Programming BASIC Stamp II

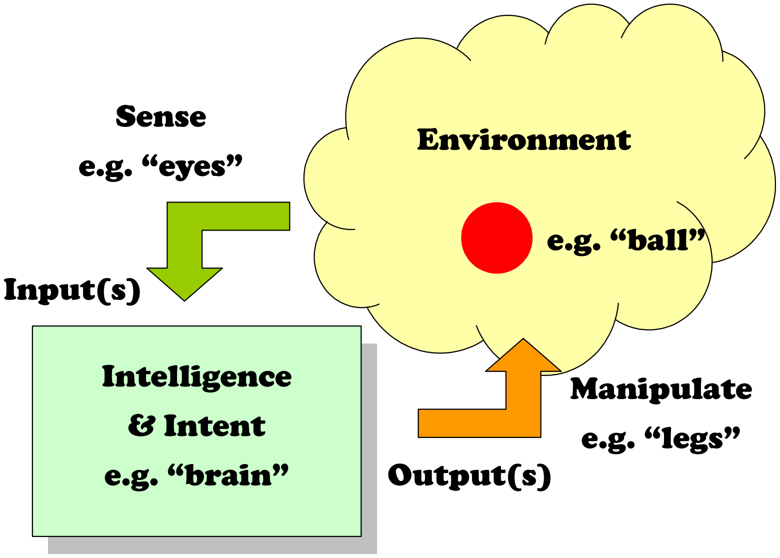
SS-3406 Introduction to Robotics

RECAP

A Robot

- Four major aspects:
  - Intent
  - Intelligence
  - Sensing
  - Actions

+ Human made



Sense
e.g. "eyes"

Input(s)

Environment

e.g. "ball"

Intelligence
& Intent
e.g. "brain"

Output(s)

Manipulate
e.g. "legs"

"Brain" for the intelligence

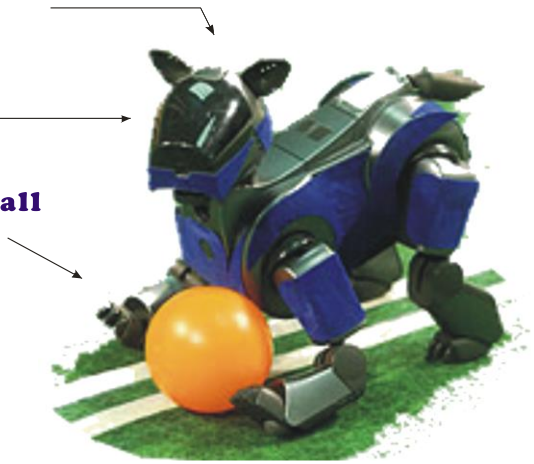
"Intent" to win the game

"Eyes" to sense the ball

"Legs" to manipulate the ball

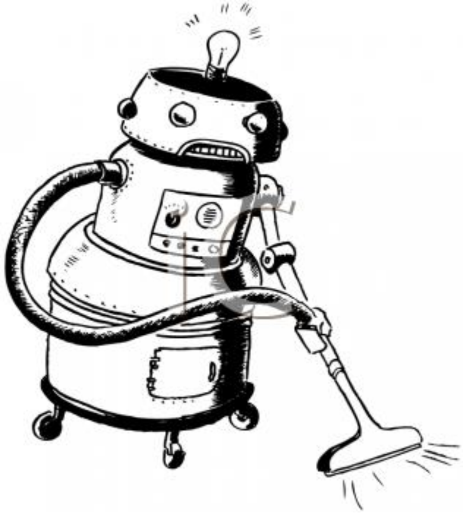
Human made!

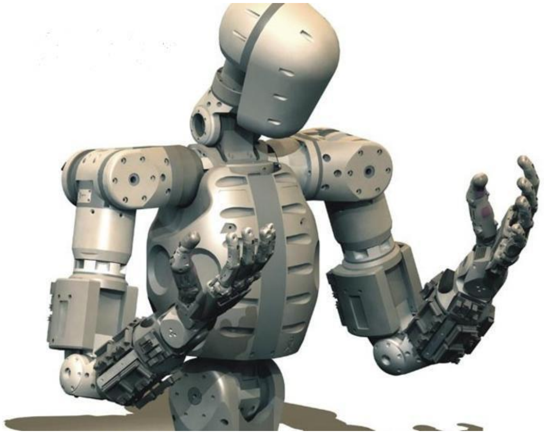
SS-3406

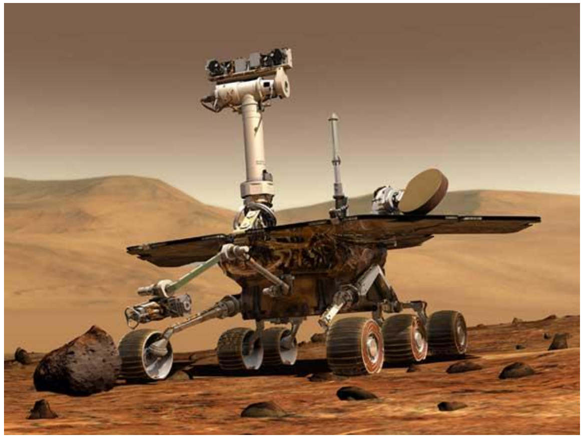


Intent

- Your imagination!



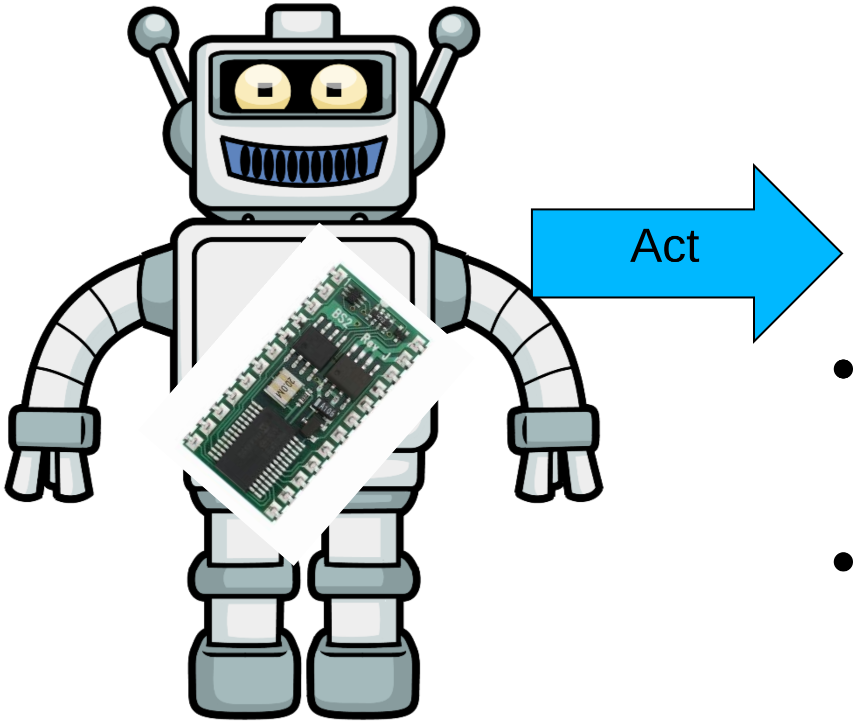




Intelligence

- The programming.
  - Correct instructions.
  - Correct order of instructions.
  - Keep doing things unconditionally (DO … LOOP).
  - Counting (FOR … NEXT).

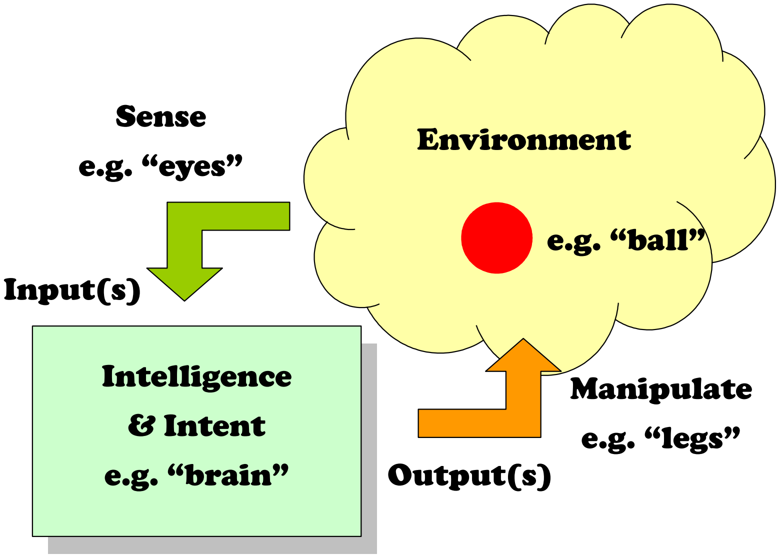
Actions



- Perceptual:
  - Message on screen: DEBUG.
  - Lighting: LED.
  - Sound: Buzzer.
- Locomotion:
  - Motors: Servo.
- Manipulation:
  - Motors: Servo.

So Far …

- Four major aspects:
  - Intent ○
  - Intelligence √
  - Sensing
  - Actions √ √ √ √



"Brain" for the intelligence

"Intent" to win the game

"Eyes" to sense the ball

"Legs" to manipulate the ball

Human made!

SS-3406

PROGRAMMING: SENSORS

DEBUGIN

- DEBUG: MCU sends message to PC (user)
- DEBUGIN: PC (user) sends (**keyboard**) message to MCU
  - Types of input: DEC, BIN, NUM, etc
  - See: http://www.parallax.com/go/PBASICHelp/Content/LanguageTopics/Commands/DEBUGIN.htm
- Exercise: Can you figure out what the program will do? Try it out.

```
waitTime VAR Word

DO
  DEBUG "Hello!  How long should I wait (in ms)? Please enter a value: "
  DEBUGIN DEC waitTime
  PAUSE waitTime
LOOP
```

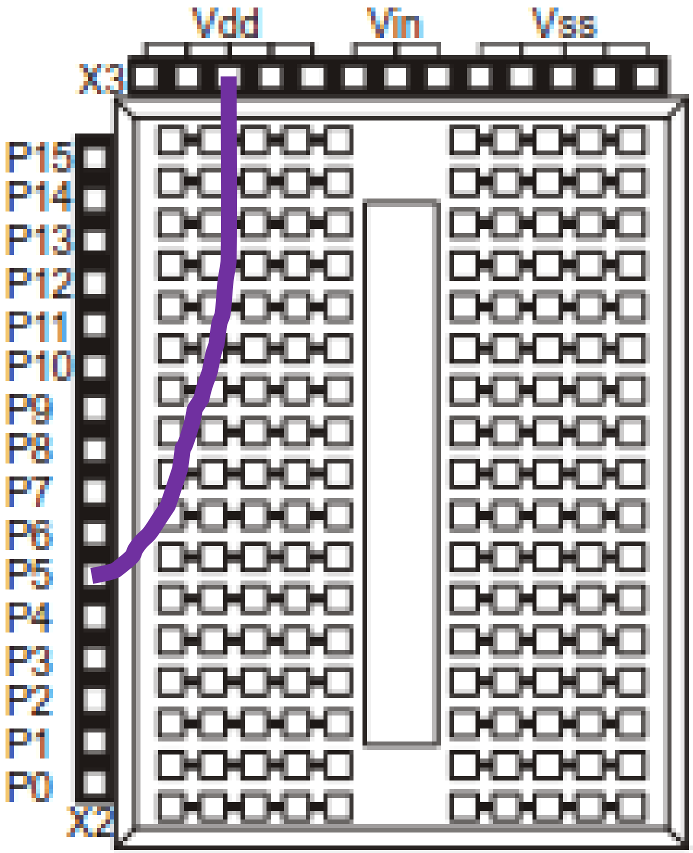
Reading Inputs

- IN#, e.g. to read PIN 5

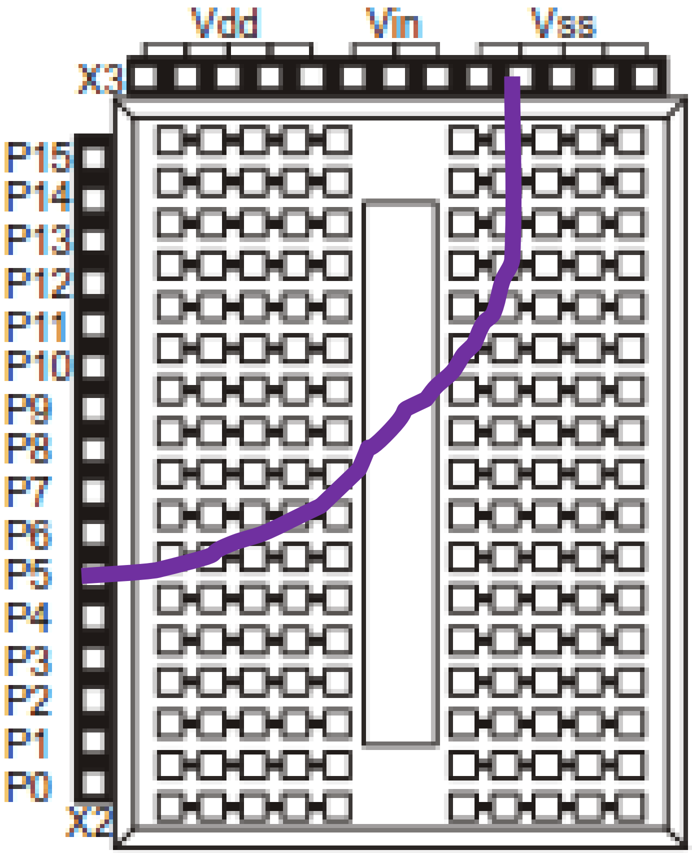
```
myVariable VAR Bit    ‘a variable

myVariable = IN5      ‘the variable will have the value of PIN5 (1 or 0)
```

- Exercise: try to manually (use wire) supply 1 (Vdd) or 0 (Vss) to PIN 5 and read it.
  - Use LOOP.
  - Use DEBUG
  - What happen when the wire (try to touch the wire) is not connected (open)? Do you get 1 or 0?

Manually Give 1 or 0

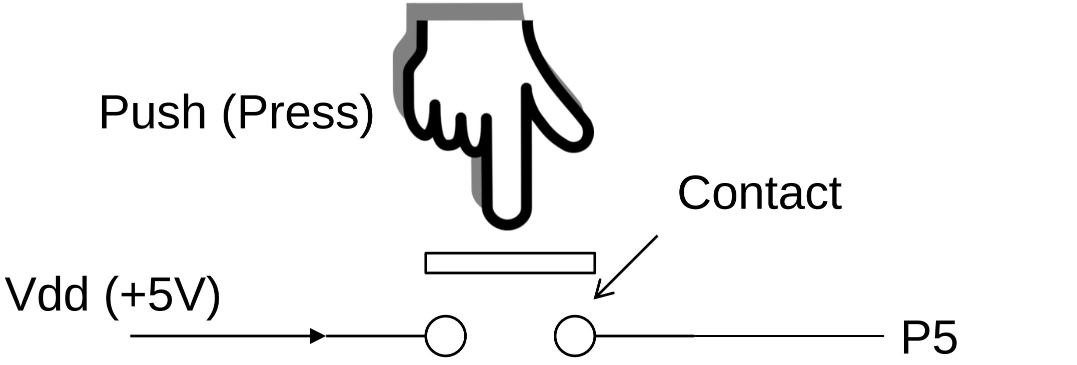


P5 = 1



P5 = 0

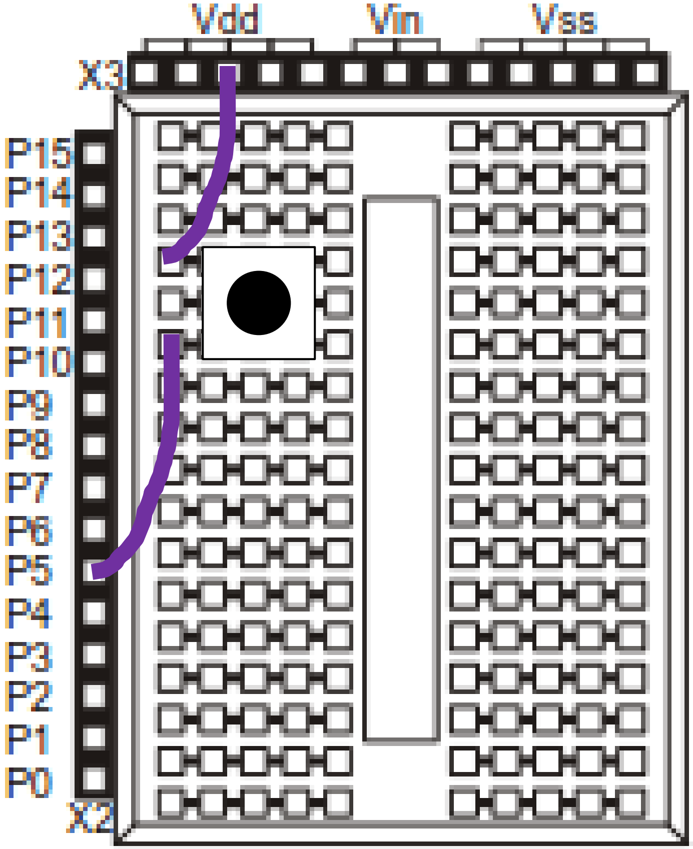
Push Button Switch

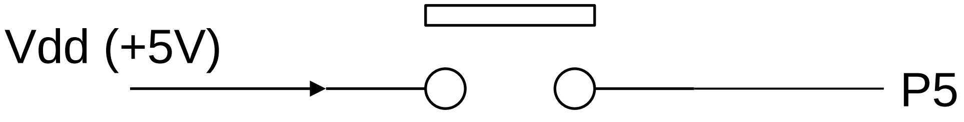


Open: no voltage, undefined
Closed: +5V (1)

- Exercise: use the same program to read the input (PIN 5) and see the effect of pressing/releasing the switch and touching the wires in both states.
- How do we make “open” to give 0, i.e. to Vss?

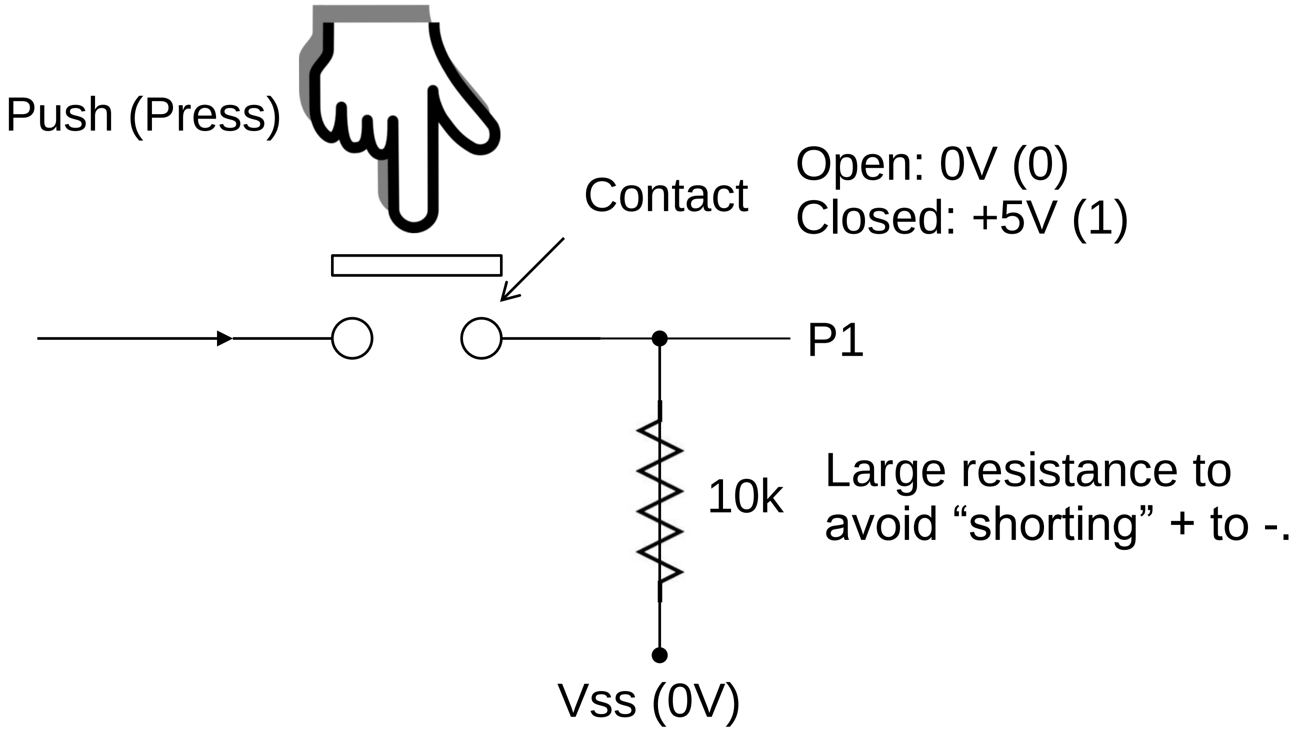
Incomplete Switch Connection





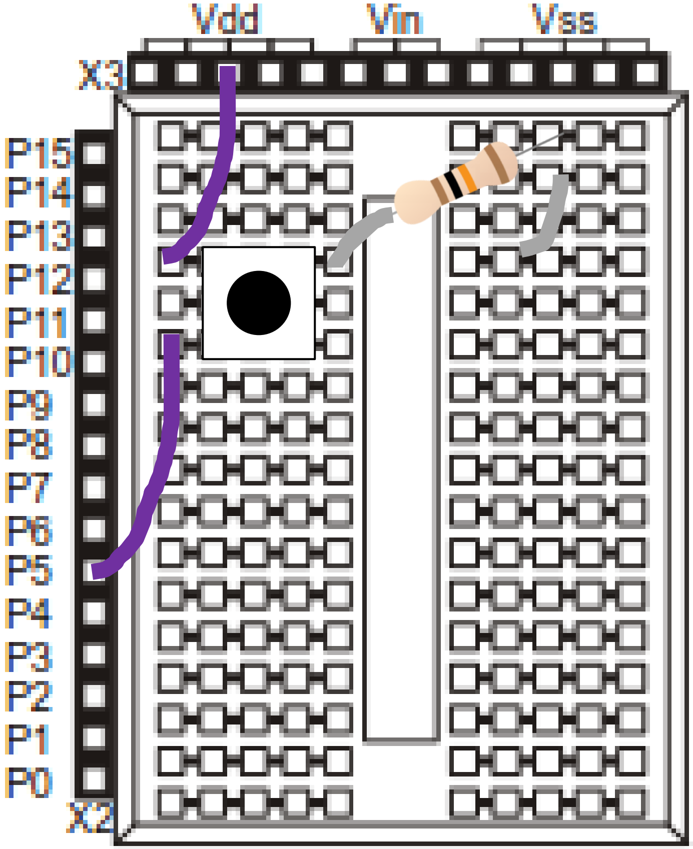
Pressed: 1 (Vdd)
Not pressed: Undefined (floating)

Push Button Switch Circuit

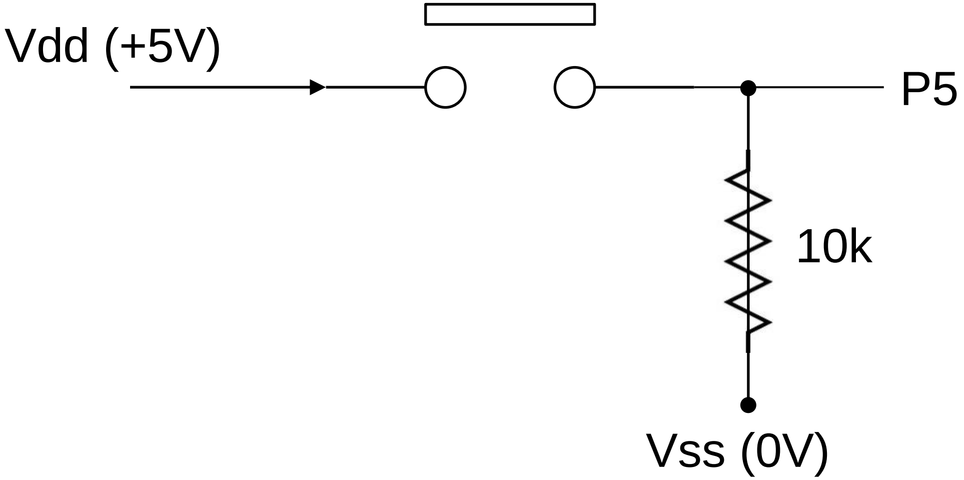


- Exercise: With the addition of the above connection, try the program that reads the switch – press / release.

Switch Connection

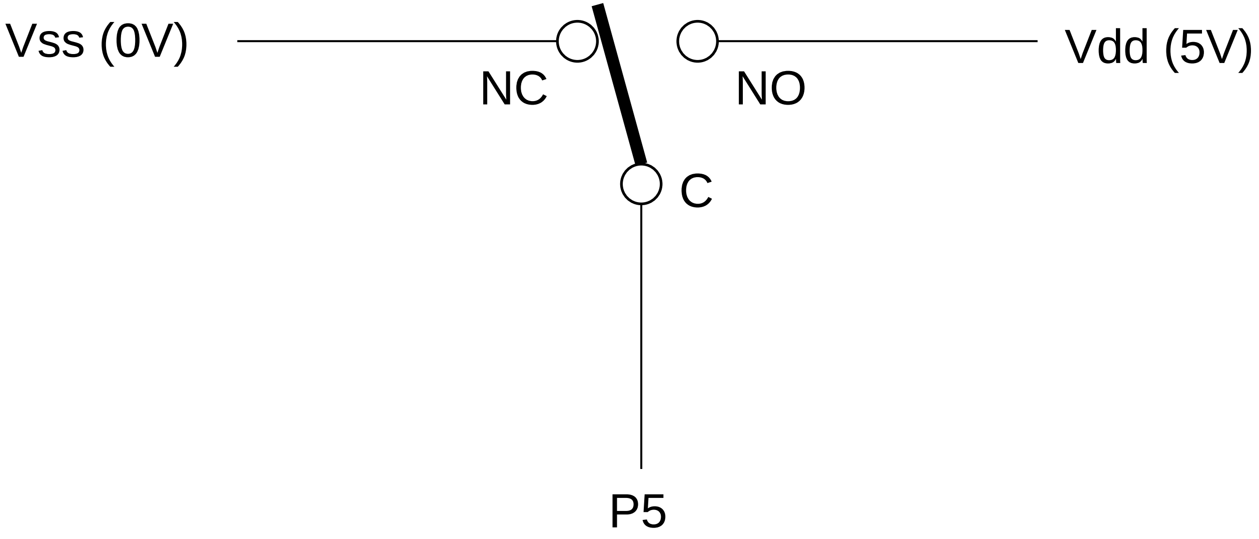


Pressed: 1 (Vdd)
Not pressed: 0 (Vss)



10k = 10,000 = 1,0, 3 zeros = brown, black, orange

Limit Switch Connection



```
myVariable VAR Bit    'a variable

DO
    myVariable = IN5      'the variable will have the value of PIN5 (1 or 0)
    DEBUG ? myVariable     'display the reading
LOOP
```

Controls in Programming

- We can’t do complex or intelligent task if instructions must follow a straight line sequence: need to be able to “control” the order of the executions (flow of the program) in real time.
- Loop
  - DO … LOOP
    - Variations: DO UNTIL (condition) … LOOP, DO WHILE (condition) … LOOP
  - FOR Counter = StartValue TO EndValue … NEXT
    - Variation: FOR Counter = StartValue TO EndValue {STEP StepValue} ... NEXT
- Decision
  - IF (condition) THEN … ELSE … ENDIF
    - Variation: IF (condition) THEN … ELSEIF (condition) … ELSE … ENDIF
- Delay
  - PAUSE

IF (condition) … ELSE

- Conditions:
  - Equal: =
  - Not Equal: <>
  - Less than: <
  - Greater than: >

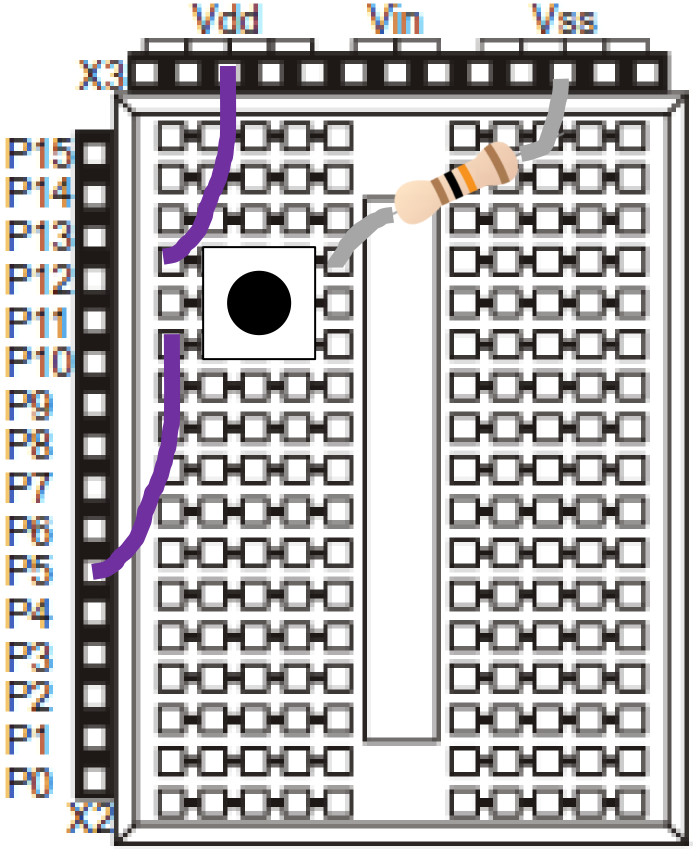
```
IF (condition)
  ' Actions for condition true
ELSE
  ' Actions for condition is false
```

- Conditions control the execution of the program.
- Exercise: Try:

```
testVar VAR byte
testVar = 3

IF (testVar = 4) THEN
  DEBUG "Same"
ELSE
  DEBUG "Not the same"
ENDIF
```

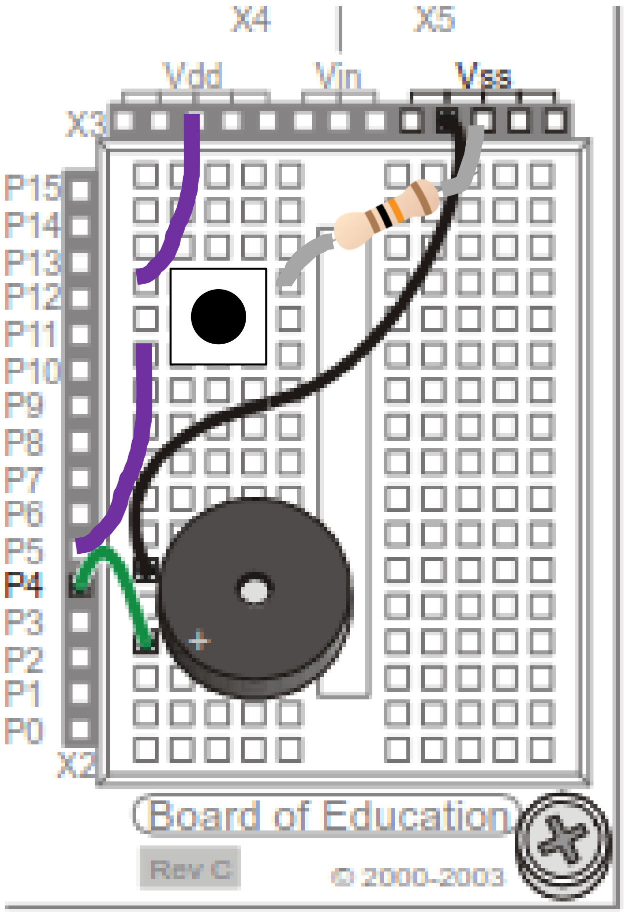
Exercises

- Modify your program such that when you press the switch, a message “Ouch!” is displayed (use DEBUG) and when the switch is not pressed a message “Hmmm” is displayed.



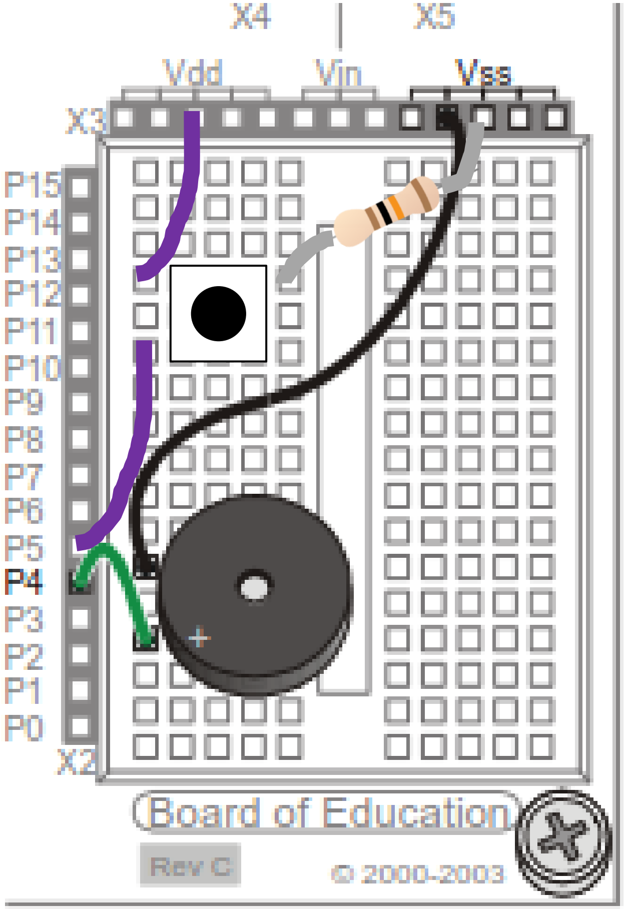
Exercise

- Add a buzzer to the circuit.
- Modify the program so that when the switch is pressed, the buzzer will sound, and when the switch is not pressed, the buzzer is silent.



Exercise

- Further modify the program such that when the switch is pressed, the buzzer gives out high pitch, and when the switch is released, the buzzer gives low pitch.



Exercise

- Add LED to the previous circuit.
- Do something interesting with flashing the LED and buzzing the buzzer when the switch is being pressed and released.
- Can you count the number of times the switch is being pressed?

DO UNTIL , DO WHILE

- DO … LOOP “unconditionally” repeats the block of instructions, i.e. infinitely.
- We can set a “condition” to stop the loop.
  - DO UNTIL (condition) ... LOOP
    - Keep doing if *condition* is **not true**, until *condition* becomes true.
  - DO WHILE (condition) ... LOOP
    - Keep doing if *condition* is **true**. It will stop if *condition* is/becomes false.

DO UNTIL

- Exercise: Try

```
testVar VAR byte
testVar = 0

DEBUG CLS, "Enter a character: ", CR

DO UNTIL (testVar = "q")
  DEBUGIN STR testVar \1
  DEBUG "You have typed: ", testVar, CR
  DEBUG "Enter a character: ", CR
LOOP

DEBUG "You hit q, quit!"
```

Combining Conditions: OR

- In BASIC Stamp, 0 = False, any other value = True.
- E.g. 1, 5, 23, etc = True. Usually, we use 1 only.

- OR operation:
  - A OR B is True if either A OR B is True.
  - OR is represented by a vertical line | in programming
  - E.g. A OR B is written as A | B

```
a VAR Bit
b VAR Bit
a = 1 ' True
b = 0 ' False

' Try to change the state (True/False) of the above variable and see the output of the program
DO
  DEBUG "Enter 1 or 0, a = "
  DEBUGIN BIN1 a
  DEBUG " Enter 1 or 0, b = "
  DEBUGIN BIN1 b
  IF (a | b) THEN
    DEBUG " a OR b is True"
  ELSE
    DEBUG " a OR b is False"
  ENDIF
  DEBUG CR
LOOP
```

Combining Conditions: AND

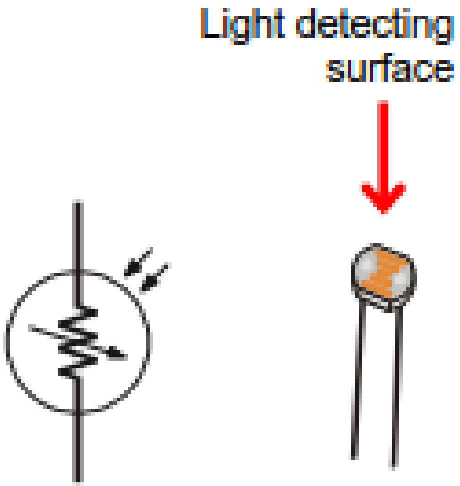
- In BASIC Stamp, 0 = False, any other value = True.
- E.g. 1, 5, 23, etc = True. Usually, we use 1 only.

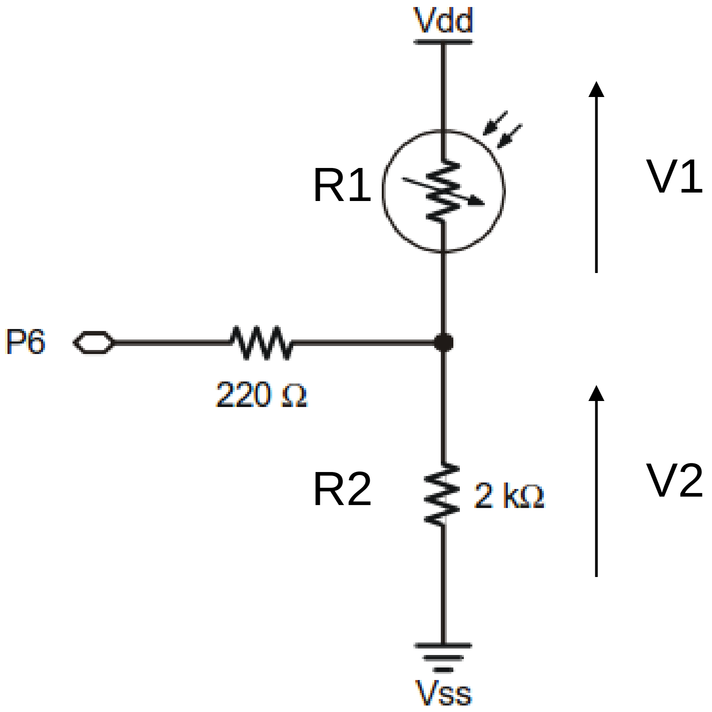
- AND operation:
  - A AND B is True only if A AND B are True.
  - AND is represented by a & symbol in programming
  - E.g. A AND B is written as A & B

```
a VAR Bit
b VAR Bit
a = 1 ' True
b = 0 ' False

' Try to change the state (True/False) of the above variable and see the output of the program
DO
  DEBUG "Enter 1 or 0, a = "
  DEBUGIN BIN1 a
  DEBUG " Enter 1 or 0, b = "
  DEBUGIN BIN1 b
  IF (a & b) THEN
    DEBUG " a AND b is True"
  ELSE
    DEBUG " a AND b is False"
  ENDIF
  DEBUG CR
LOOP
```

Light Sensor: Photoresistor

- Also called LDR (Light Dependent Resistor).

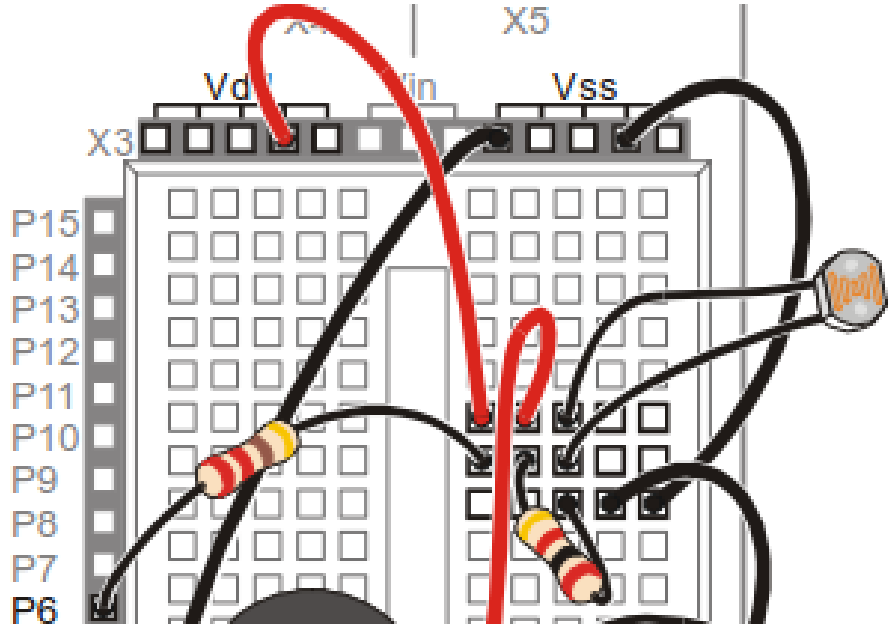


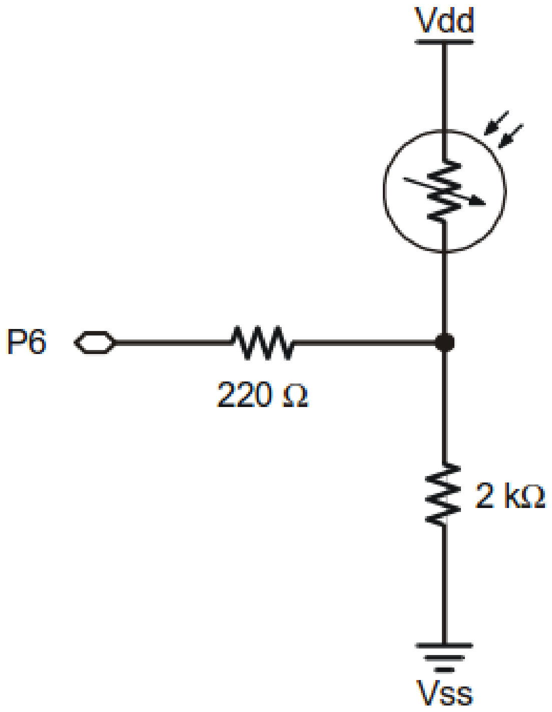


$$V2 = \frac{Vdd * R2}{R1 + R2}$$

```
myVariable VAR Bit

DO
      myVariable = IN6
      DEBUG ? myVariable
LOOP
```

220 = 2,2,1 zero. = red red brown
2000 = 2,0, 2 zeros = red black red

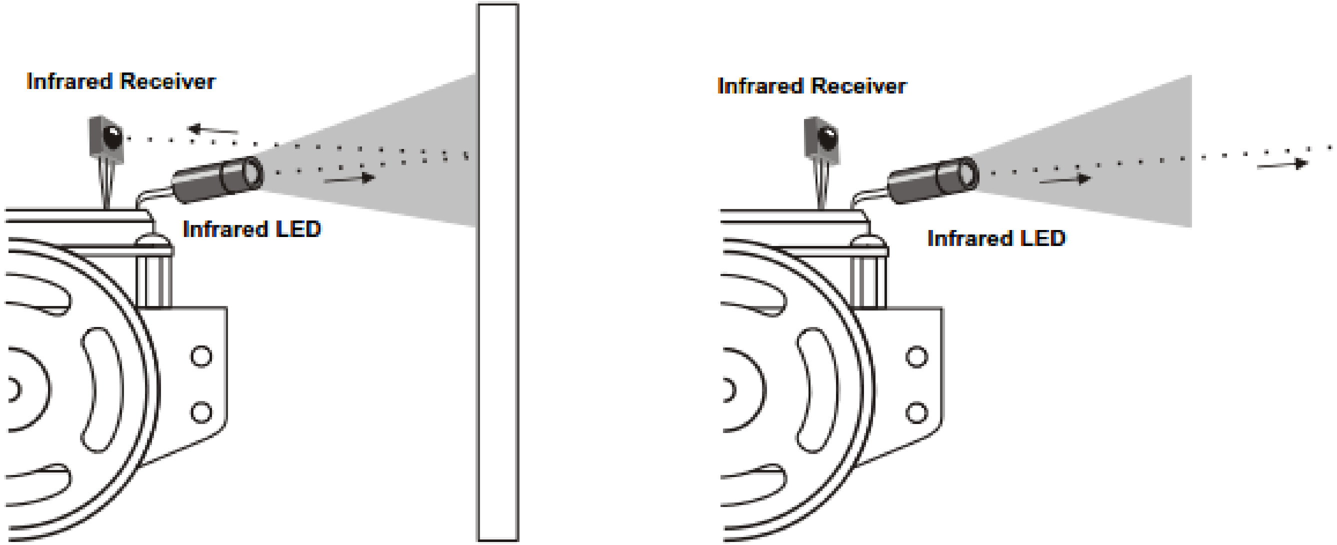




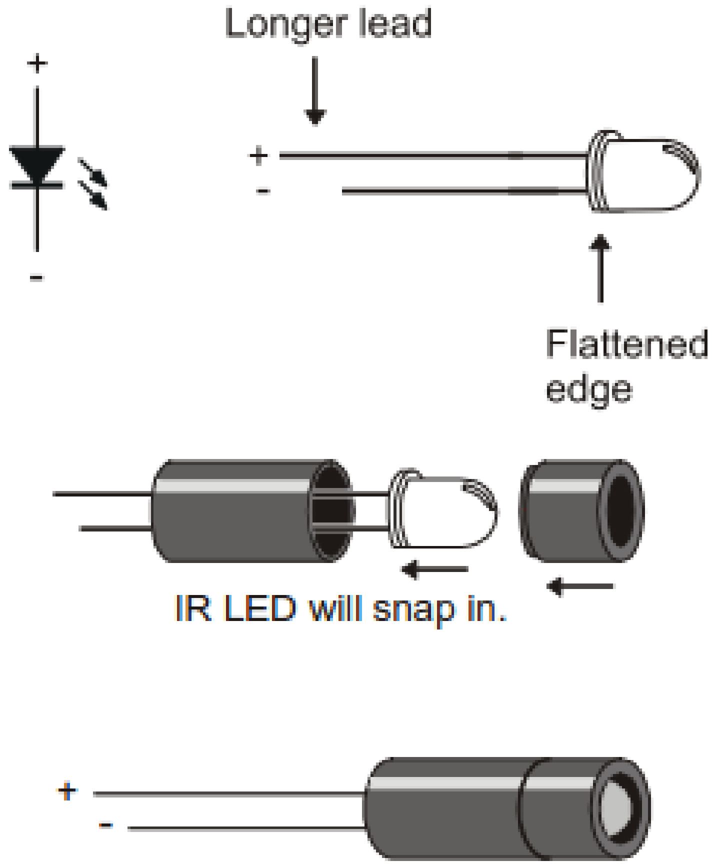
```
DO
  DEBUG "P6 = ", BIN1 IN6, CR
  PAUSE 100
LOOP
```

Opto Sensor 1: IR reflective

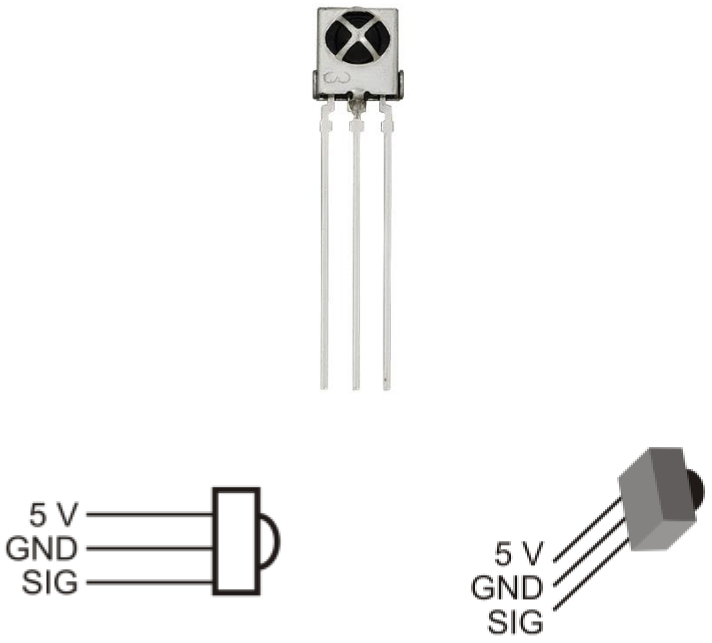
- Usually use to detect objects that reflect the light (IR in this case).
  - IR: Infra Red (invisible light)



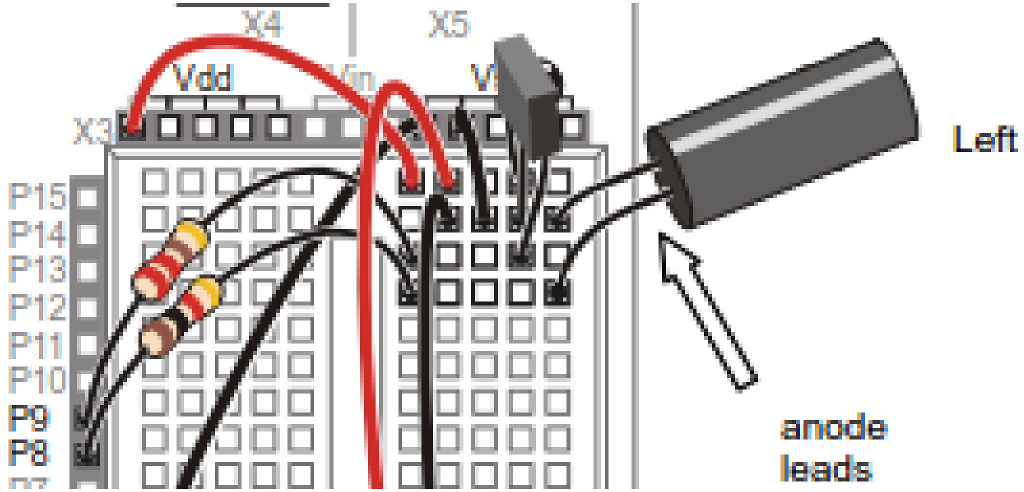
Left: Infrared reflected, obstacle detected. *Right: Infrared not reflected, no obstacle detected.*

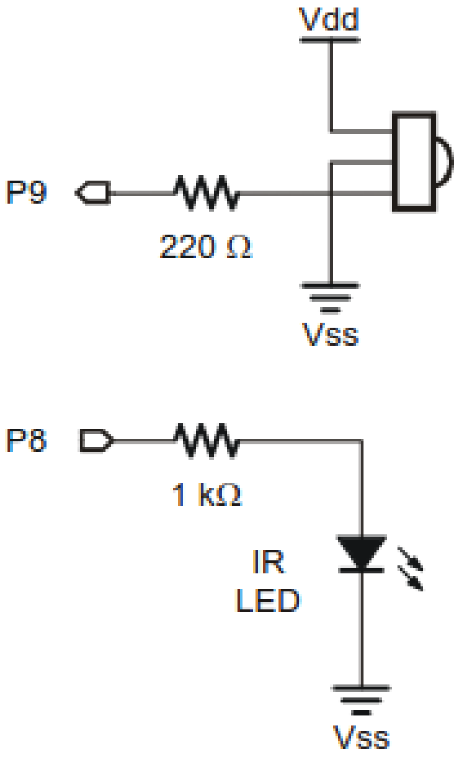


IR Transmitter (LED)



IR Receiver
(Detects IR at 38.5kHz)

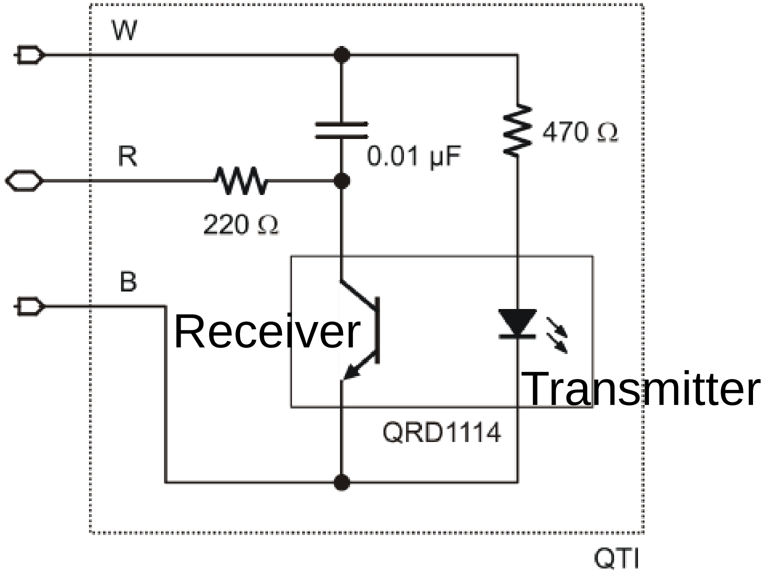




```
irDetectLeft VAR Bit

DO
  FREQOUT 8, 1, 38500 ' Transmit 38.5kHz IR
  irDetectLeft = IN9 ' Read IR receiver
  DEBUG HOME, "irDetectLeft = ", BIN1 irDetectLeft
  PAUSE 100
LOOP
```

Opto Sensor 2: QTI (Reflective)

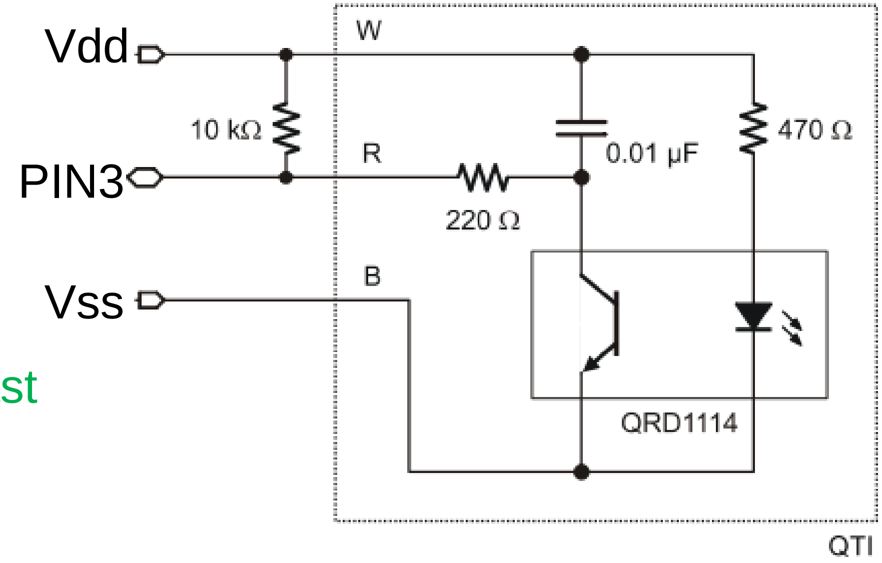
- QTI Line Follower: The IR transmitter sends out IR light, the IR receiver receives the reflected IR light (if any).
  - Receiver turns ON if receives IR, R is connected to B (Vss = 0)
  - Receiver turns OFF if not receive IR, R is connected to W (Vdd = 1)



Programming QTI

- Exercise: Connect W to Vdd, B to Vss, 10k resistor across W and R, read R into PIN3. Check the reading with and without reflection (by white paper at <1 cm).

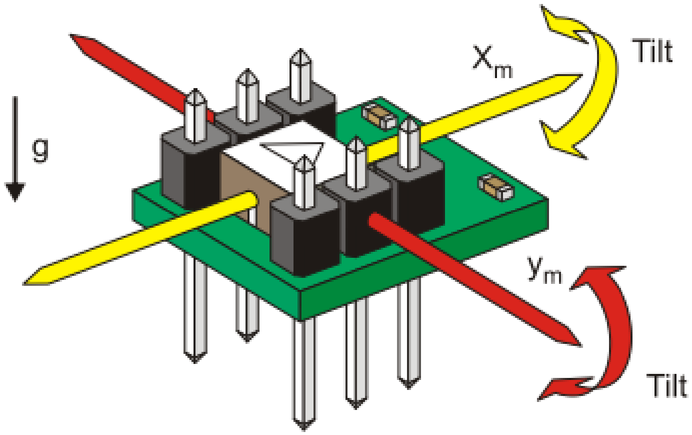
```
qti VAR Bit

DO
   qti = IN3  'Read PIN 3
   DEBUG HOME, BIN1 qti
   PAUSE 100  ‘ Avoid reading too fast
LOOP
```



PROGRAMMING: MORE SENSORS & ACTUATOR/EFFECTOR

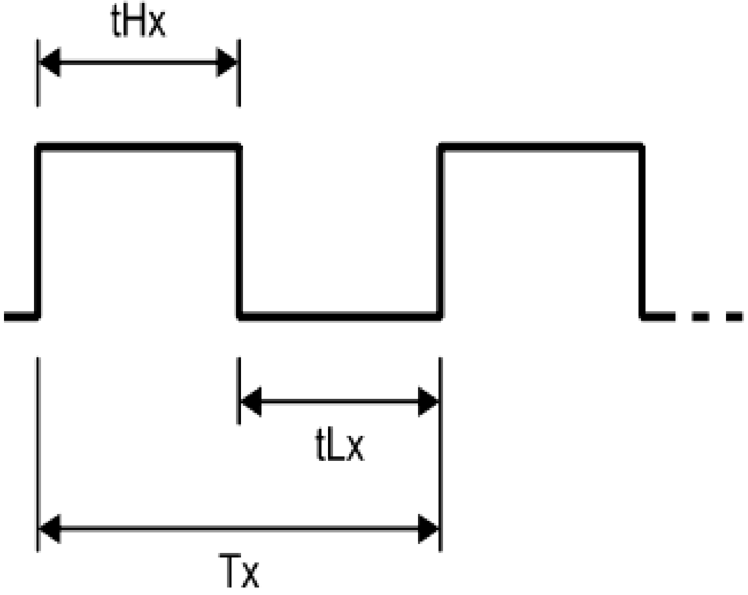
Accelerometer

- Senses orientation in two axes.
- Single axis rotation, position sensing.
- Detects collision (vibration, motion).
- Outputs in PWM.

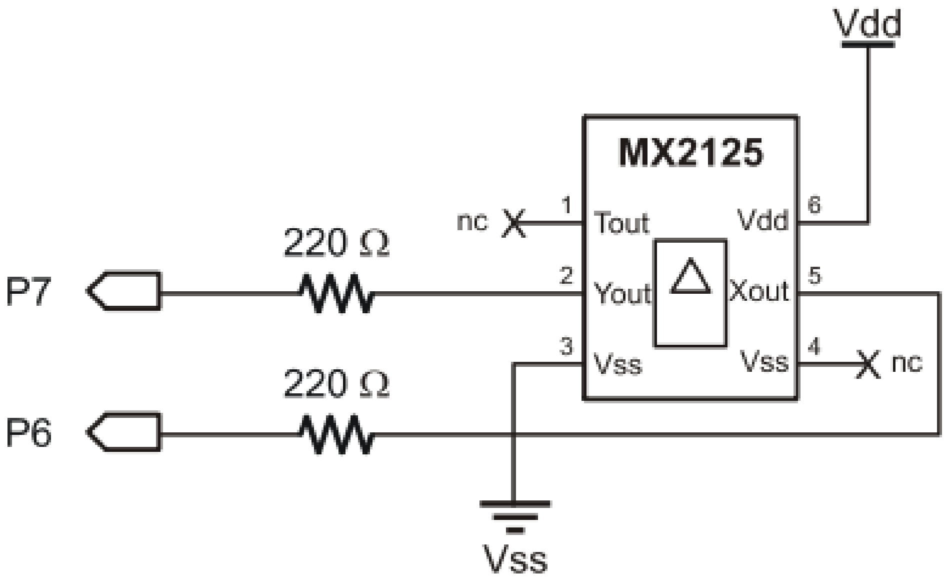


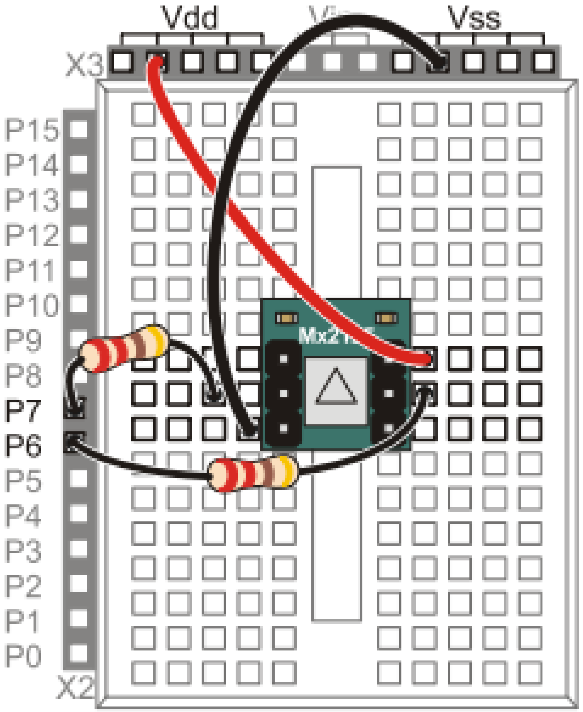
Memsic 2125 Operation

- Acceleration proportional to tHx / Tx.
- Frequency is 100Hz.
- Tx almost fixed at 1/100 s = 10ms.
- Measure tHx to know acceleration about an axis.
- At 50%, i.e. 10/2 ms = 5ms, corresponds to 0 deg.
- Use PULSIN instruction to read tHx.



Memsic 2125 Connection





P6 reads the tilt around X-axis.
P7 reads the tilt around Y-axis.

Memsic 2125 Programming

- PULSIN *pin, state, variable*
  - *pin* is the input PIN to read
  - *state* is the state to measure the width, (1 or 0)
  - *variable* where the pulse width is stored
- 1 is 2us.
  - 0 deg is 5ms = 2500

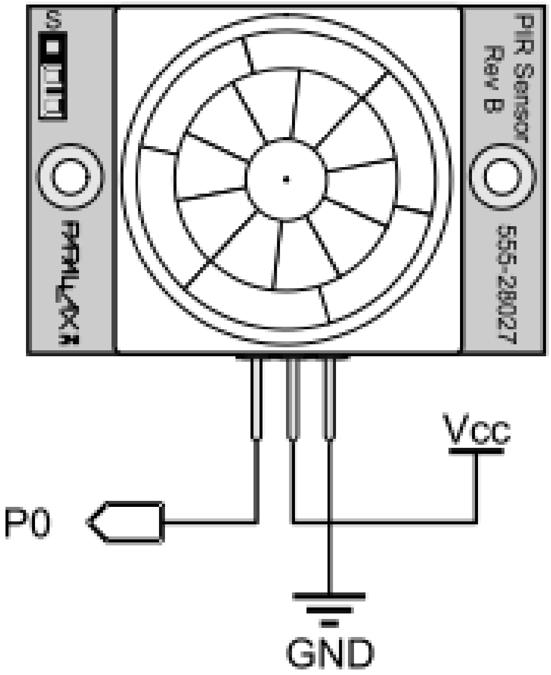
- 0 deg = 2500,
- 90 deg = ?
- -90 deg =?

```
x VAR Word
y VAR Word

DEBUG CLS

DO
  PULSIN 6, 1, x   ‘read x-axis tilt
  PULSIN 7, 1, y   ‘read y-axis tilt
  DEBUG HOME, DEC4 ? x, DEC4 ? y
  PAUSE 100
LOOP
```

PIR Sensor

- Senses motion by changes in IR.
- Animals, including human, emit IR due to their body temperature.
- PIR can detect presence of human, or animals.

PIR Sensor Connection & Programming



```
PAUSE 40000 ' PIR warm-up time

DO
   DEBUG HOME, BIN1 IN0 ' Display state of P0
   PAUSE 100 ' Small Delay
LOOP ' Repeat Forever
```

P0 = 1 if movement detected
P0 = 0 if no movement detected

Program Blocks: Subroutine

- **Subroutines**, also called **methods** or **functions**, are block of program codes that can be easily reused in the main part of a program.
  - We can get things done without understanding the details in the Subroutine.
  - Reduce the number of repeated codes.

```
x VAR Word

'We want to do this two times
'Assume we can't use FOR loop
DEBUG "Enter a number: "
DEBUGIN DEC1 x
IF ( x < 5 ) THEN
  DEBUG "Less.", CR
ELSE
  DEBUG "Equal or more.", CR
ENDIF

'Second time
DEBUG "Enter a number: "
DEBUGIN DEC1 x
IF ( x < 5 ) THEN
  DEBUG "Less.", CR
ELSE
  DEBUG "Equal or more.", CR
ENDIF

DEBUG "Program ended."
```

```
x VAR Word

'We want to do this two times
'We call the subroutine two times using GOSUB
GOSUB A_subroutine
GOSUB A_subroutine

DEBUG "Program ended."
END ' Prevent program continue downward

'A subroutine: write once, use many times
    DEBUG "Enter a number: "
    DEBUGIN DEC1 x
    IF ( x < 5 ) THEN
      DEBUG "Less.", CR
    ELSE
      DEBUG "Equal or more.", CR
    ENDIF
    RETURN 'Always end with a RETURN
```

Imagine the benefit of GOSUB when you have to call it many times at different point in the program.

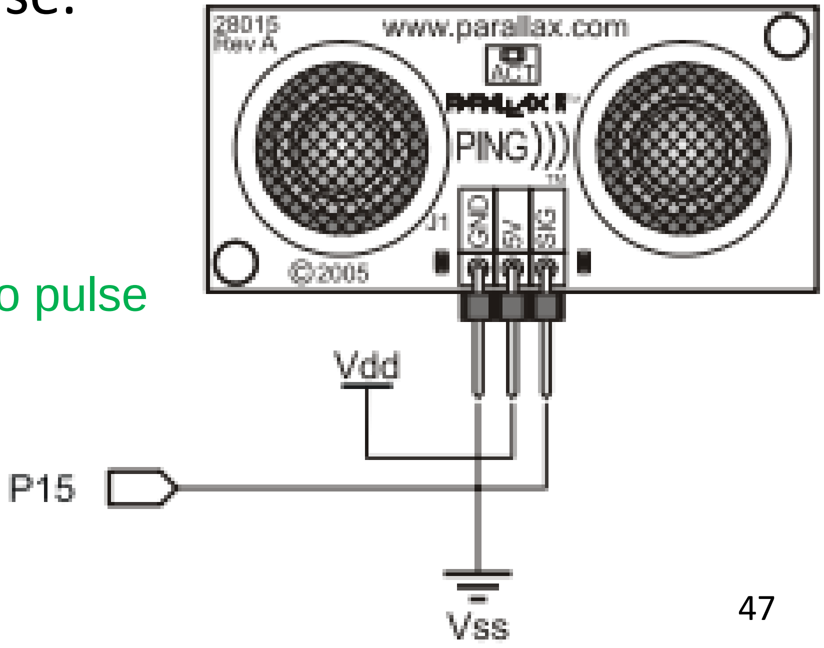
PING Sensor

- Detect distance of the object (that reflect the sonar wave) in front.
- PING Sensors are used to detect object in front.
  - Hard object reflect more sonar wave.
  - Gives distance.

PING Sensor Connection & Programming

- Make use of available subroutine to read the signal and then convert into inches or cm.
  - Black box approach: you don’t have to understand every lines of code.
  - But, you must know how to use.

```
Get_Sonar:
  Ping = IsLow ' make trigger 0-1-0
  PULSOUT Ping, Trigger ' activate sensor
  PULSIN Ping, IsHigh, rawDist ' measure echo pulse
  rawDist = rawDist */ Scale ' convert to uS
  rawDist = rawDist / 2 ' remove return trip
  RETURN
```



```
' -----[ I/O Definitions ]-------------------------------------------------
Ping PIN 15

' -----[ Constants ]-------------------------------------------------------
Trigger CON 5 ' trigger pulse = 10 uS
Scale CON $200 ' raw x 2.00 = uS
RawToIn CON 889 ' 1 / 73.746 (with **)
RawToCm CON 2257 ' 1 / 29.034 (with **)
IsHigh CON 1 ' for PULSOUT
IsLow CON 0

' -----[ Variables ]-------------------------------------------------------
rawDist VAR Word ' raw measurement
inches VAR Word
cm VAR Word

DO
  GOSUB Get_Sonar ' get sensor value

  inches = rawDist ** RawToIn ' convert to inches
  cm = rawDist ** RawToCm ' convert to centimeters

  DEBUG HOME, "Distance = ", DEC inches, " inches, ", DEC cm, " cm."
  PAUSE 100
LOOP
END
```

Gripper

- Control a standard servo to open (release) and close (grip) the gripper.

```
‘ Move servo horn to one end and adjust
the mechanism to fully open the gripper
DO
  PULSOUT 14, 250 ‘ to release
  PAUSE 20
LOOP

‘ Move servo horn to the other end and
adjust the mechanism to fully close the
gripper
DO
  PULSOUT 14, 1200 ‘ to grip
  PAUSE 20
LOOP
```

Quick Sum Up

- You tried on a number of sensors:
  - Keyboard (DEBUGIN), push button switch, limit switch, LDR, IR reflective sensors
  - They will become handy in making a responsive robot.
- You learned a few more concepts in controlling the flow of your program (intelligence):
  - IF … ELSE … ENDIF, LOOP UNTIL.
  - There are more ways.
- You get to know a few advance parts:
  - Sensors: accelerometer, PIR sensor, PING sensor.
  - Actuator/effector: gripper.