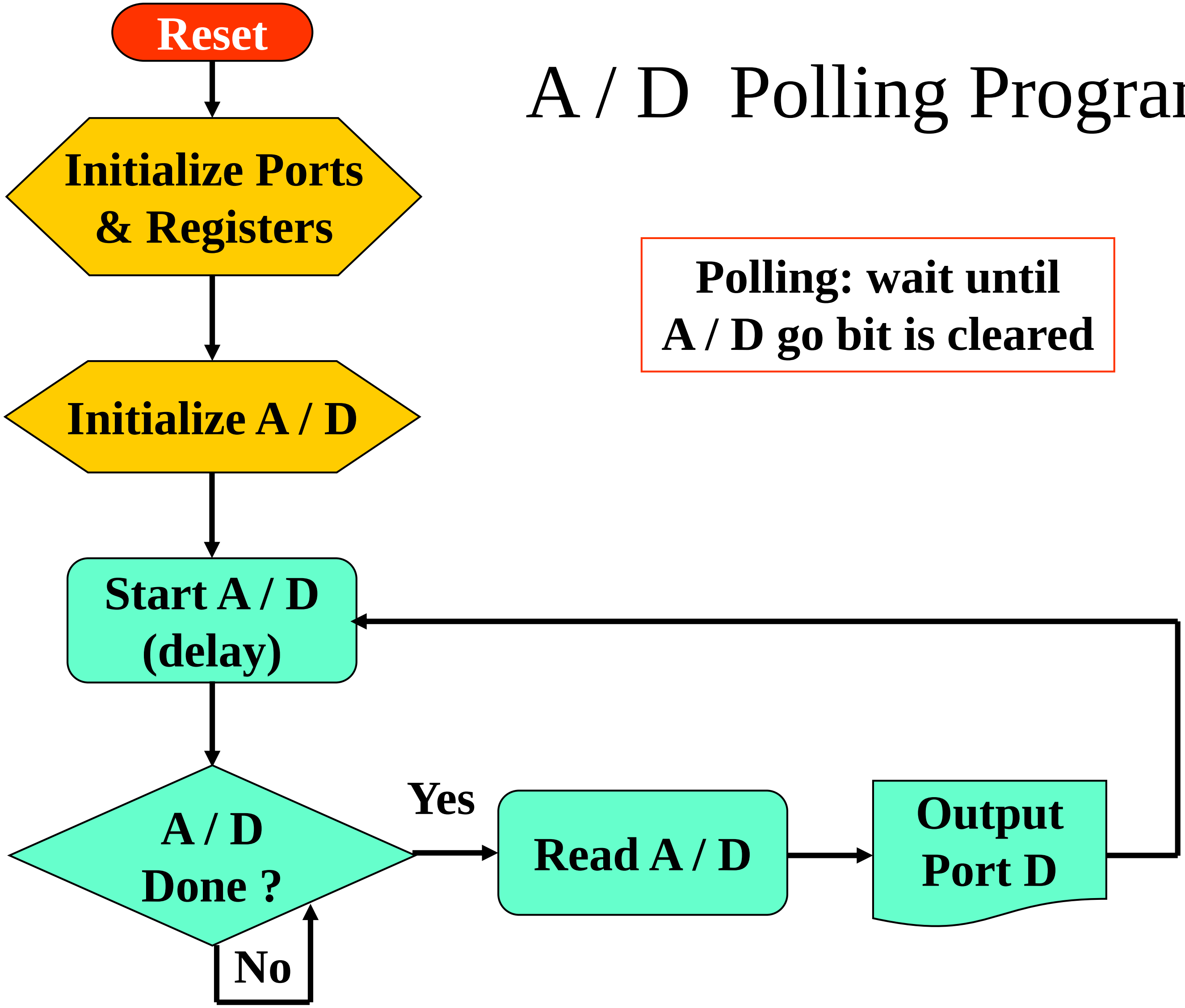
```
ORG   004h
```

- In C, the compiler takes care of context saving and putting the routine in the correct program memory location (recall that interrupt service routines must begin with "isr" for the Software Standard).

```
void   interrupt   isrProcess (void)
```

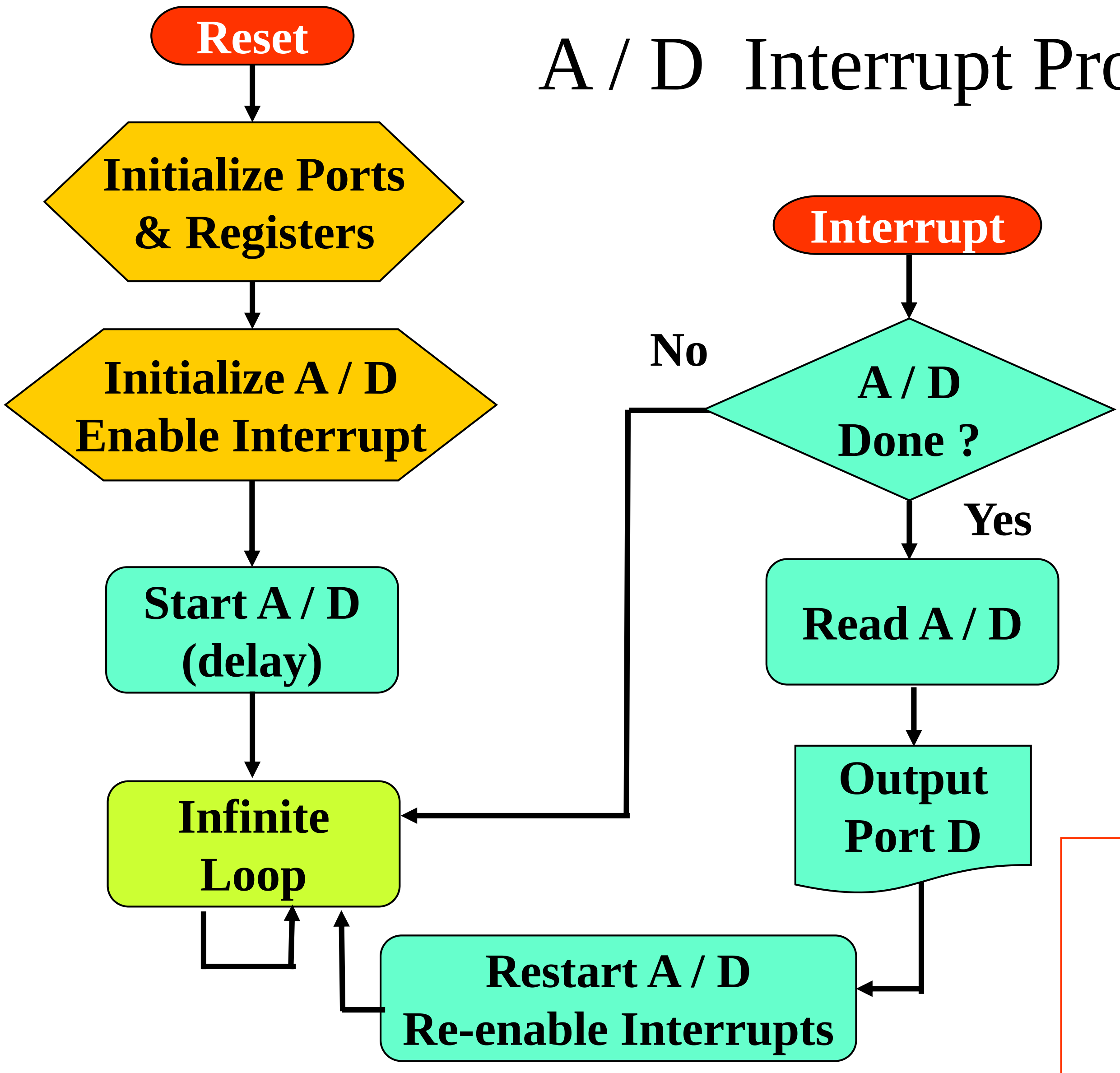
- Interrupts are the heart of real time processing.

A / D Polling Program



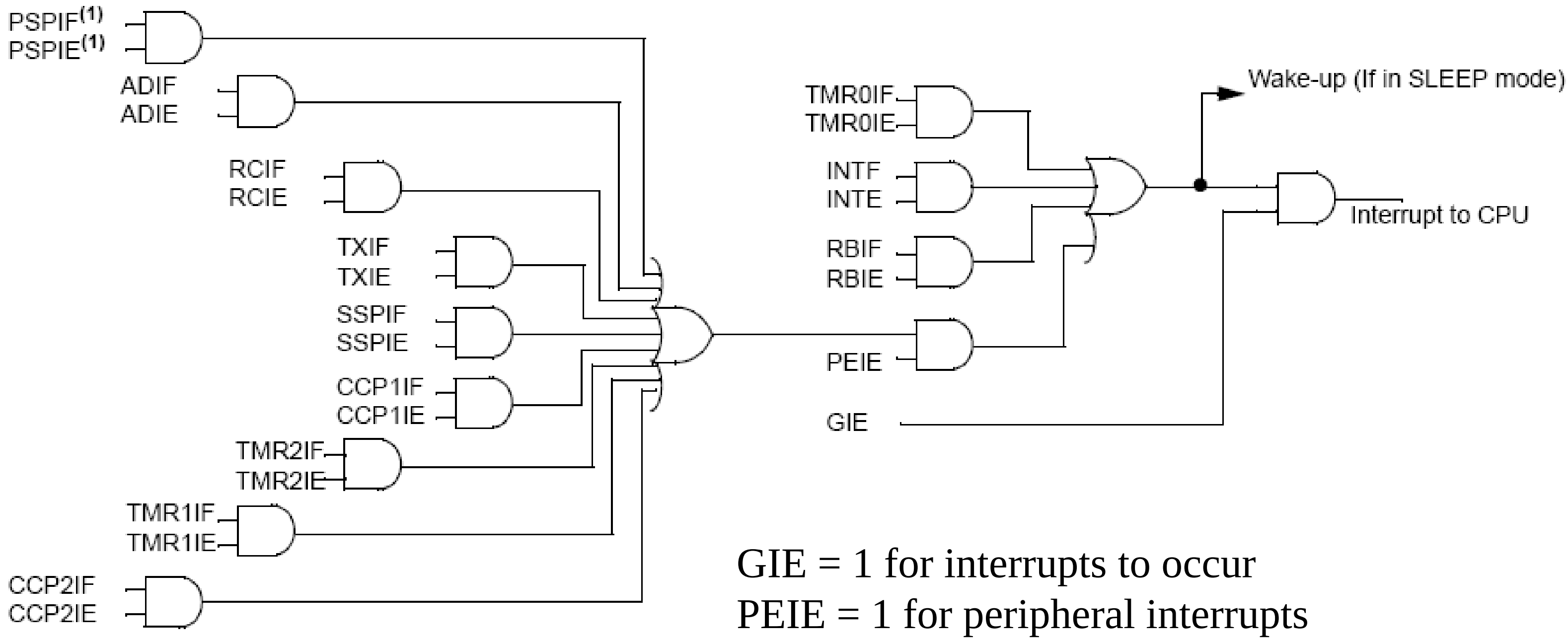
Polling: wait until A / D go bit is cleared

A / D Interrupt Program



Interrupt processor when A / D is finished

PIC16F74 Interrupt Logic



GIE = 1 for interrupts to occur
PEIE = 1 for peripheral interrupts

Flags indicate the interrupts that triggered
Enables determine which interrupts can execute

INTCON Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **TMR0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
A mismatch condition will continue to set flag bit RBIF. Reading PORTB will end the mismatch condition and allow flag bit RBIF to be cleared.
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

PIE1 Register

bit 7 **PSPIE(1):** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

PIR1 Register

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **PSPIF[(1)]:** Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion is completed (must be cleared in software)
0 = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 **SSPIF**: Synchronous Serial Port (SSP) Interrupt Flag
1 = The SSP interrupt condition has occurred, and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
SPI
A transmission/reception has taken place.
I^2C Slave
A transmission/reception has taken place.
I^2C Master
A transmission/reception has taken place.
The initiated START condition was completed by the SSP module.
The initiated STOP condition was completed by the SSP module.
The initiated Restart condition was completed by the SSP module.
The initiated Acknowledge condition was completed by the SSP module.
A START condition occurred while the SSP module was IDLE (multi-master system).
A STOP condition occurred while the SSP module was IDLE (multi-master system).
0 = No SSP interrupt condition has occurred

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

PIE2 & PIR2 Registers

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	CCP2IE
bit 7							bit 0

bit 7-1 **Unimplemented:** Read as '0'

bit 0 **CCP2IE**: CCP2 Interrupt Enable bit

1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0
—	—	—	—	—	—	—	CCP2IF
bit 7							bit 0

bit 7-1 **Unimplemented:** Read as '0'

bit 0 **CCP2IF**: CCP2 Interrupt Flag bit

Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred

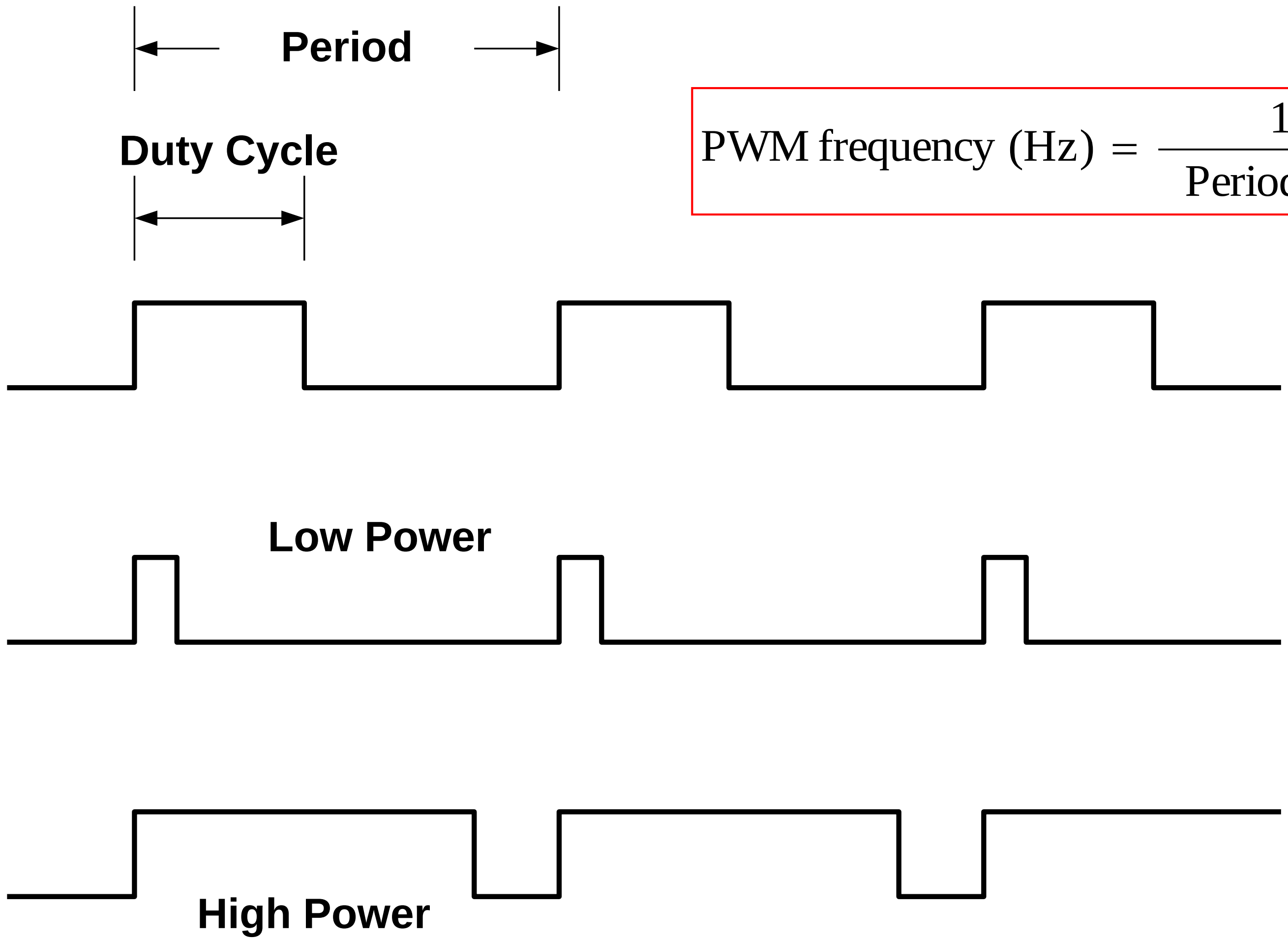
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred

PWM mode:
Unused

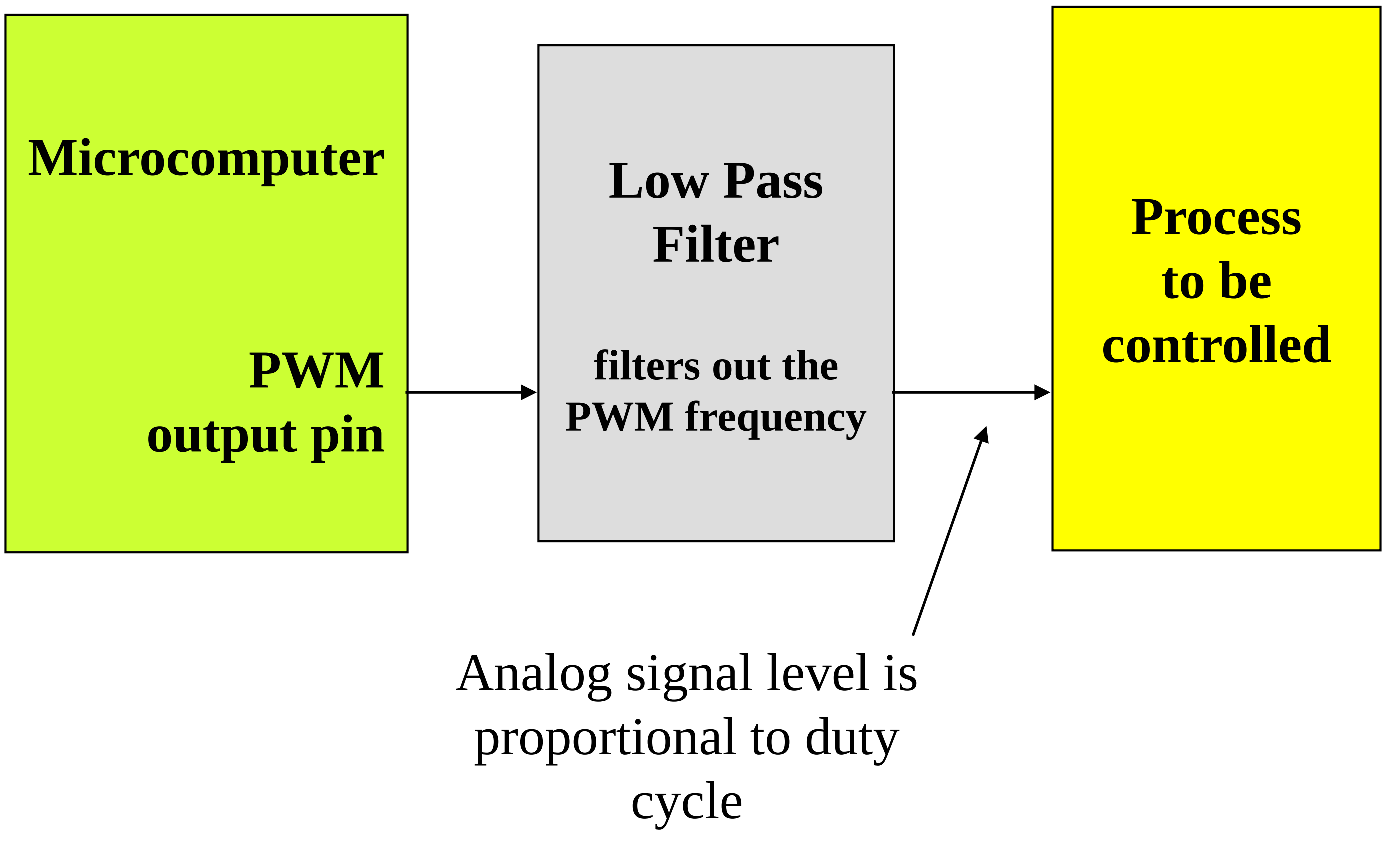
Pulse Width Modulation

Period (Frequency) fixed
Duty Cycle proportional to power needed
Frequency typically > 20 kHz so not audible

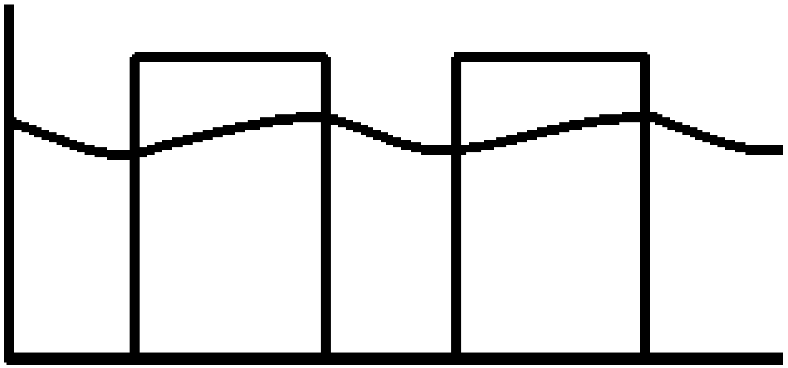
$$\text{PWM frequency (Hz)} = \frac{1}{\text{Period (sec)}}$$



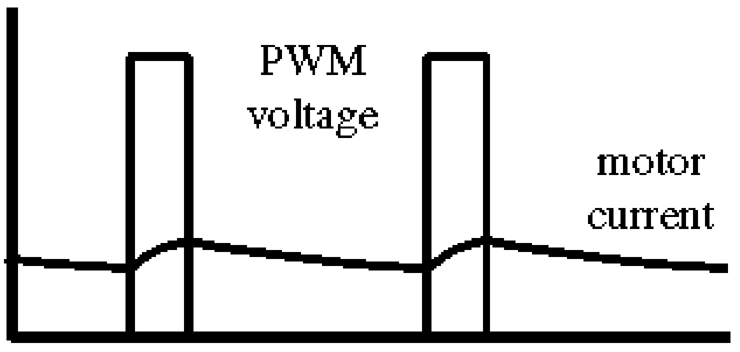
Usually requires a filter



PWM After Filtering



high duty cycle



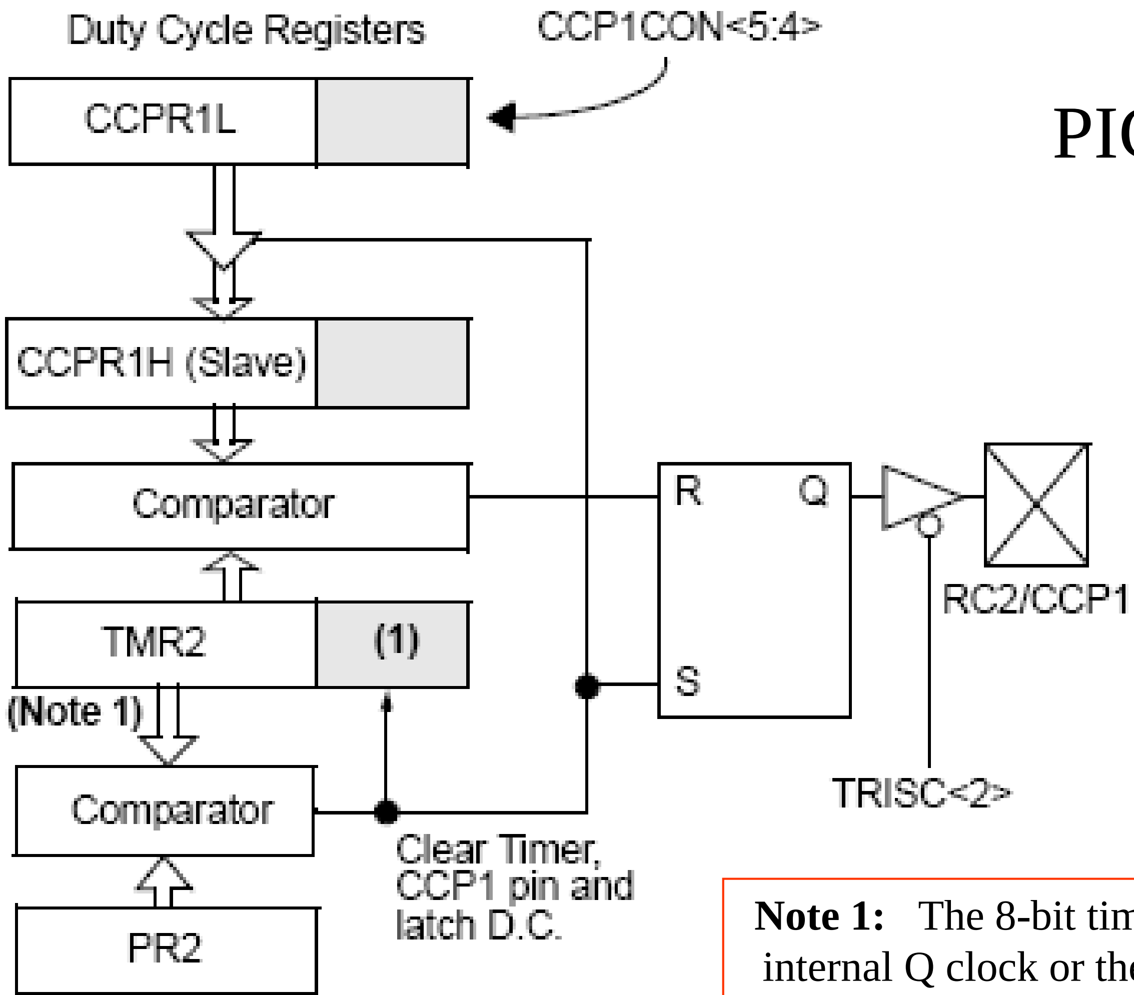
low duty cycle

Recall that a first order filter has a single time constant

Advantages of Pulse Width Modulation for Microcomputer DC Motor Control

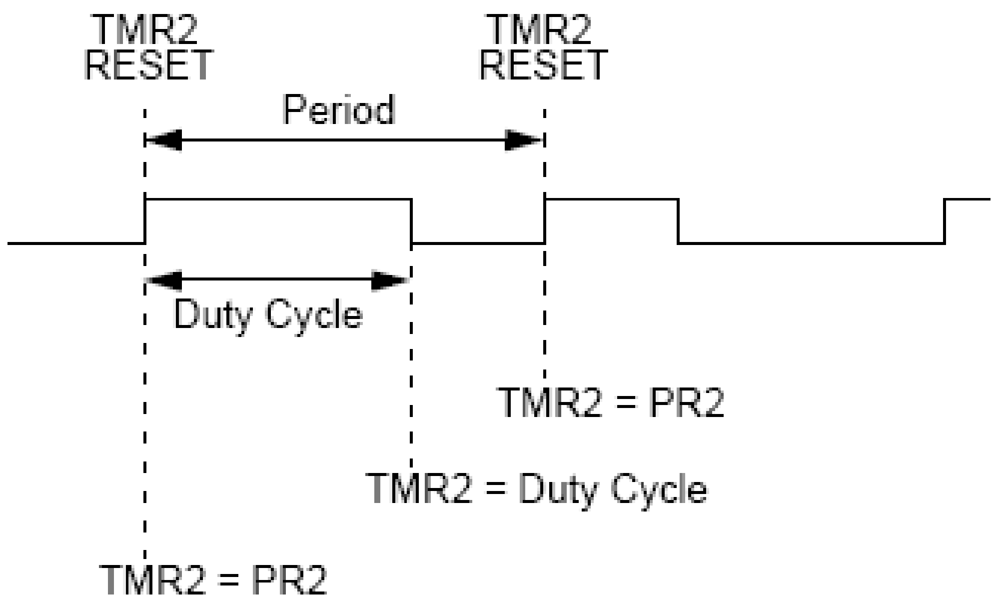
- PWM signals are generated in an independent module within the microcomputer & does not require use of the CPU.
- PWM (on – off) power signals can be very efficiently produced by power electronics (called a class D amp)
- All DC motors have inductance which naturally filters the PWM signals producing smooth currents in the motor
- Switching electronic components are lower cost & more reliable (especially for high power)

PIC16F73 PWM Module



Note 1: The 8-bit timer is concatenated with the 2-bit internal Q clock or the 2 bits of the prescaler to create the 10-bit time-base.
The internal Q clock is concatenated with TMR2 to allow greater precision of the PWM signal.

PIC16F73 PWM Timing



PWM period = [(PR2) + 1] • 4 • TOSC • (TMR2 prescale value)

PWM duty cycle = (CCPR1L:CCP1CON<5:4>) • TOSC • (TMR2 prescale value)

Note: Depending on the value in PR2, you can achieve a duty cycle longer than the period. This is an error in set-up.

CCP1CON Register

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCPxX:CCPxY**: PWM Least Significant bits

Capture mode:
Unused

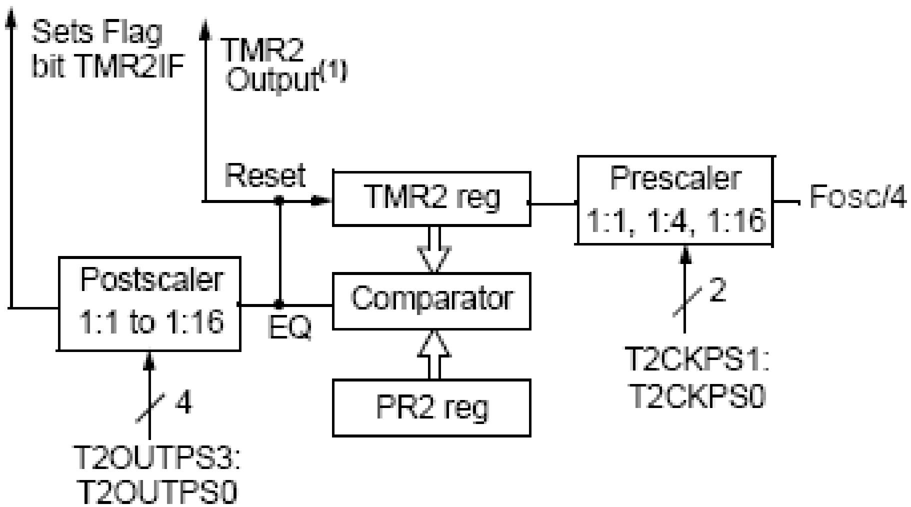
Compare mode:
Unused

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

`0000` = Capture/Compare/PWM disabled (resets CCPx module)
`0100` = Capture mode, every falling edge
`0101` = Capture mode, every rising edge
`0110` = Capture mode, every 4th rising edge
`0111` = Capture mode, every 16th rising edge
`1000` = Compare mode, set output on match (CCPxIF bit is set)
`1001` = Compare mode, clear output on match (CCPxIF bit is set)
`1010` = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
`1011` = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 clears Timer1; CCP2 clears Timer1 and starts an A/D conversion (if A/D module is enabled)
`11xx` = PWM mode

Timer 2 Operation



T2CON Register

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

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

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0001 = 1:2 Postscale
0010 = 1:3 Postscale
•
•
•
1111 = 1:16 Postscale

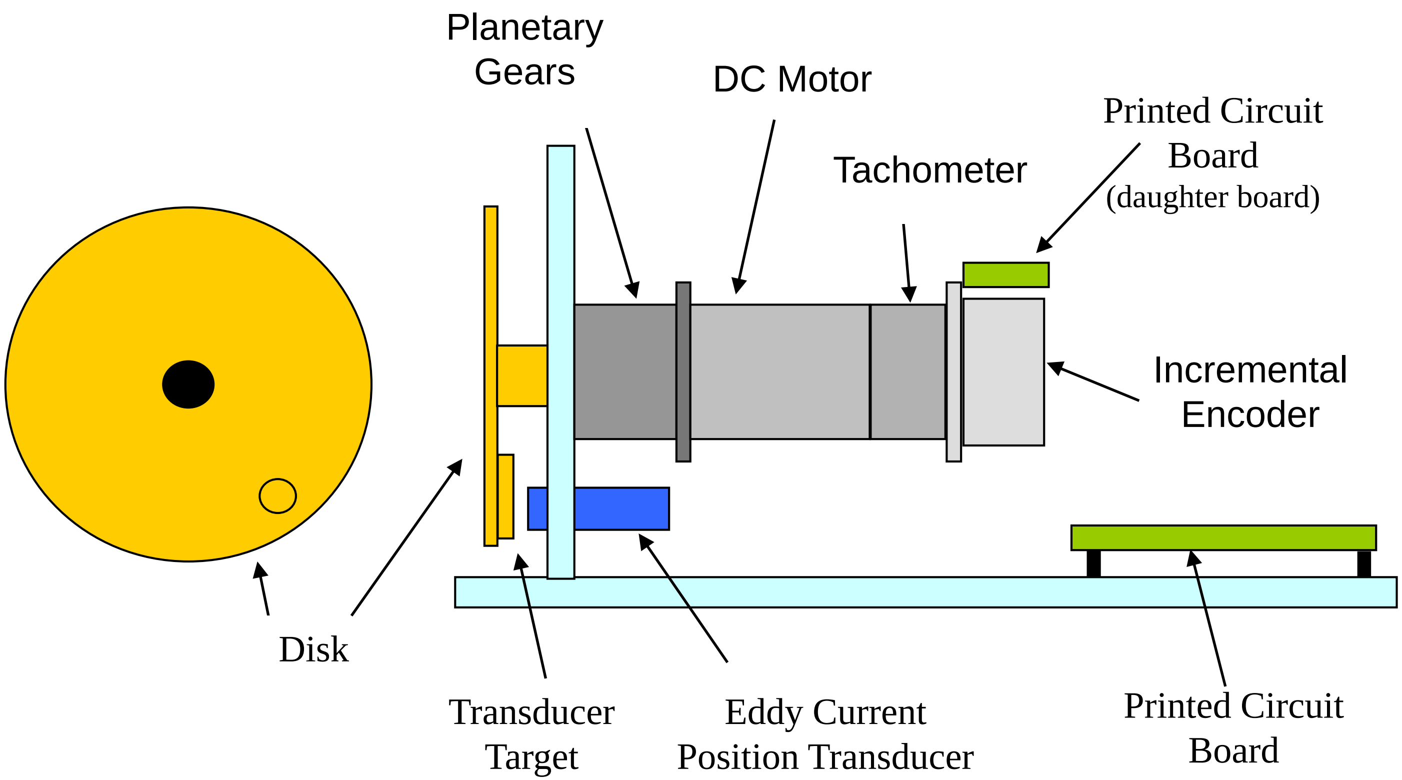
bit 2 **TMR2ON**: Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

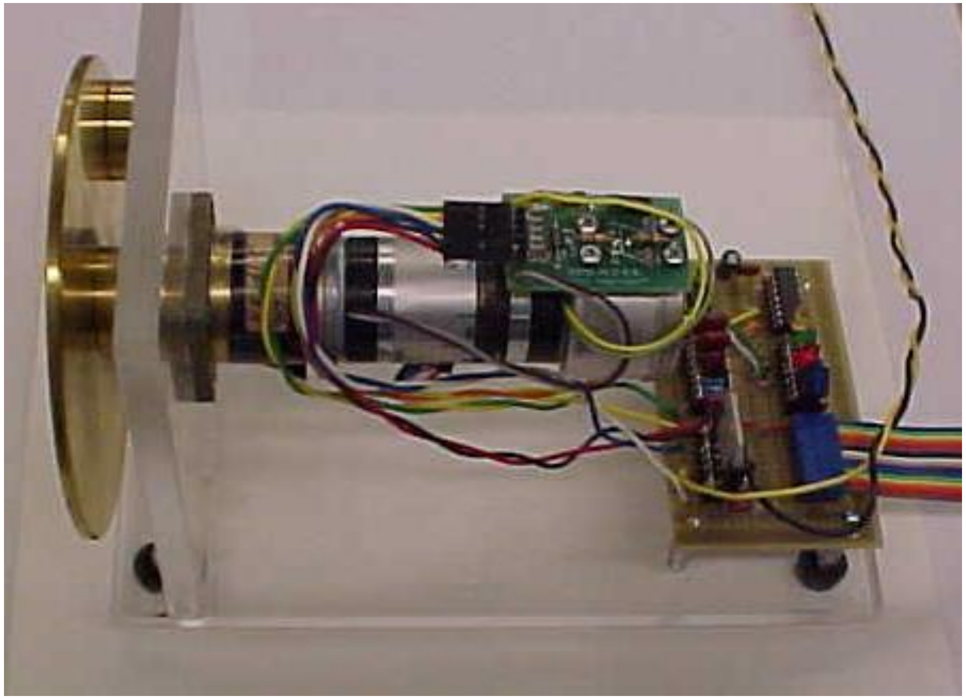
bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

DC Motor Case Study Schematic



Index the Motor



Use sensor on output disk to position motor prior to starting.

DC Motor Case Study

- Index motor using a sensor on the load

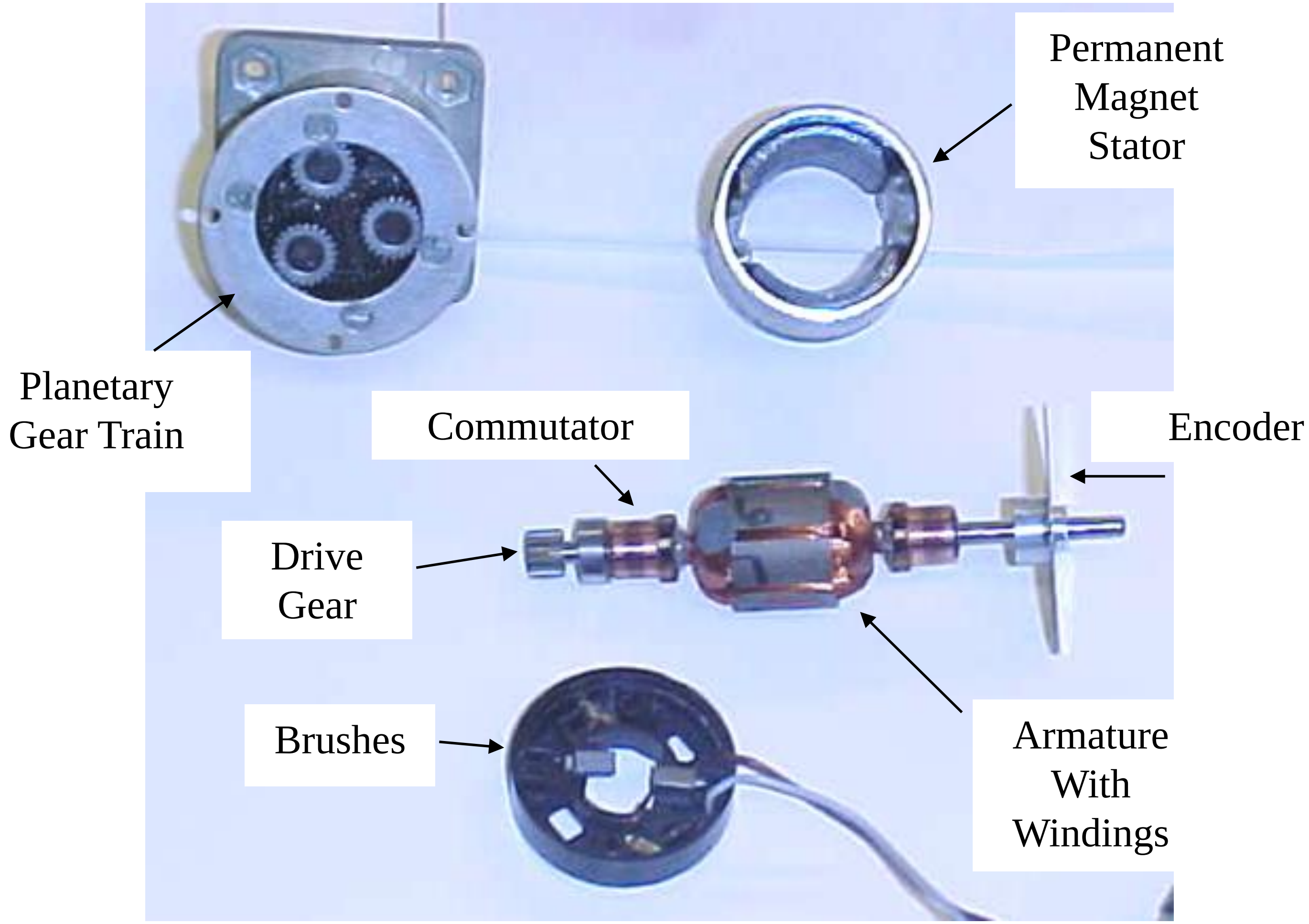
- Microcomputer will perform 2 functions
  - Compute velocity reference to follow a trapezoidal trajectory
  - Measure the actual motor velocity & control motor to track reference

- Different Modes
  - Mode 0 : Proportional Compensator
  - Mode 1 : Proportional Compensator + Bias
  - Mode 2 : Proportional – Derivative (P D) Compensator
  - Mode 3 : Proportional – Integral – Derivative (P I D) Compensator

Example: Consider a CNC Machine

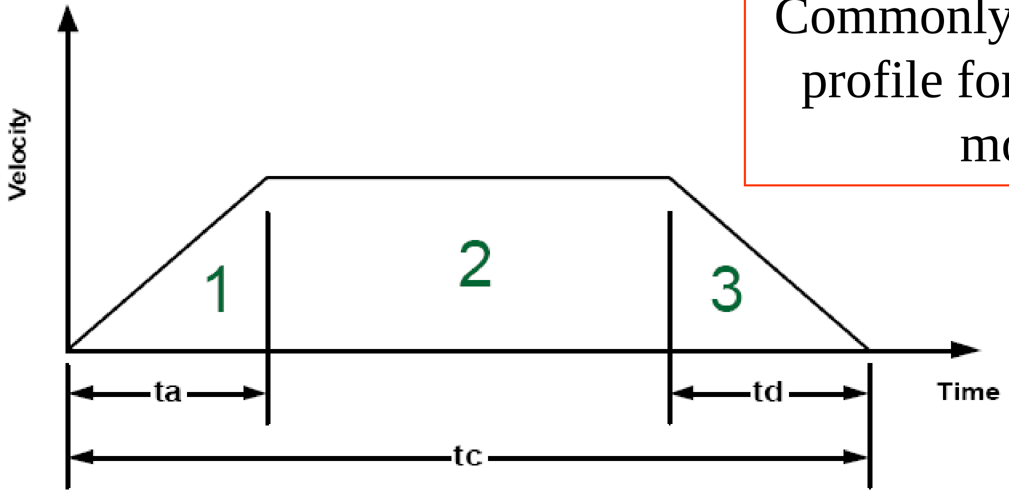
- System controller will tell the table to go to a certain position relative to an index
- Motion controller will:
    1. Move the table & find the index mark
    2. Compute the velocity profile required for the move
    3. Control the movement of the table by measuring its actual velocity, comparing it to the computed velocity profile & adjusting the power in a DC motor to have the actual velocity track the reference velocity

Motor Components



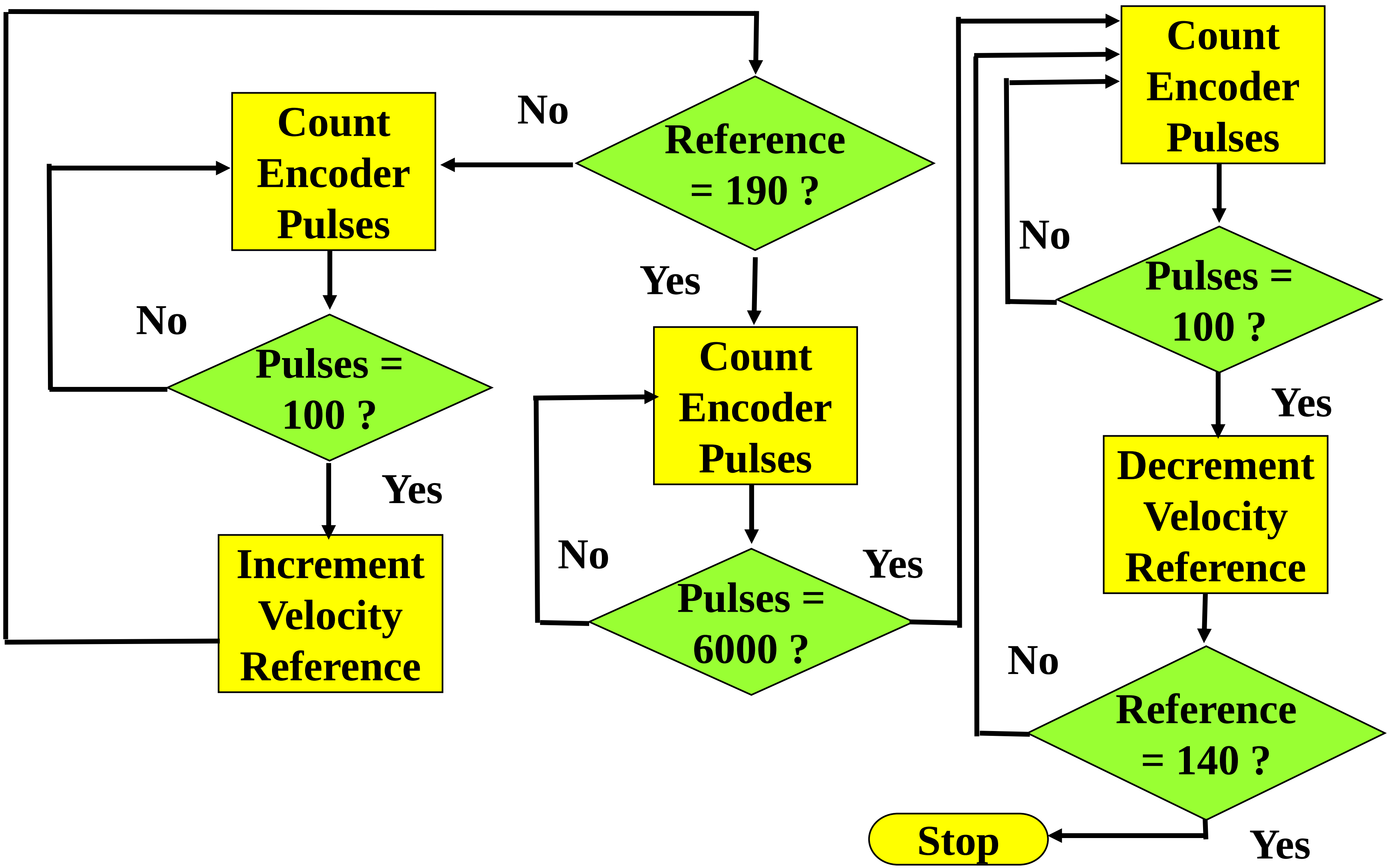
Trapezoidal Velocity Profile

1. Inertia Limited (acceleration)
2. Viscous Friction Limited (constant speed)
3. Inertia Limited (deceleration)

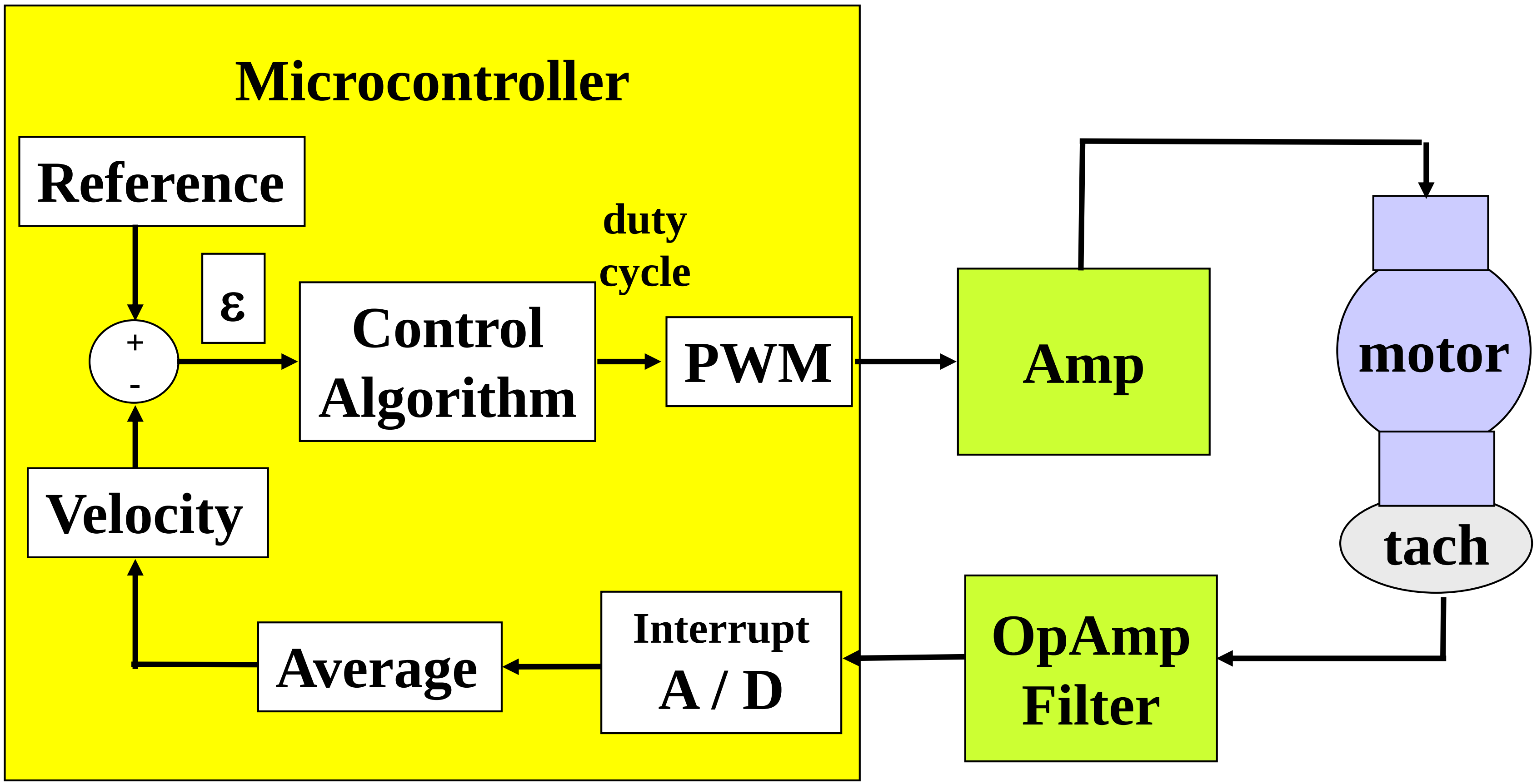


Commonly used velocity profile for incremental motion.

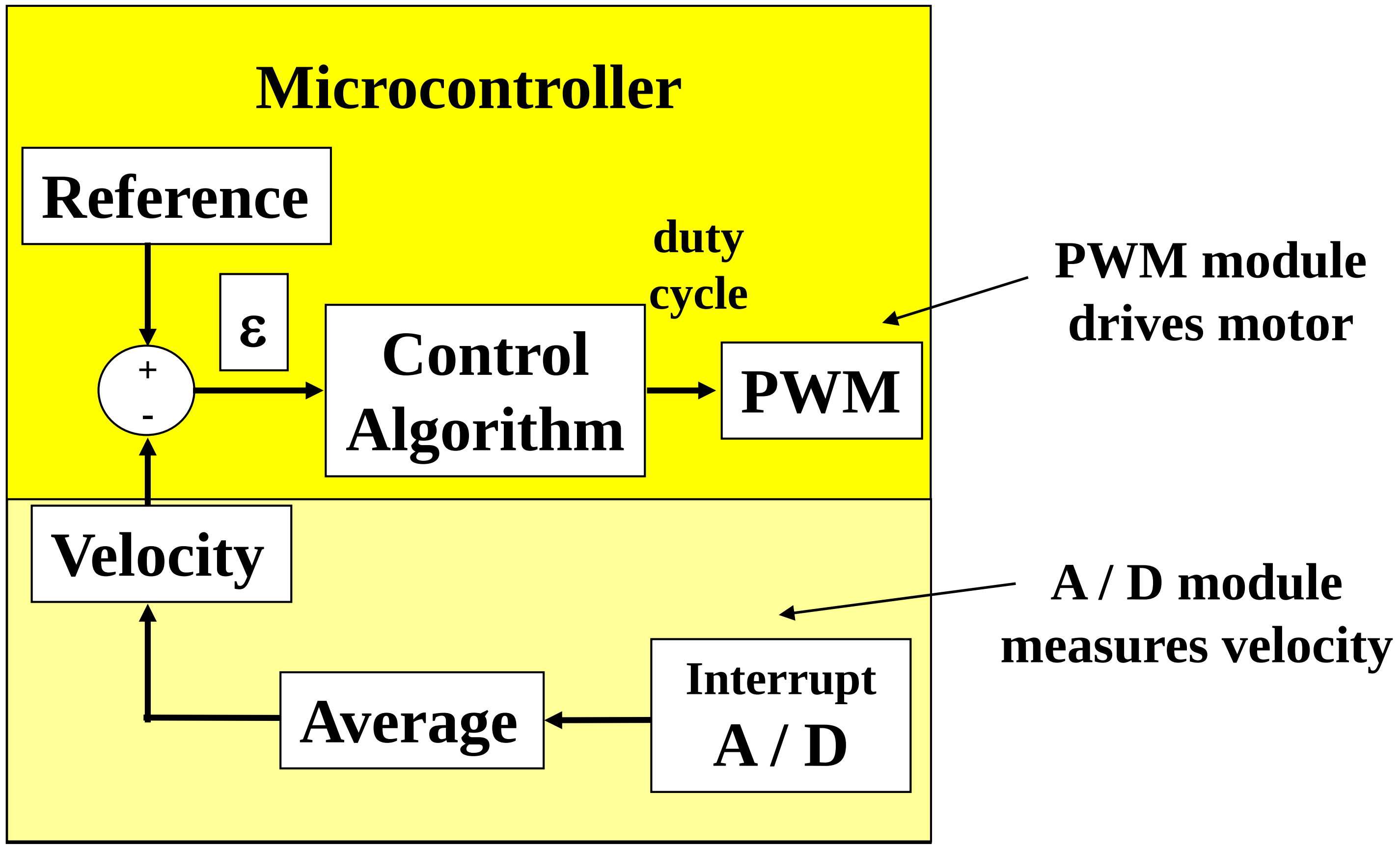
Microcomputer will generate profile



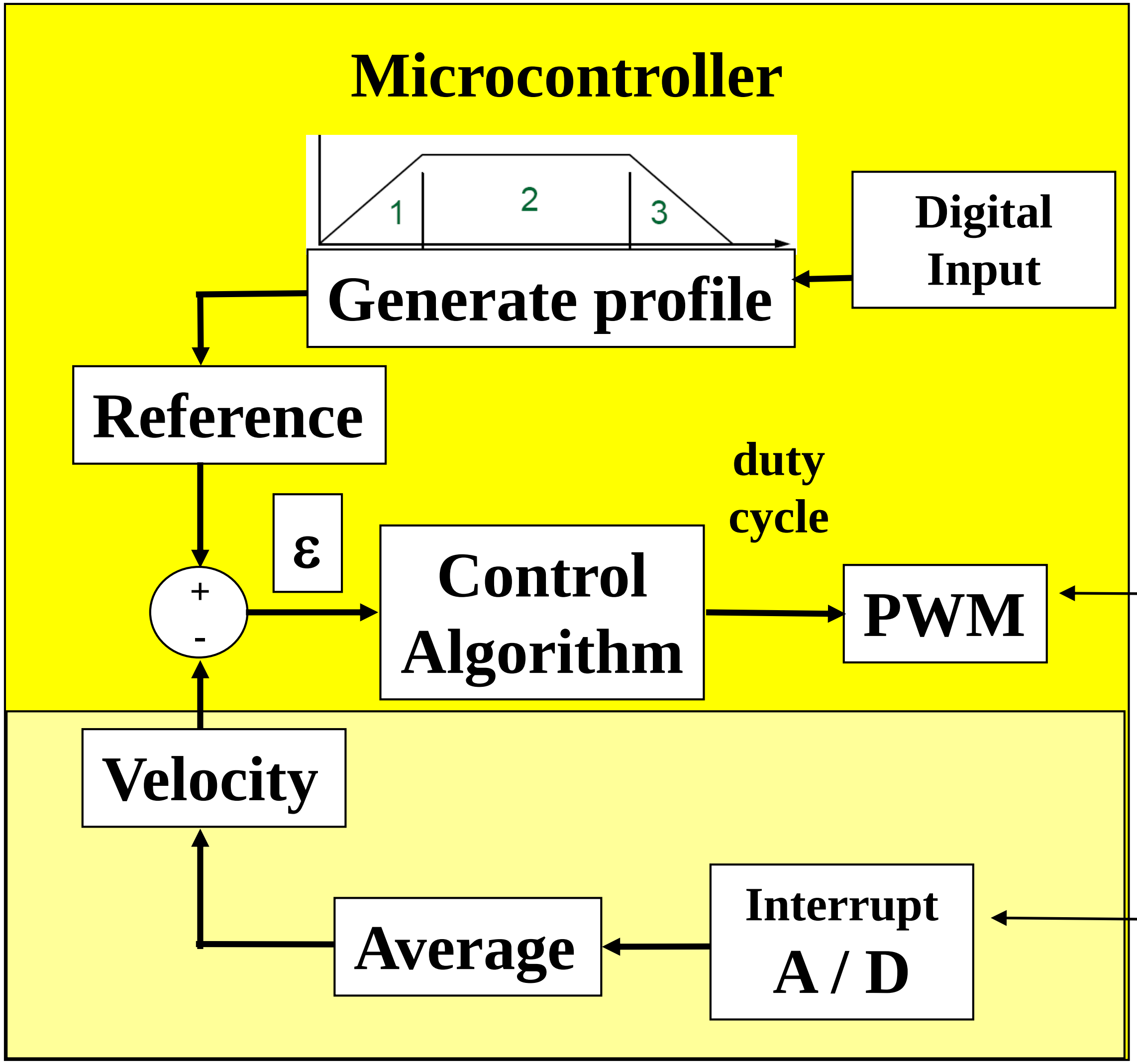
Microcomputer will also control velocity



Actual Motor Velocity from Analog Tachometer is Measured with A/D Converter & averaged to remove noise



Microcomputer's 3 Functions

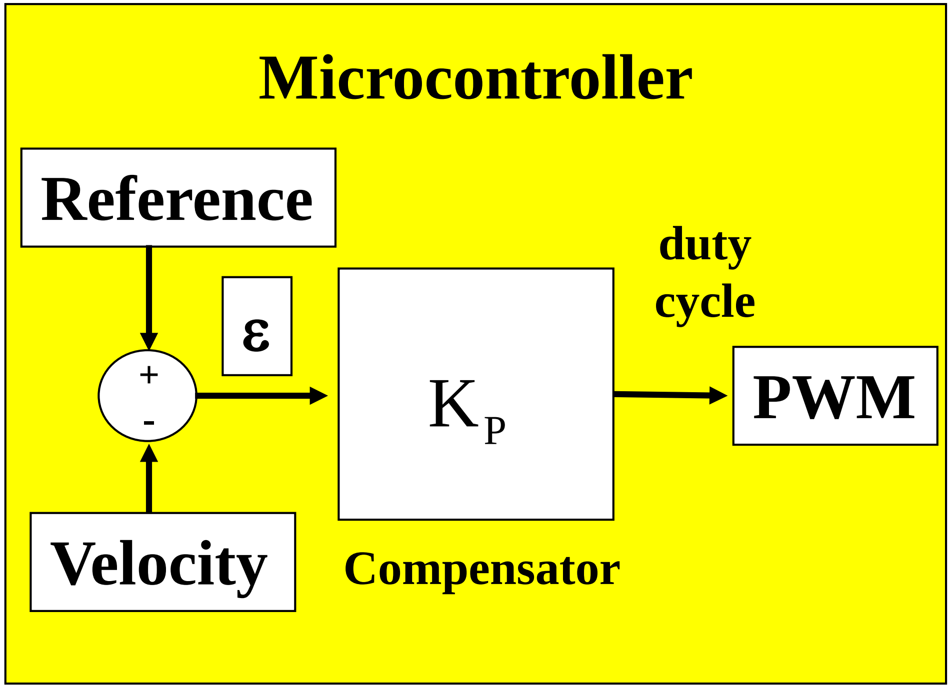


Digital Input measures angle from encoder

PWM module drives motor

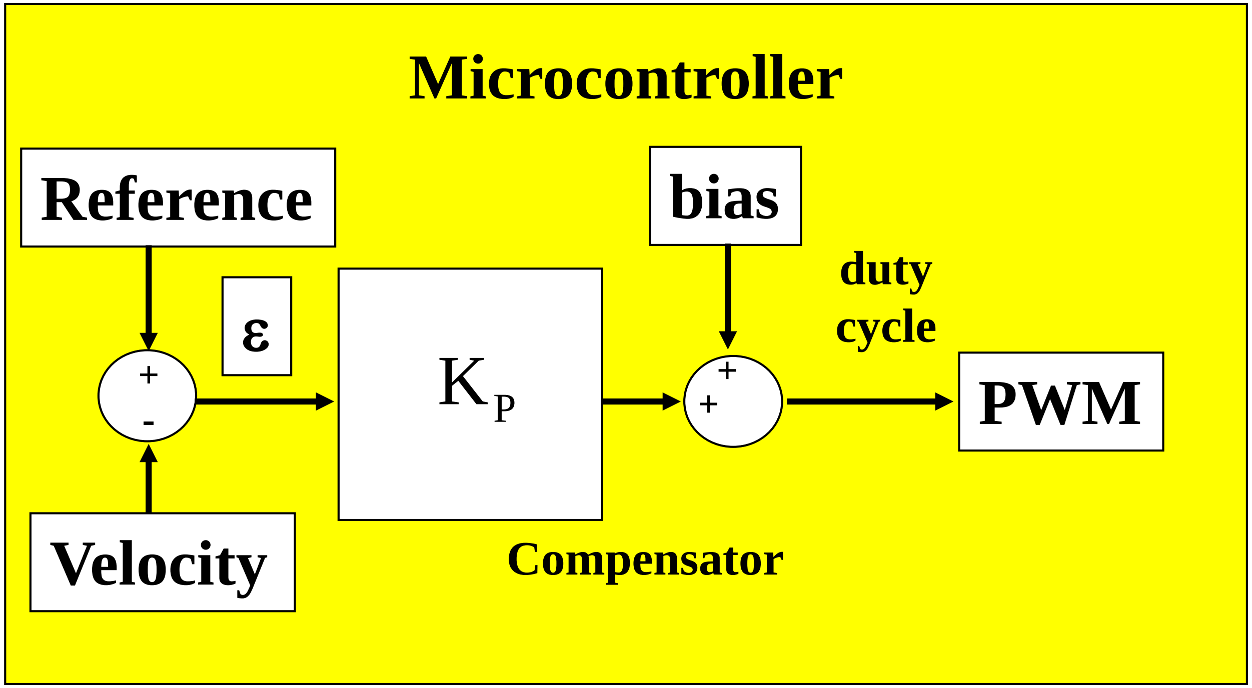
A / D module measures velocity from tachometer

Mode 0: Proportional Control



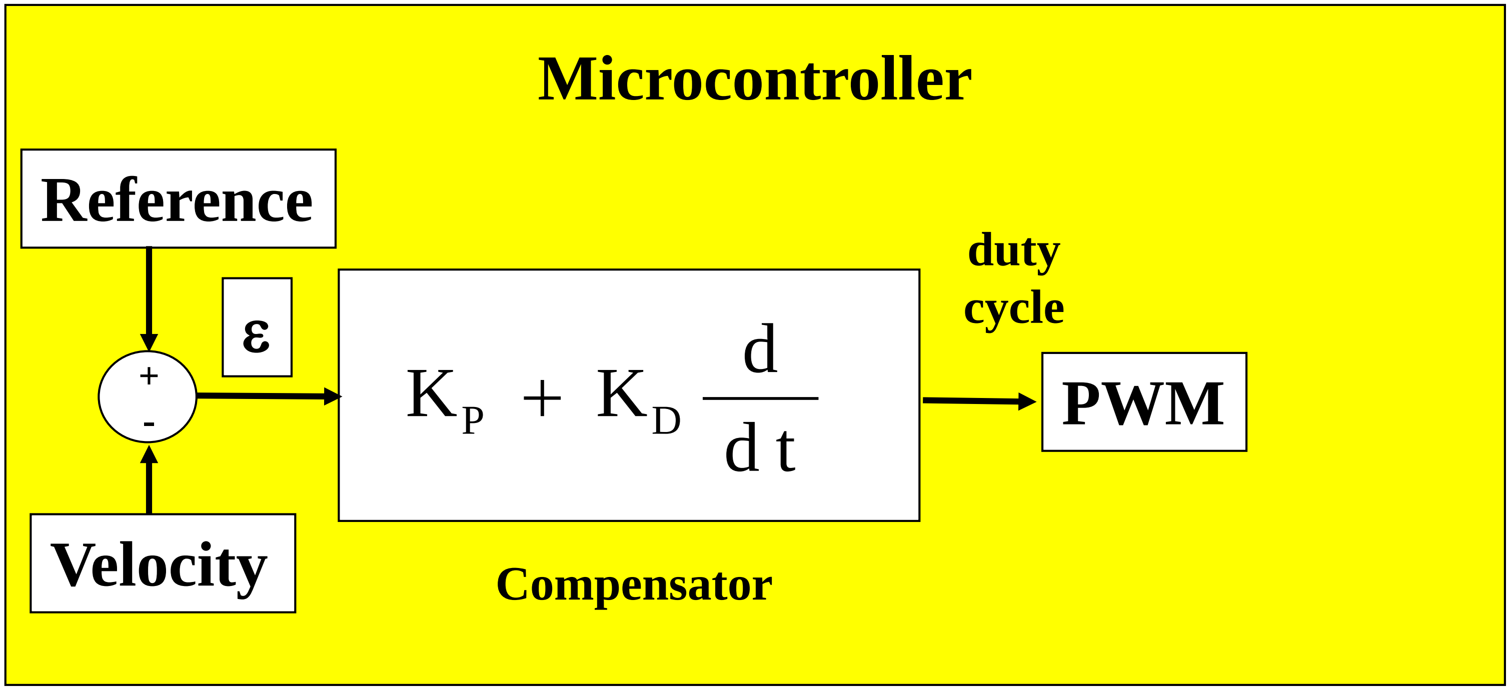
Multiply the error by a proportionality constant K_p
Make the duty cycle equal to the result

Mode 1: Proportional Control + Bias



Multiply the error by a proportionality constant K_p
Add a fixed bias (constant) to the result
Make the duty cycle equal to the final result

Mode 2: Proportional + Derivative (P D) Control

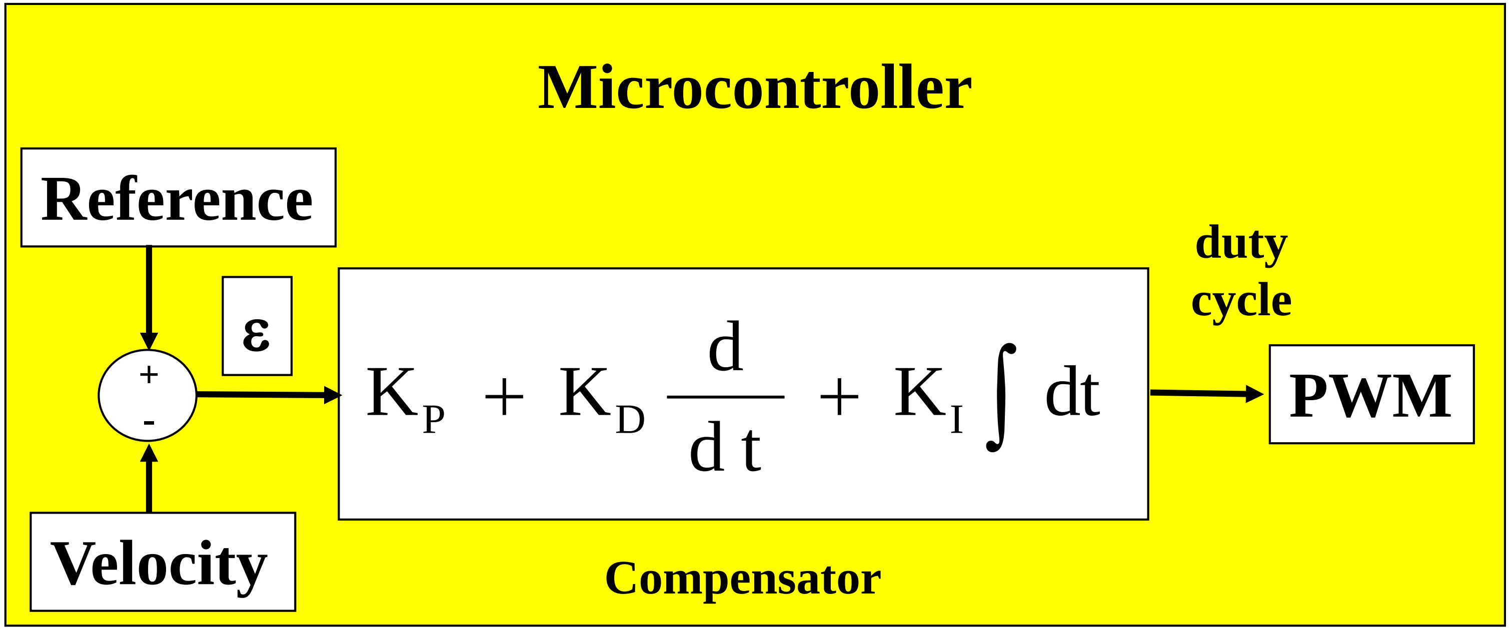


Multiply the error by a constant K_p

Multiply the derivative of the error by a constant K_d

Make the duty cycle equal to the sum

Mode 3: Proportional + Integral + Derivative (P I D) Control

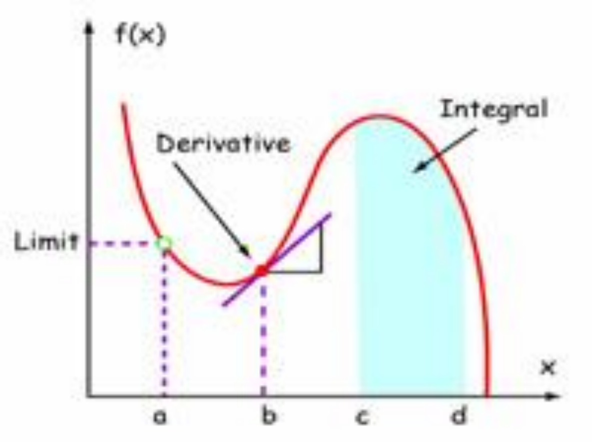


Multiply the error by a constant K_p

Multiply the derivative of the error by a constant K_D

Multiply the integral of the error by a constant K_I

Make the duty cycle equal to the sum

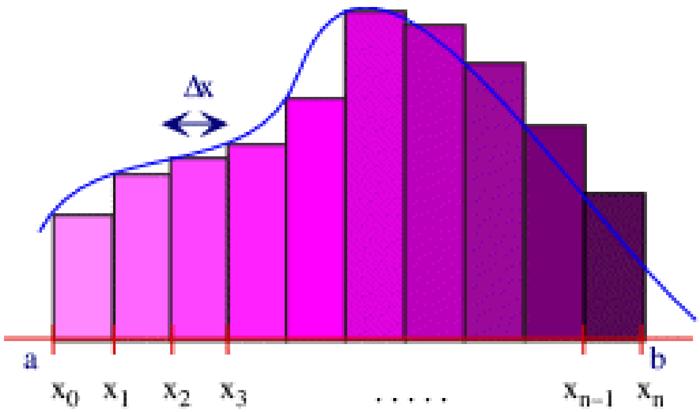


Finite Difference

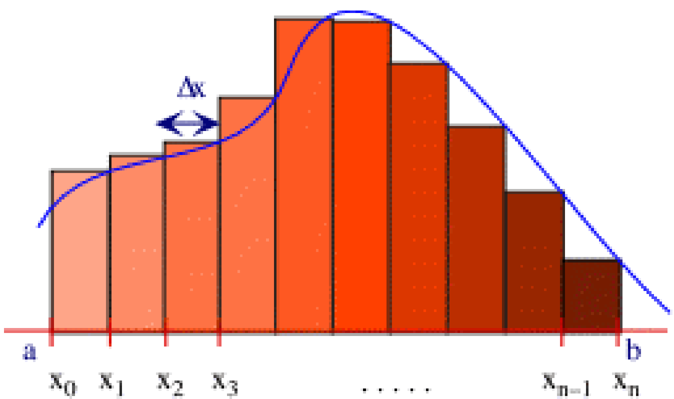
$$\frac{d\,f(t)}{d\,t} = \frac{f\,(t_{i+1}) \;-\; f\,(t_i)}{\Delta\,t}$$

$$K_D\,\frac{d\,f(t)}{d\,t} = \underbrace{\frac{K_D}{\Delta\,t}}_{\text{you would precompute this}}\left[\,f\,(t_{i+1}) \;-\; f\,(t_i)\,\right]$$

Riemann Integral



Sum as an approximation to integration



$$\int f(t)\,dt = \sum f(t_i)\,\Delta t$$

$$K_I \int f(t)\,dt = \underbrace{\left(K_I\,\Delta t\right)}_{\text{you would precompute this}} \sum f(t_i)$$

Difference Equation

PID Compensator:

$$\varepsilon_n = Vref_n - Vm_n$$

$$PWM_n = K_p\,\varepsilon_n + K_d\,(\varepsilon_n - \varepsilon_{n-1}) + K_i\,\Big(\underbrace{\textstyle\sum_{j=0}^{n} \varepsilon_j}_{\boxed{\varepsilon_n + \sum_{j=0}^{n-1} \varepsilon_j}}\Big)$$

$$PWM_{n-1} = K_p\,\varepsilon_{n-1} + K_d\,(\varepsilon_{n-1} - \varepsilon_{n-2}) + K_i\,\Big(\textstyle\sum_{j=0}^{n-1} \varepsilon_j\Big)$$

$$K_i\,\Big(\textstyle\sum_{j=0}^{n-1} \varepsilon_j\Big) = PWM_{n-1} - K_p\,\varepsilon_{n-1} - K_d\,(\varepsilon_{n-1} - \varepsilon_{n-2})$$

$$PWM_n = PWM_{n-1} + (K_p + K_d + K_i)\,\varepsilon_n + (-2K_d - K_p)\,\varepsilon_{n-1} + K_d\,\varepsilon_{n-2}$$

MAC Instruction

- Array of errors $\{\varepsilon_{n}, \varepsilon_{n+1}, \varepsilon_{n+2}, \varepsilon_{n+3}\}$
- Array of constants $\{K_1, K_2, K_3, K_4\}$

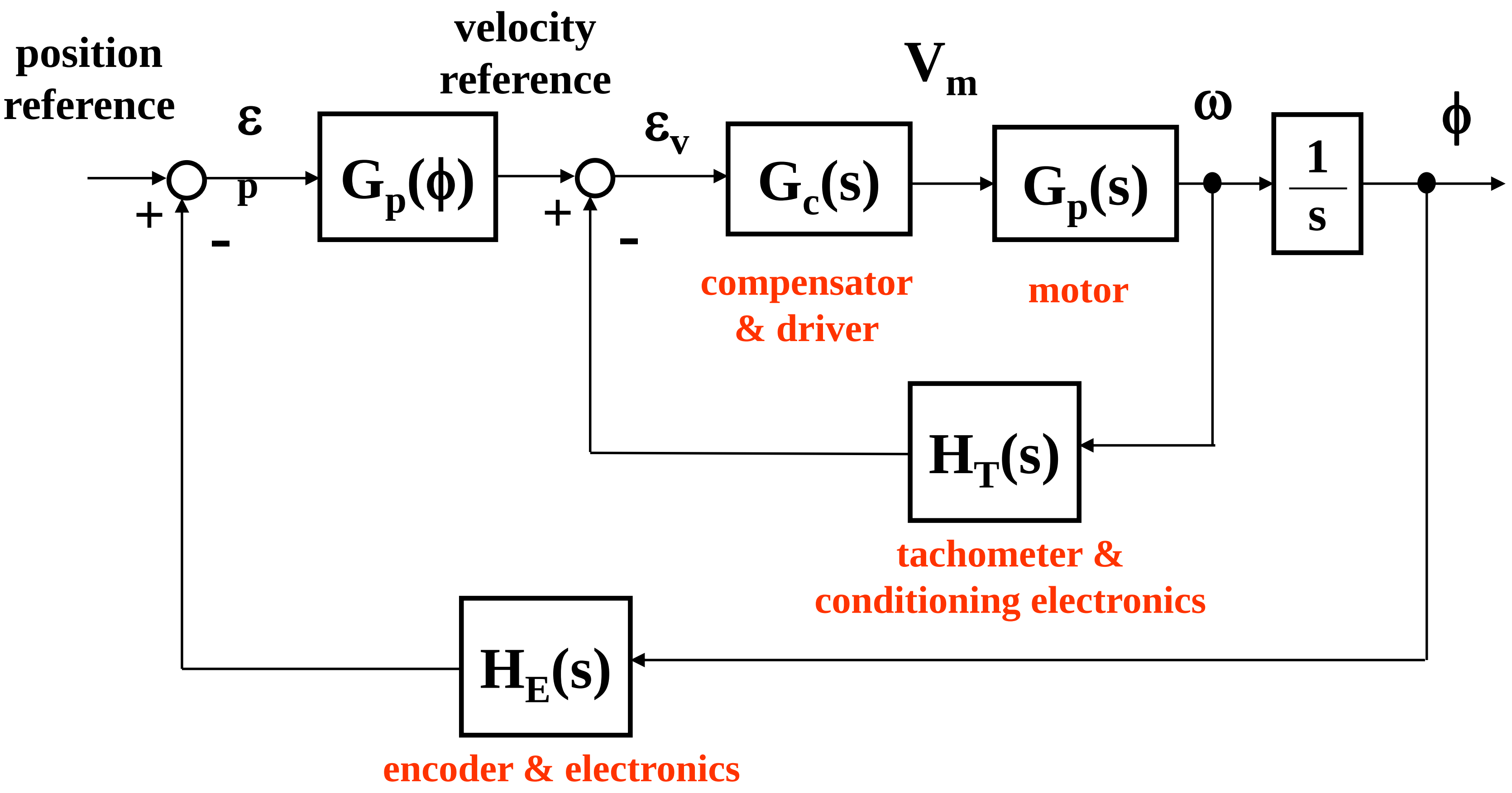
- In ONE instruction, computes

$$u = K_1 \varepsilon_n + K_2 \varepsilon_{n+1} + K_3 \varepsilon_{n+2} + K_4 \varepsilon_{n+3}$$

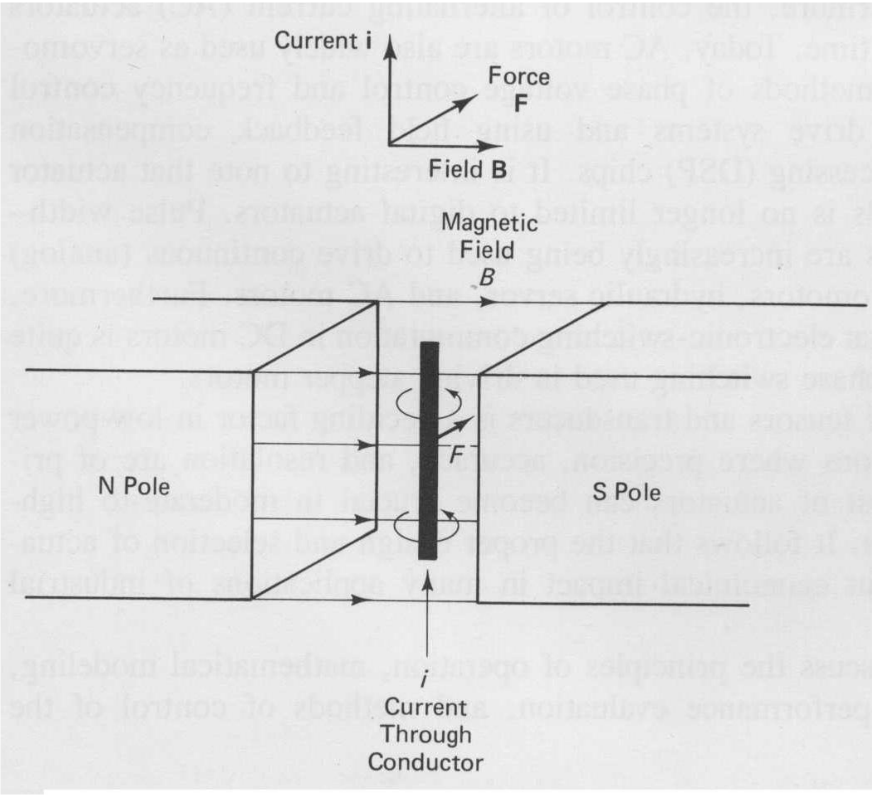
Advantages of Embedded Digital Control

- **Less susceptible to noise and parameter variation.**
- **Very high accuracy and high speed.**
- **Handles repetitive tasks extremely well.**
- **Complex control laws and signal processing methods can be implemented.**
- **High reliability.**
- **Long term, low drift, easy retrieval storage for large amounts of data.**
- **Fast data transmission possible.**
- **Low operational voltages; low power.**
- **Low overall cost.**

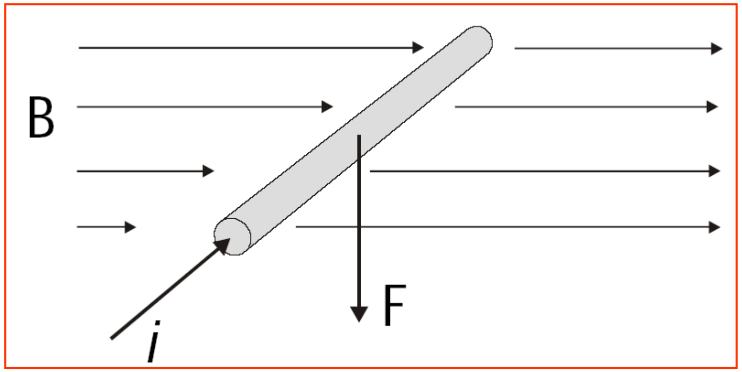
Block Diagram for DC Motor Control System



Operating Principle of a D.C. Motor



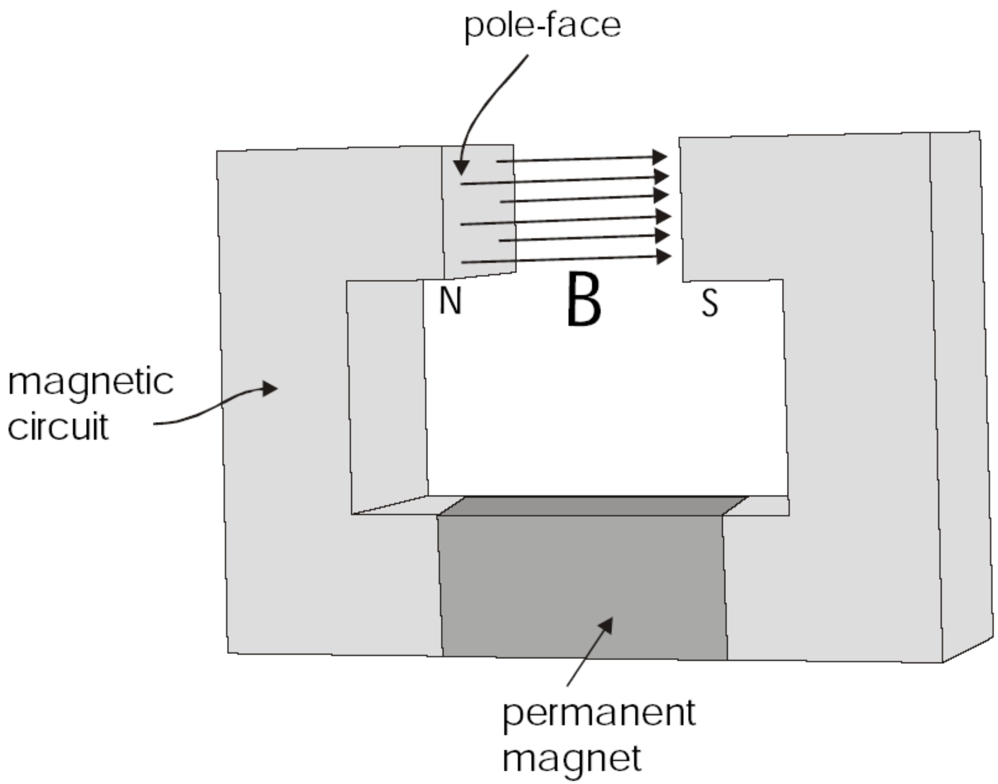
Current carrying wire of length L in a magnetic field B



If B & i are perpendicular

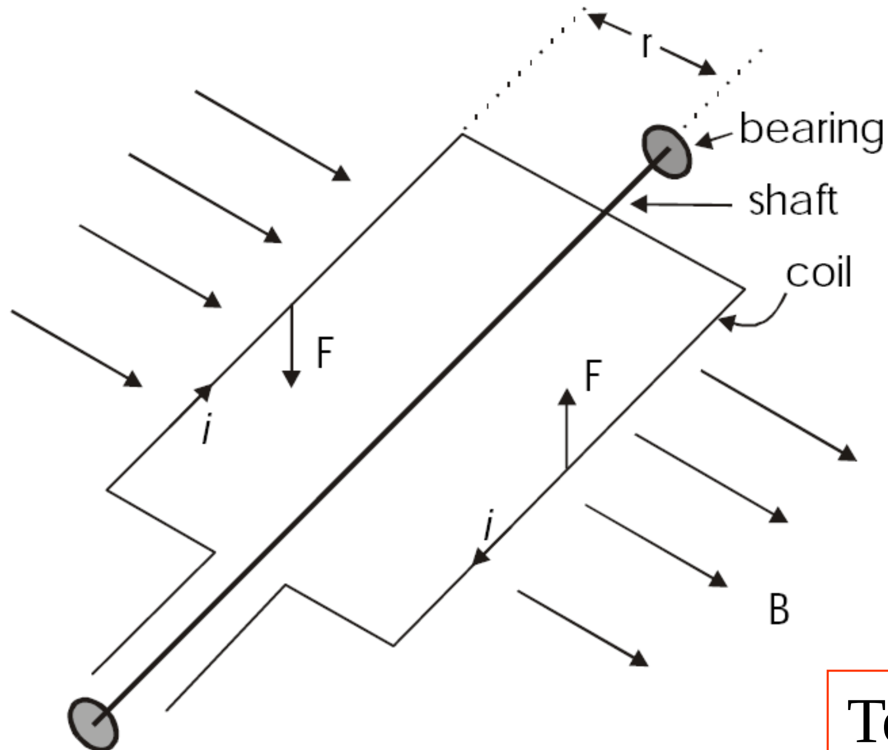
$$F = B\ L\ i$$

Brush Type Permanent Magnet Motors



The magnetic field is generated by a permanent magnet in the stationary part (the stator). The magnetic circuit is a “soft” magnetic material (iron).

Producing Torque



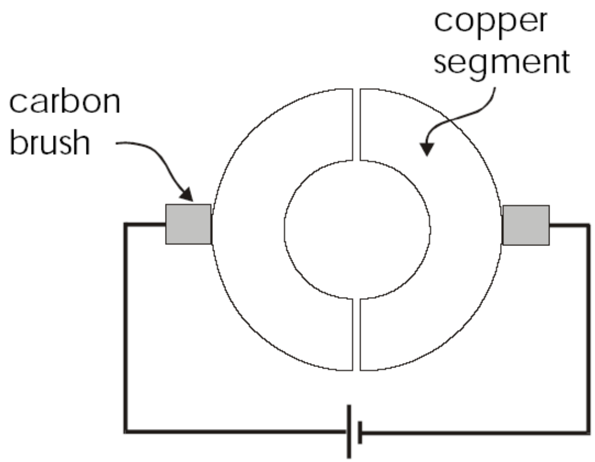
To produce torque, the current carrying wire is wound in a coil. This is called the armature or rotor.

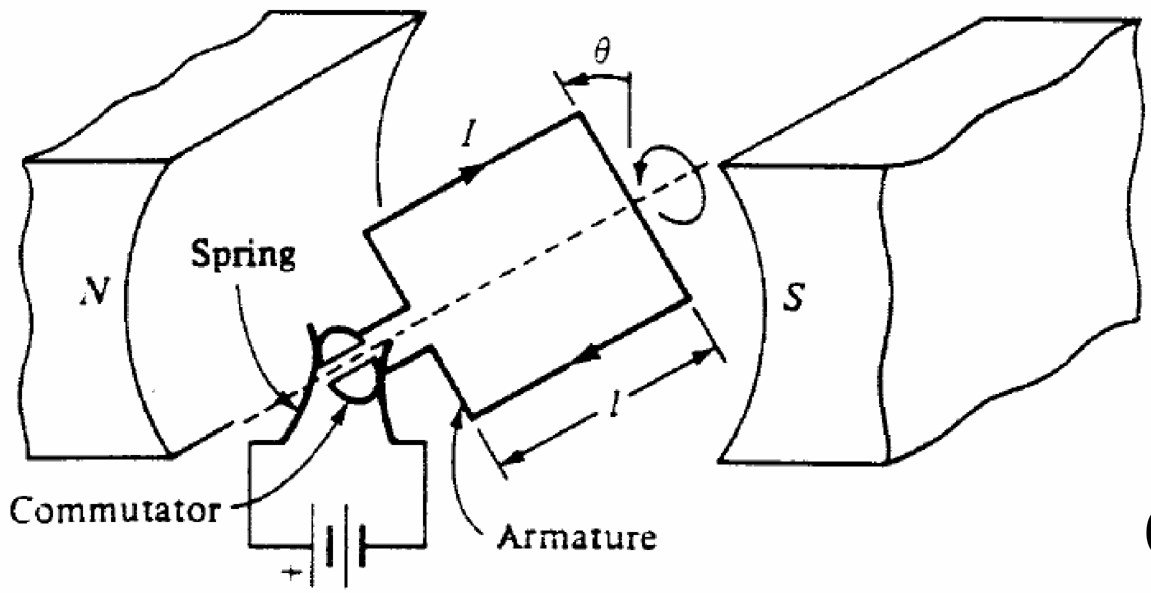
Torque $T = 2\,F\,r$

$= 2\,B\,L\,r\,i$

$= K_T\; i$

A commutator is used to keep the wire segments on the armature perpendicular to the magnetic field.
If the armature makes a 180^0 rotation, the current in the same coil is reversed.

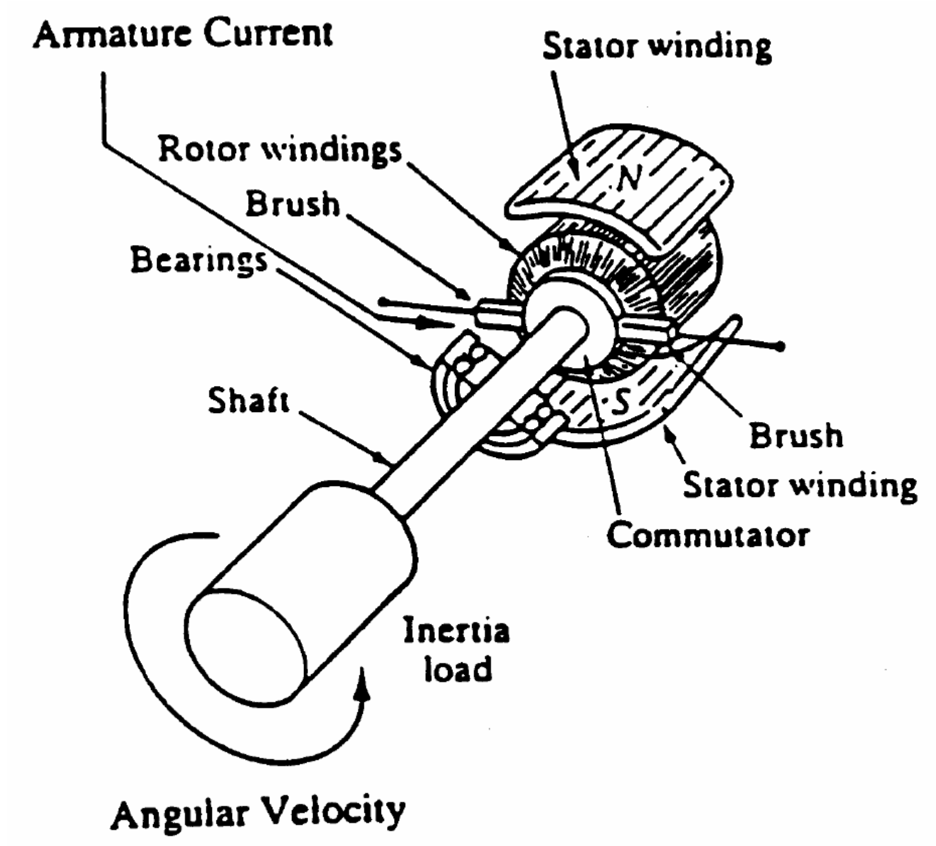




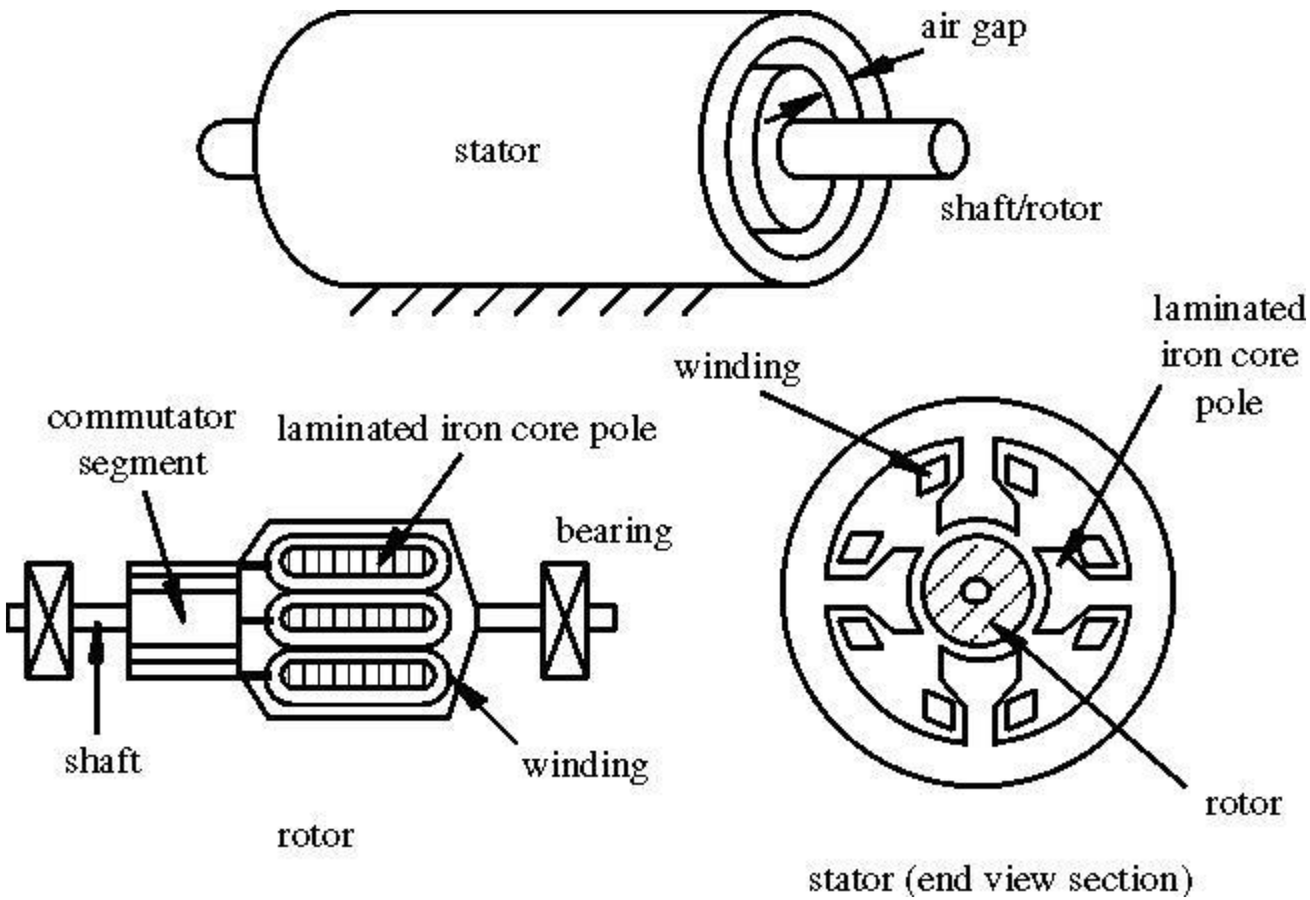
Each copper segment is connected to the ends of the coil. The commutator is composed of multiple segments.

Commutation

Elements of a D.C. Motor

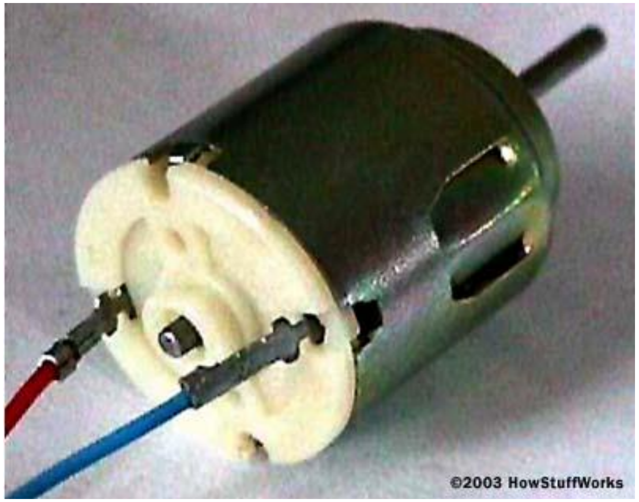


Elements of DC Motor Showing Construction

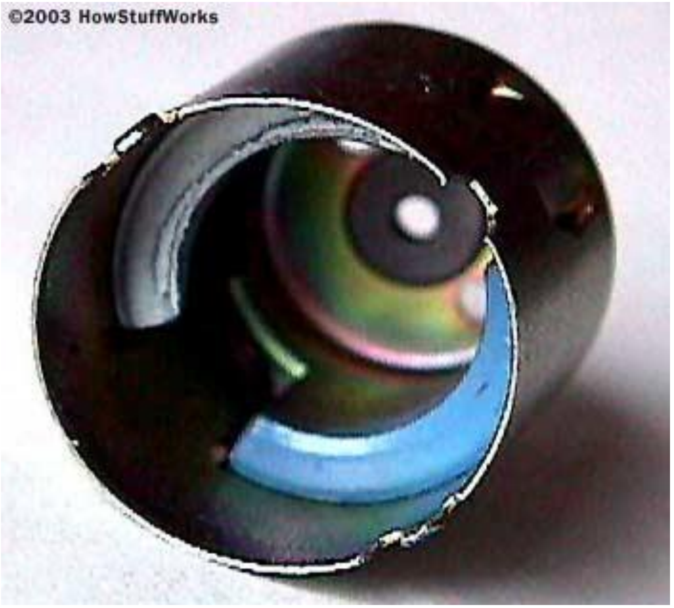


D.C. Motor Pictures

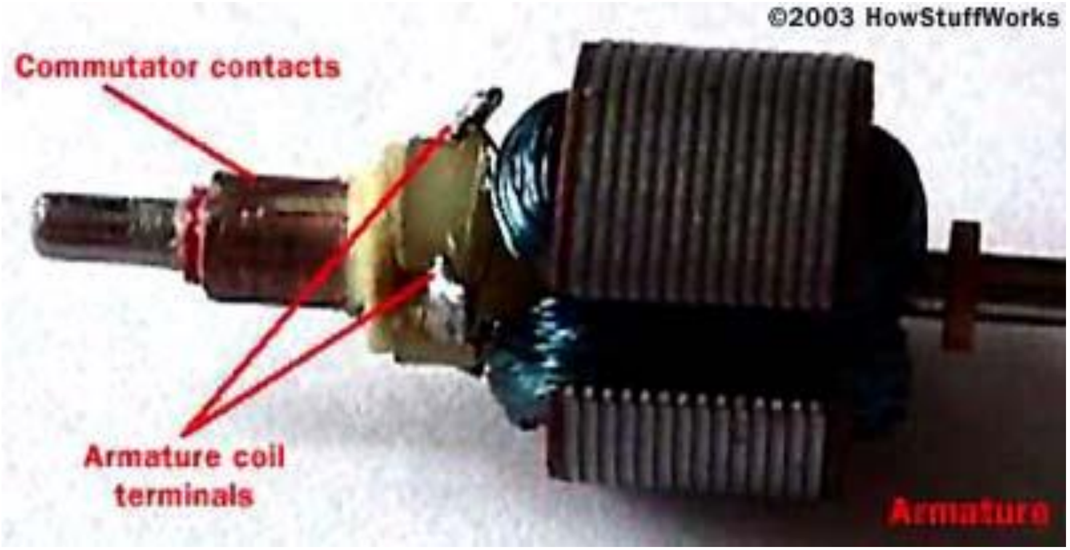
Small permanent magnet DC motor.

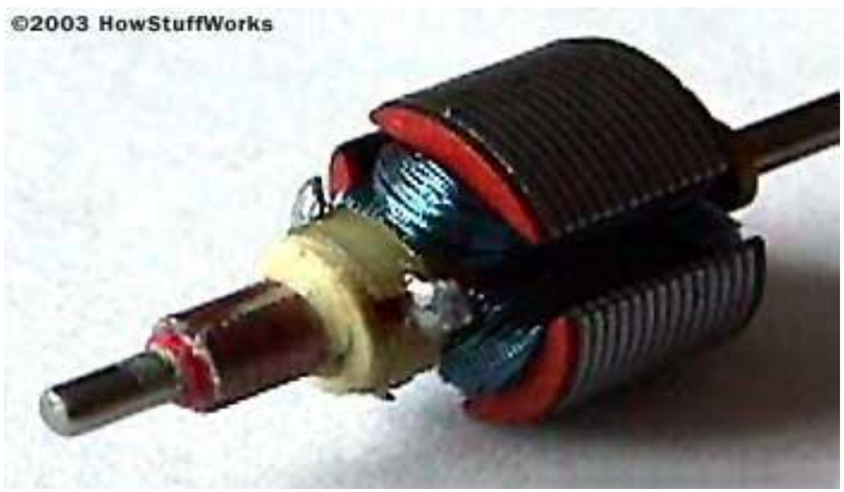


Stator with Magnets



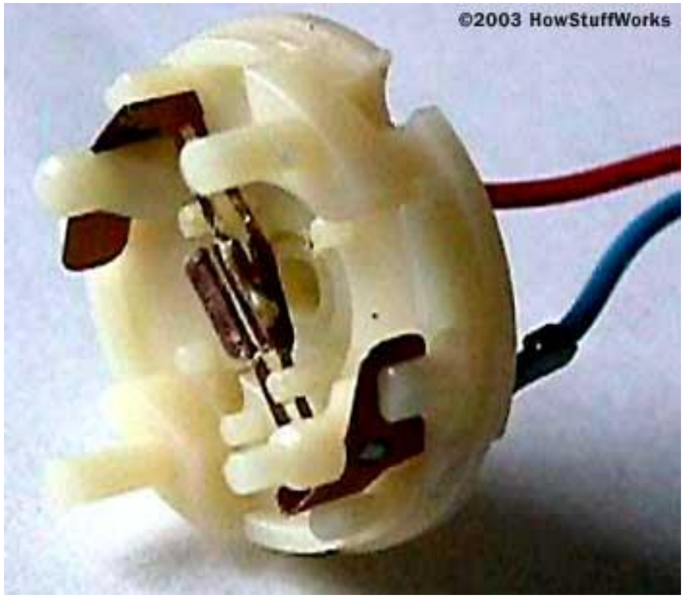
D.C. Motor Armature (Rotor) & Commutator

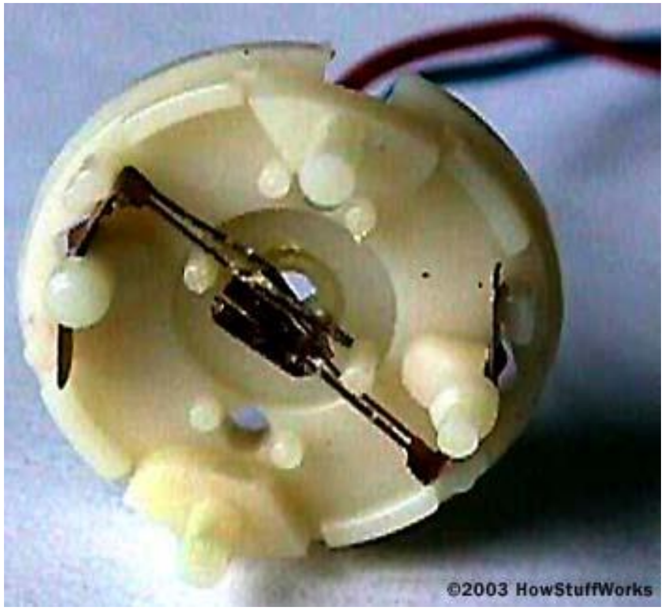




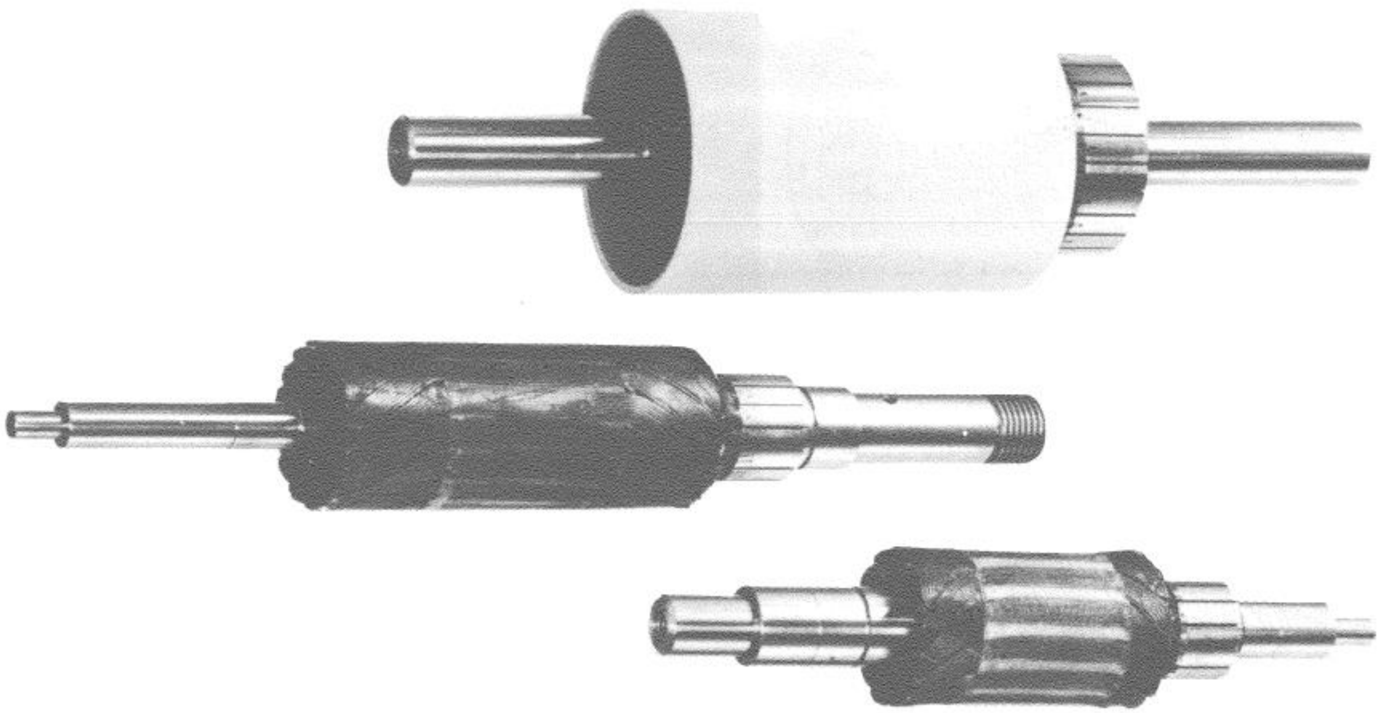
Armature has 3 coils so commutator would have 6 segments.

Brushes

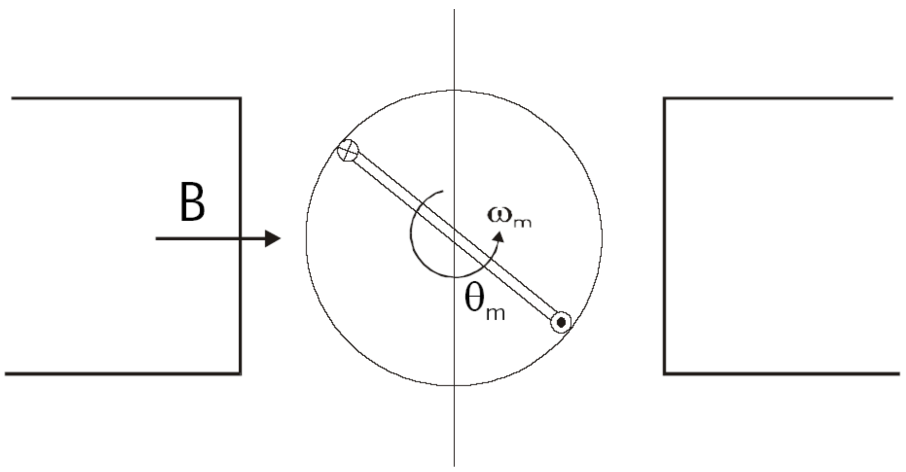




Moving Coil D.C. Motors



Back e.m.f (ElectroMotive Force - Voltage)



According to Faraday's law, a coil of length L rotating in a uniform magnetic field of flux density B will generate an e.m.f. given by :

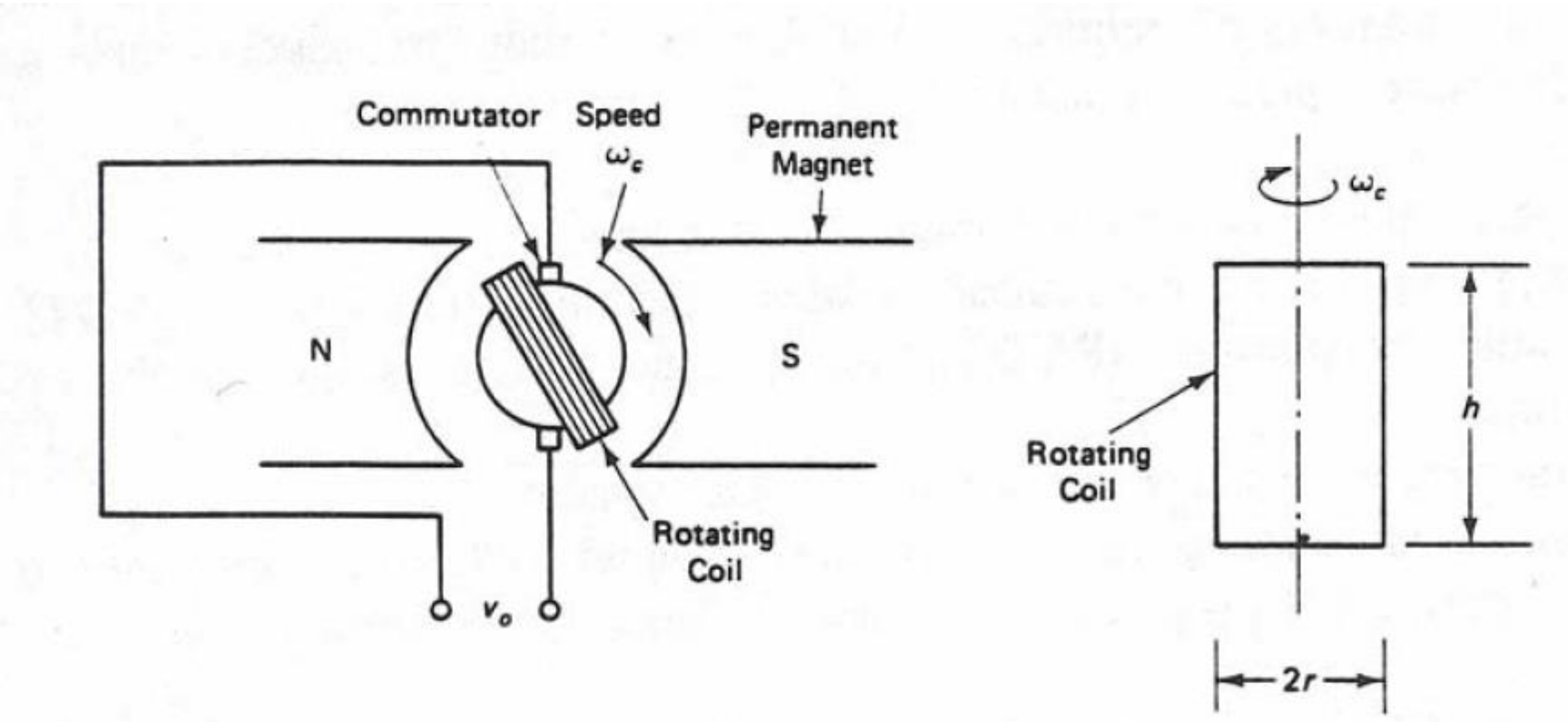
$$e = -\frac{d\,\lambda}{d\,t}$$

where λ is the flux linking the coil

Voltage $e = 2\,B\,L\,r\,\omega_n$

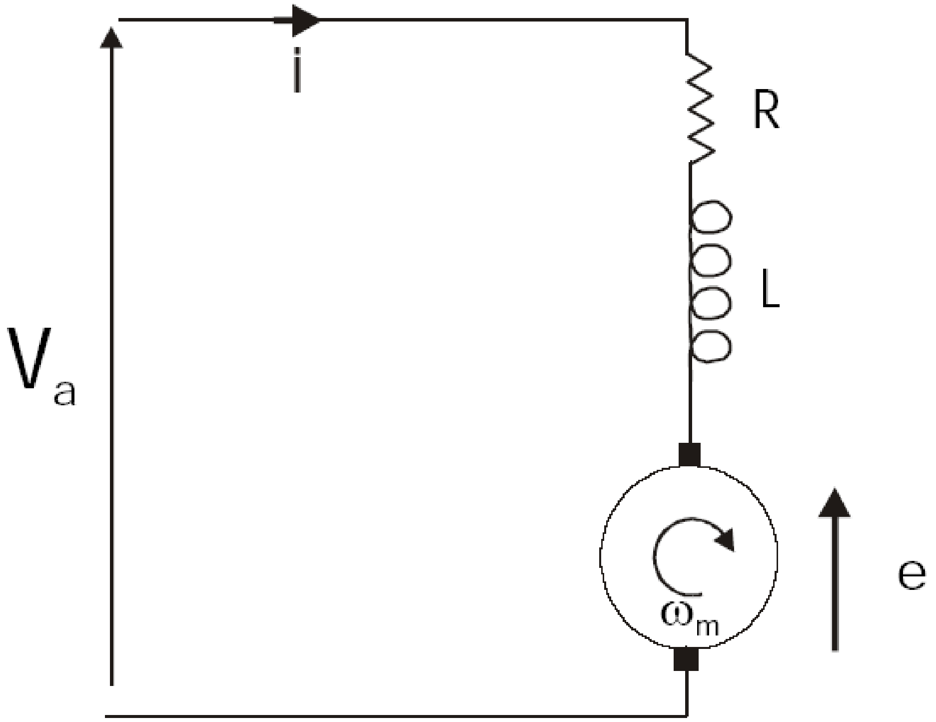
$= K_E\ \ \omega_n$

Tachometer

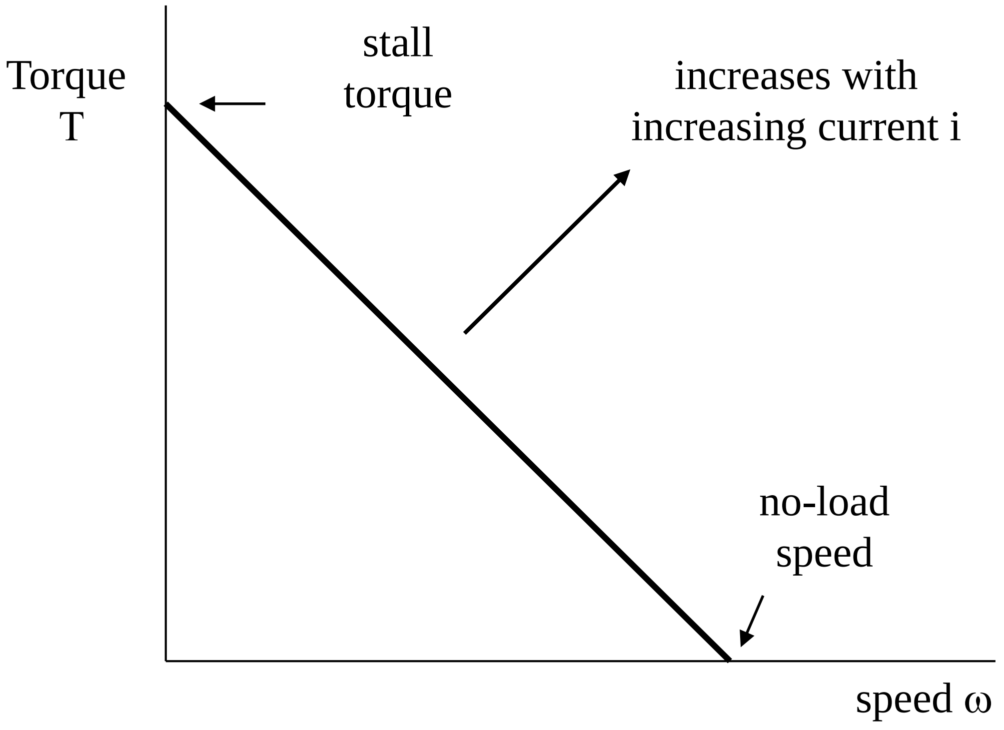


Commutator
Speed
ω_c
Permanent
Magnet
N
S
Rotating
Coil
v_o
ω_c
h
Rotating
Coil
$2r$

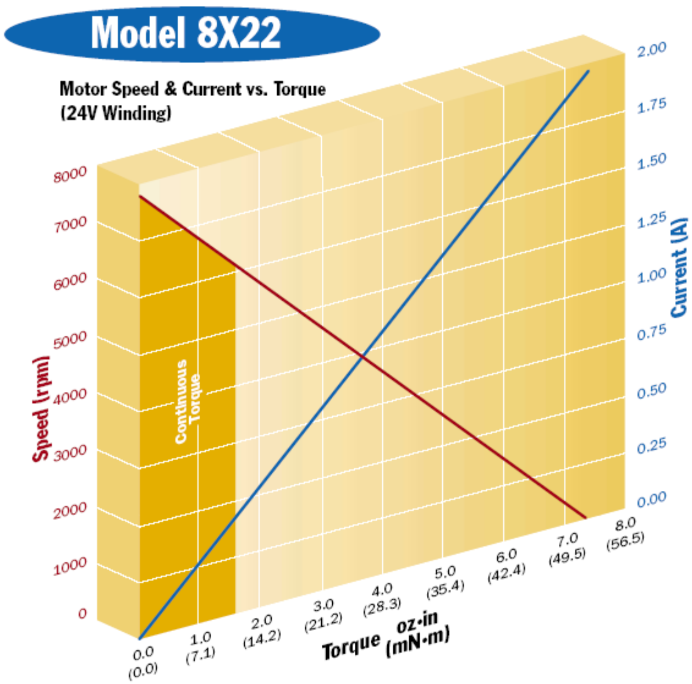
Electrical Side of the DC Motor



Torque – Speed – Current



Pittman Motor Specifications



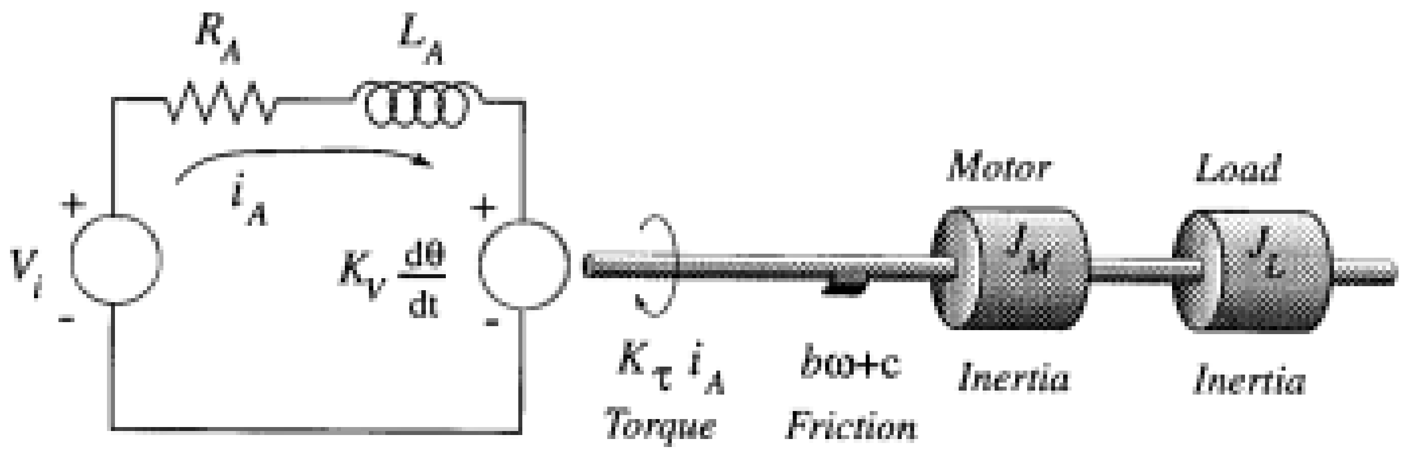
Motor Data

Line No.	Parameter	Symbol	Units	8X22
1	Continuous Torque (Max.)[1]	T_C	oz·in (N·m)	1.6 (11.2×10^{-3})
2	Peak Torque (Stall)[2]	T_{PK}	oz·in (N·m)	7.4 (52.0×10^{-3})
3	Motor Constant	K_M	oz·in/√W (N·m/√W)	1.12 (7.9×10^{-3})
4	No-Load Speed	S_{NL}	rpm (rad/s)	7847 (822)
5	Friction Torque	T_F	oz·in (N·m)	0.35 (2.5×10^{-3})
6	Rotor Inertia	J_M	oz·in·s^2 (kg·m^2)	1.4×10^{-4} (9.89×10^{-7})
7	Electrical Time Constant	τ_E	ms	0.52
8	Mechanical Time Constant	τ_M	ms	15.6
9	Viscous Damping—Infinite Source Impedance	D	oz·in/krpm (N·m/(rad/s))	0.0153 (1.03×10^{-6})
10	Viscous Damping—Zero Source Impedance	K_D	oz·in/krpm (N·m/(rad/s))	0.92 (6.20×10^{-5})
11	Maximum Winding Temperature	θ_{MAX}	°F (°C)	311 (155)
12	Thermal Impedance	R_{TH}	°F/watt °C/watt	75.9 (24.4)
13	Thermal Time Constant	τ_{TH}	min	7.75
14	Motor Weight (Mass)	W_M	oz (g)	4.69 (133.0)
15	Motor Length, 81XX/82XX	L_1	in max (mm max)	2.070 (52.6)
16	Motor Length, 83XX/84XX	L_1	in max (mm max)	2.007 (51)

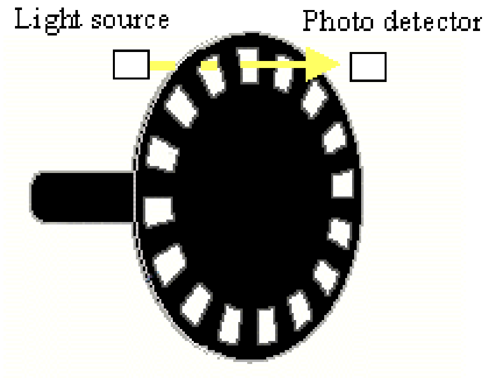
Model 8XX2 Winding Data (Other windings available upon request)

Line No.	Parameter	Symbol	Units	8X22			
17	Reference Voltage	E	V	12.0	19.1	24.0	30.3
18	Torque Constant	K_T	oz·in/A (N·m/A)	1.94 (13.7×10^{-3})	3.07 (21.7×10^{-3})	3.88 (27.4×10^{-3})	4.88 (34.5×10^{-3})
19	Back-EMF Constant	K_E	V/krpm (V/rad/s)	1.43 (13.7×10^{-3})	2.27 (21.7×10^{-3})	2.87 (27.4×10^{-3})	3.61 (34.5×10^{-3})
20	Resistance	R_T	Ω	3.10	7.61	12.1	19.1
21	Inductance	L	mH	1.57	3.93	6.27	9.92
22	No-Load Current	I_{NL}	A	0.25	0.16	0.12	0.10
23	Peak Current (Stall)	I_P	A	3.88	2.51	1.99	1.59

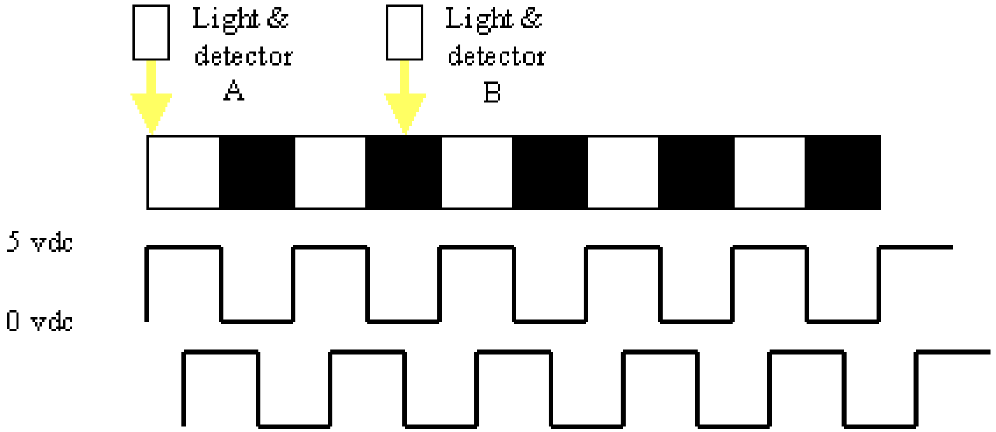
Model of a DC Motor



Optical Encoder

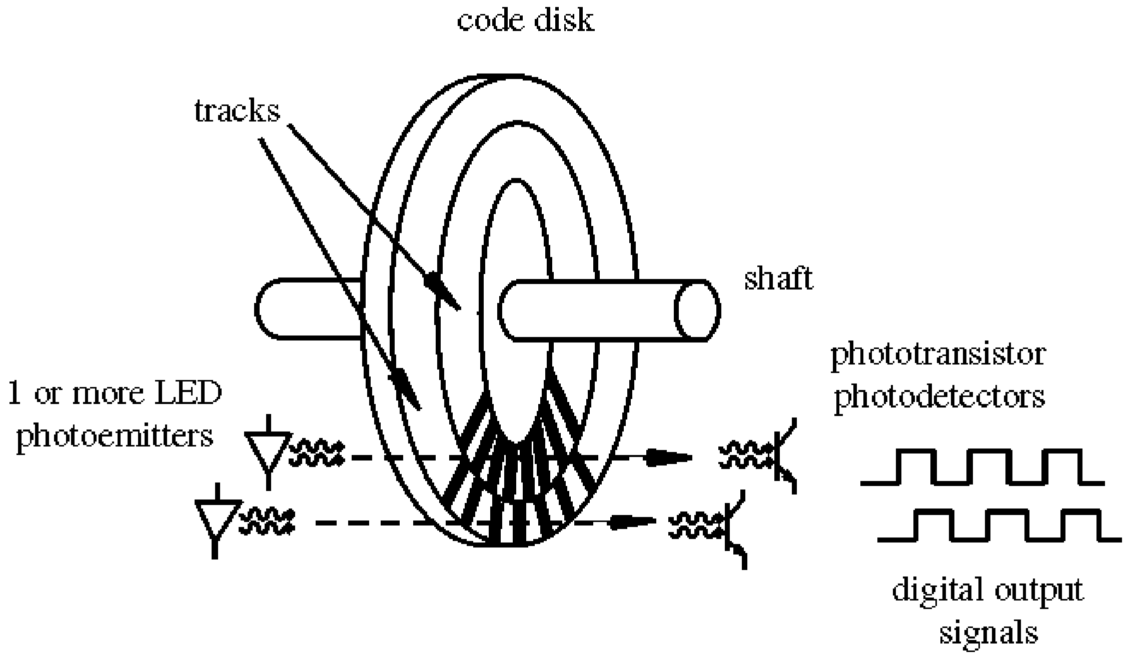


(a) operation



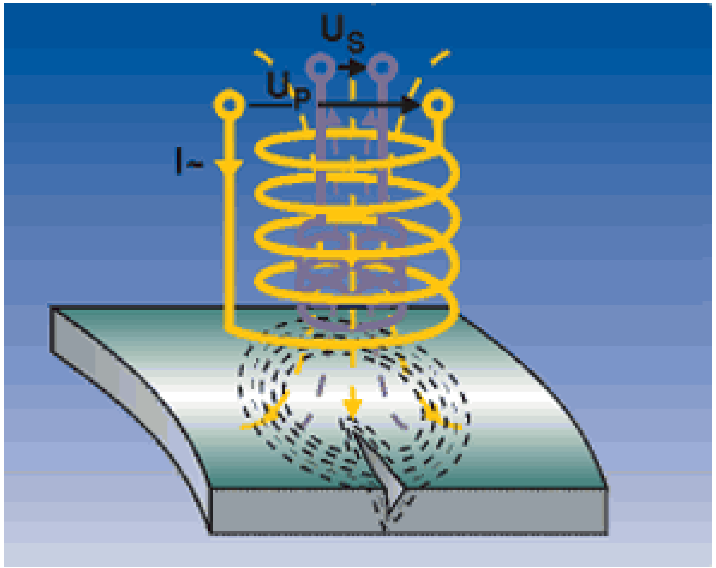
(b) quadrature

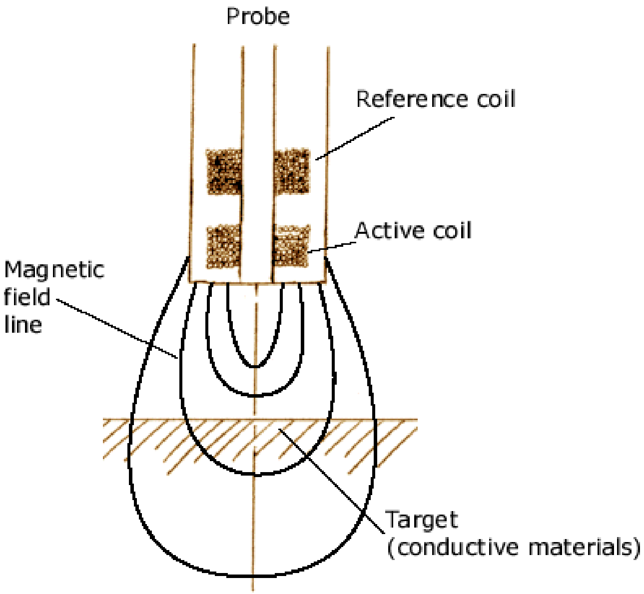
Optical Encoder Operation



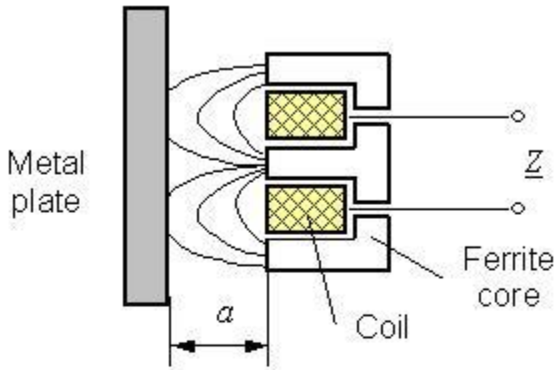
Eddy Current Sensor

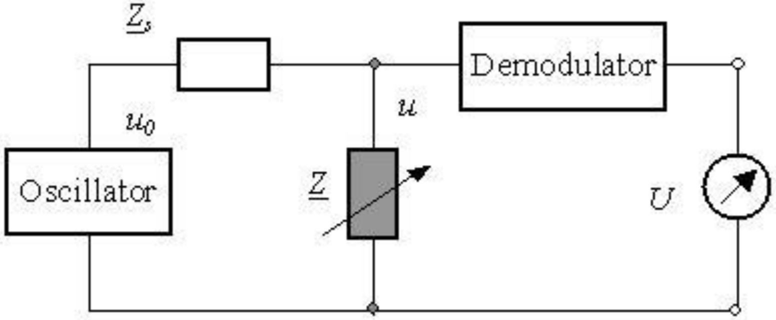
The **Eddy Current Sensor** uses the effect of eddy (circular) currents to sense the proximity of non-magnetic but conductive materials. Some eddy current transducers contains two coils: an **active coil** (main coil) and a **balance coil**. The active coil senses the presence of a nearby conductive object, and balance coil is used to balance the output bridge circuit and for temperature compensation.

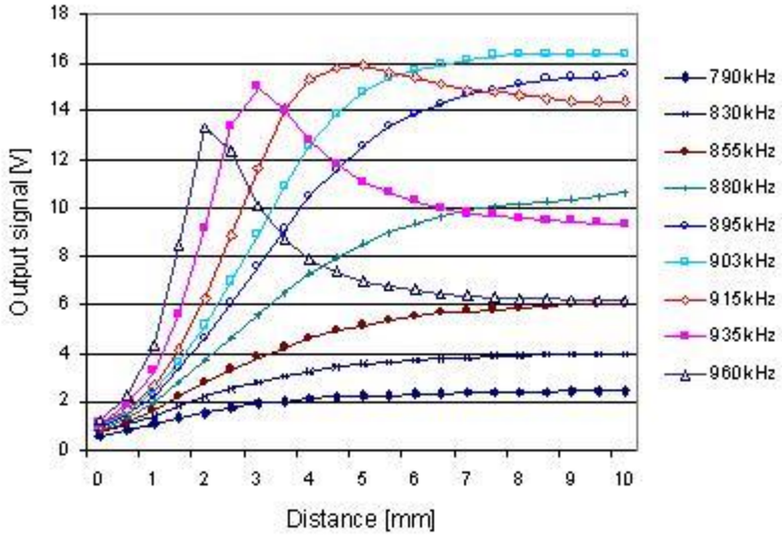


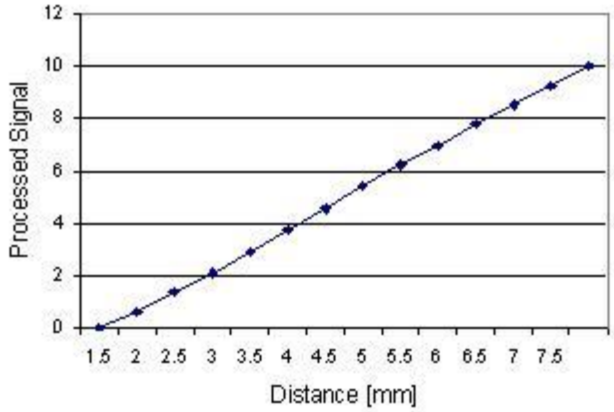


Demodulation & Processing

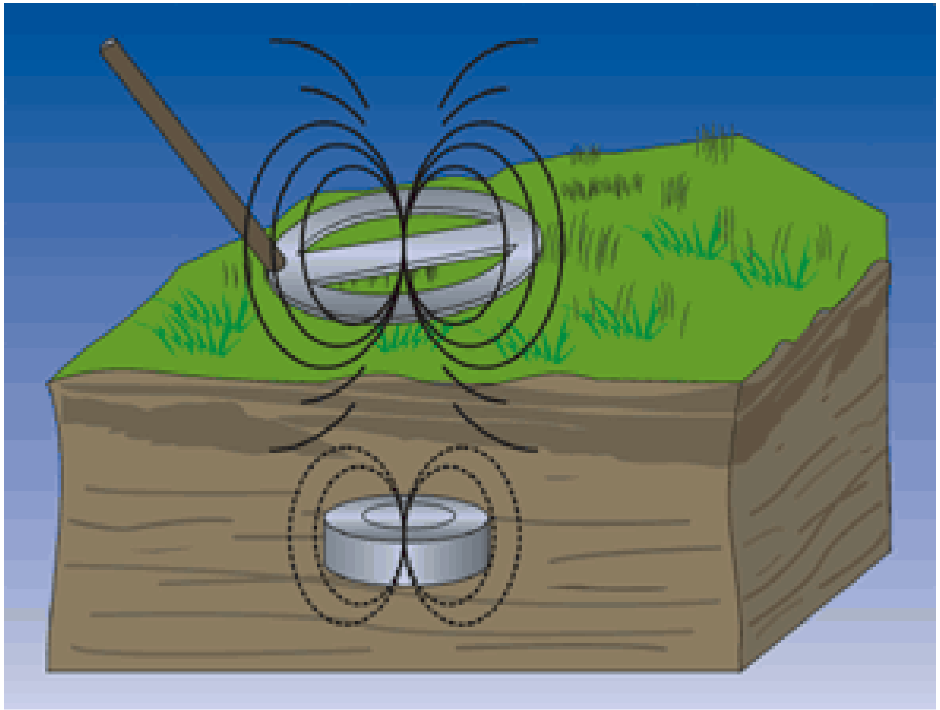








Metal Detectors



Common Specifications

Common specifications for commercially available eddy current transducers are listed below:

- *Size*: about 2 to 75 mm in diameter, 20 to 40 mm long
- *Range*: 0.25 to 30 mm
- *Resolution*: Up to 0.1 μm
- *Nonlinearity*: 0.5%
- *Bridge Circuit Frequency*: 50 kHz to 10 MHz

DC Motor Driver