# Sensors: Sensing and Data Acquisition

Prof. Yan Luo

*For UMass Lowell 16.480/552*

# Outline

- Sensors
- Sensor interfacing
- Sensor data conversion and acquisition
- PIC microcontroller programming
- Lab 1: Sensor design and data acquisition (a light intensity sensor)

# Basic Principle of Sensors

- Transducer: a device that converts energy from one form to another
- Sensor: converts a physical parameter to an electric output
  - Electric output is desirable as it enables further signal processing.
- Actuator: coverts an electric signal to a physical output

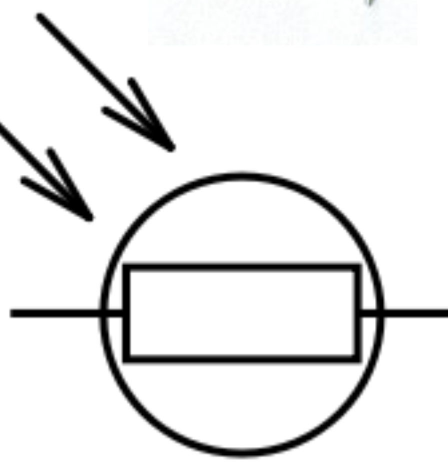
# Sensors

- Cameras
- Accelerometer
- Rate gyro
- Strain gauge
- Microphone
- Magnetometer
- Chemical sensors
- Optical sensors

- Analog sensors
  - Continuously varying output
- Digital sensors
  - on/off
  - Pulse trains (freq convey measurement)

# Example: Photoresistor

- Or Light Dependent Resistor (LDR)
  - Resistance decreases with increasing light intensity
  - Made of semiconductor
  - Photons absorbed cause electrons to jump into conduction band

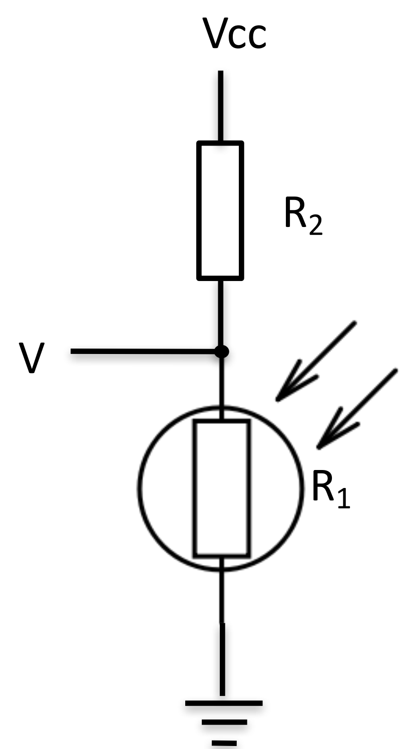


# Interfacing with Sensors

- Interface circuitry
- ADC
- Interfaces of the embedded system
- Software drivers and APIs

# Example voltage divider circuit

$V = Vcc \times R_1/(R_1+R_2)$



# Analog-Digital Converter (ADC)

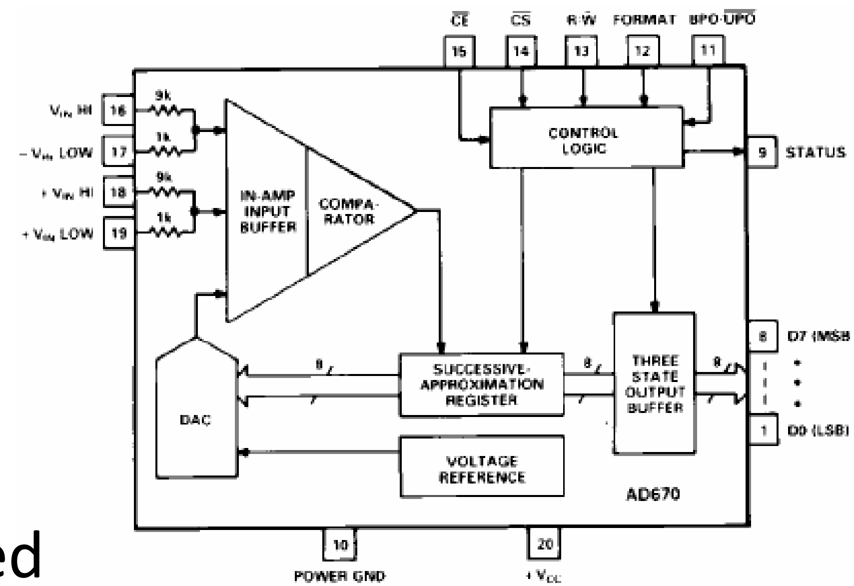
- Types of ADC
  - Integrating ADC
    - Internal voltage controlled oscillator
    - slow
  - Successive approximation ADC
    - Digital code driving the analog reference voltage
  - Flash ADC
    - A bank of comparators
    - Fast

# ADC Characteristics

- Sample rate
  - Samples per second (SPS)
  - The faster, the more expensive
  - Nyquist frequency
    - Double (or more) than the signal's highest frequency component.
- Resolution
  - Accuracy of each sample (8-bit, 11-bit etc)
  - High resolution is not always required
    - E.g. temperature sensor $0^oC$-$100^oC$ with accuracy of +/-$0.5^oC$, has only 200 meaningful voltage levels
- Calculation of original analog voltage
  - Signal = (sample / max_value) * reference_voltage

    = (153/255) * 5 = 3 volts

# Example: AD670




- 20-pin DIP
- 8-bit ADC
- Microprocessor bus interface
- 10us conversion speed
- Convenient input range
- No external components required

Internals

- Instrumentation amplifier, DAC, comparator, successive approximation register (SAR), three-state output buffer, etc.

# AD670

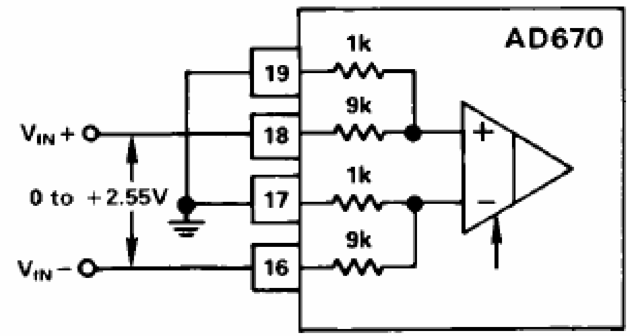
- Main control pins
  - R/W*
    - 0 – start conversion
    - 1 – read data
  - CS*
  - CE*
  - STATUS
    - 1 – indicating a conversion is in process
- Data
  - 8-bit
  - High impedance unless R/W* = 1, CS* = CE* = 0
  - Read cycle ends when either CS* or CE* brought high
- Conversion
  - Any convert start commands ignored until current conversion ends
  - Conversion cycle cannot be stopped or restarted

# Input/output of AD670

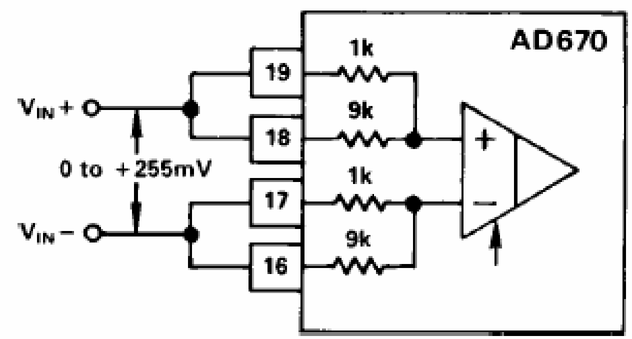
- Two input formats
  - Bipolar
  - Unipolar
  - Controlled by BPO/UPO* pin
- Four input spans
  - 0 ~ 2.55V
  - 0mv ~ 255mV
  - -1.28V ~ 1.27V
  - -128mV ~ 127mV
- Output format (FORMAT pin)
  - Two's complement
  - Offset binary

| BPO/$\overline{\text{UPO}}$ | FORMAT | INPUT RANGE/ OUTPUT FORMAT |
|---|---|---|
| 0 | 0 | Unipolar/Straight Binary |
| 1 | 0 | Bipolar/Offset Binary |
| 0 | 1 | Unipolar/2s Complement |
| 1 | 1 | Bipolar/2s Complement |

# Input connections




*2a. 0 V to 2.55 V (10 mV/LSB)*




*2b. 0 mV to 255 mV (1 mV/LSB)*

**NOTE: PIN 11, BPO/$\overline{\text{UPO}}$ SHOULD BE LOW WHEN CONVERSION IS STARTED.**

# Output Codes

| $+V_{IN}$ | $-V_{IN}$ | DIFF $V_{IN}$ | STRAIGHT BINARY (FORMAT = 0, BPO/$\overline{UPO}$ = 0) |
|---|---|---|---|
| 0 | 0 | 0 | 0000 0000 |
| 128 mV | 0 | 128 mV | 1000 0000 |
| 255 mV | 0 | 255 mV | 1111 1111 |
| 255 mV | 255 mV | 0 | 0000 0000 |
| 128 mV | 127 mV | 1 mV | 0000 0001 |
| 128 mV | –127 mV | 255 mV | 1111 1111 |

*Figure 5a. Unipolar Output Codes (Low Range)*

| $+V_{IN}$ | $-V_{IN}$ | DIFF $V_{IN}$ | OFFSET BINARY (FORMAT = 0, BPO/$\overline{UPO}$ = 1) | 2s COMPLEMENT (FORMAT = 1, BPO/$\overline{UPO}$ = 1) |
|---|---|---|---|---|
| 0 | 0 | 0 | 1000 0000 | 0000 0000 |
| 127 mV | 0 | 127 mV | 1111 1111 | 0111 1111 |
| 1.127 V | 1.000 V | 127 mV | 1111 1111 | 0111 1111 |
| 255 mV | 255 mV | 0 | 1000 0000 | 0000 0000 |
| 128 mV | 127 mV | 1 mV | 1000 0001 | 0000 0001 |
| 127 mV | 128 mV | –1 mV | 0111 1111 | 1111 1111 |
| 127 mV | 255 mV | –128 mV | 0000 0000 | 1000 0000 |
| –128 mV | 0 | –128 mV | 0000 0000 | 1000 0000 |

*Figure 5b. Bipolar Output Codes (Low Range)*

# Control Logic

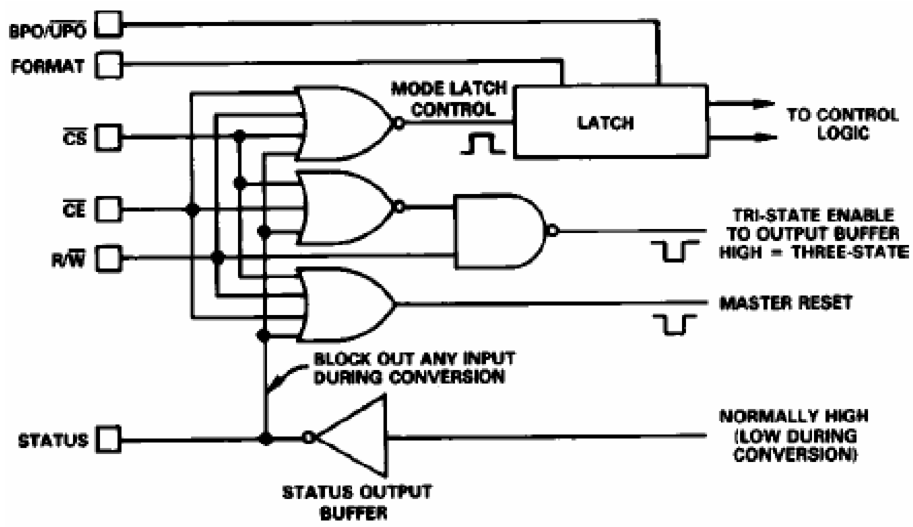



Table II. AD670 Control Signal Truth Table

| R/$\overline{W}$ | $\overline{CS}$ | $\overline{CE}$ | OPERATION |
|---|---|---|---|
| 0 | 0 | 0 | WRITE/CONVERT |
| 1 | 0 | 0 | READ |
| X | X | 1 | NONE |
| X | 1 | X | NONE |

# Timing

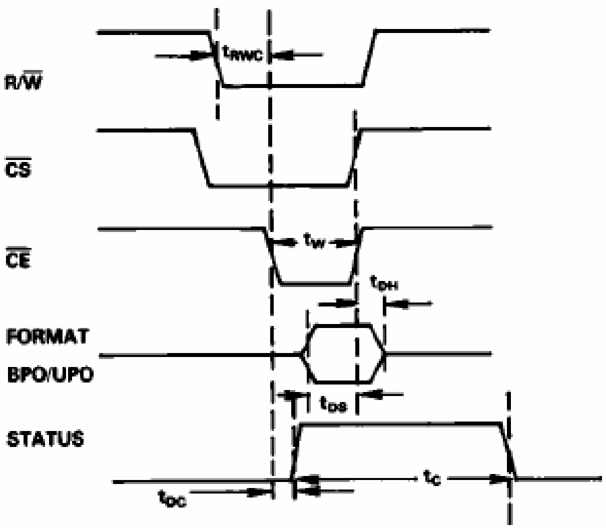



*Figure 8. Write/Convert Start Timing*

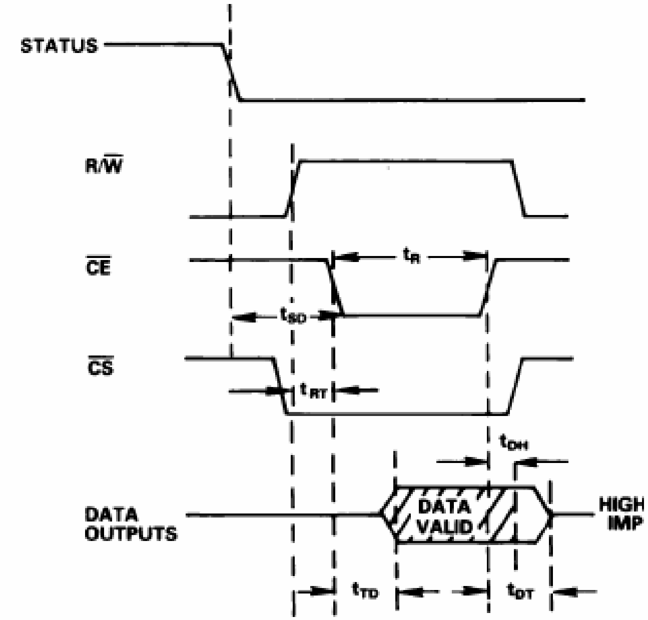



*Figure 9. Read Cycle Timing*

**Table III. AD670 TIMING SPECIFICATIONS**

| Symbol | Parameter | @ +25°C Min | Typ | Max | Units |
|---|---|---|---|---|---|
| **WRITE/CONVERT START MODE** | | | | | |
| $t_W$ | Write/Start Pulse Width | 300 | | | ns |
| $t_{DS}$ | Input Data Setup Time | 200 | | | ns |
| $t_{DH}$ | Input Data Hold | 10 | | | ns |
| $t_{RWC}$ | Read/Write Setup Before Control | 0 | | | ns |
| $t_{DC}$ | Delay to Convert Start | | | 700 | ns |
| $t_C$ | **Conversion Time** | | | **10** | µs |
| **READ MODE** | | | | | |
| $t_R$ | Read Time | 250 | | | ns |
| $t_{SD}$ | Delay from Status Low to Data Read | | | 250 | ns |
| $t_{TD}$ | **Bus Access Time** | | 200 | 250 | ns |
| $t_{DH}$ | Data Hold Time | 25 | | | ns |
| $t_{DT}$ | **Output Float Delay** | | | **150** | ns |
| $t_{RT}$ | $R/\overline{W}$ before $\overline{CE}$ or $\overline{CS}$ low | 0 | | | ns |

# Interfacing AD670 with Microprocessor

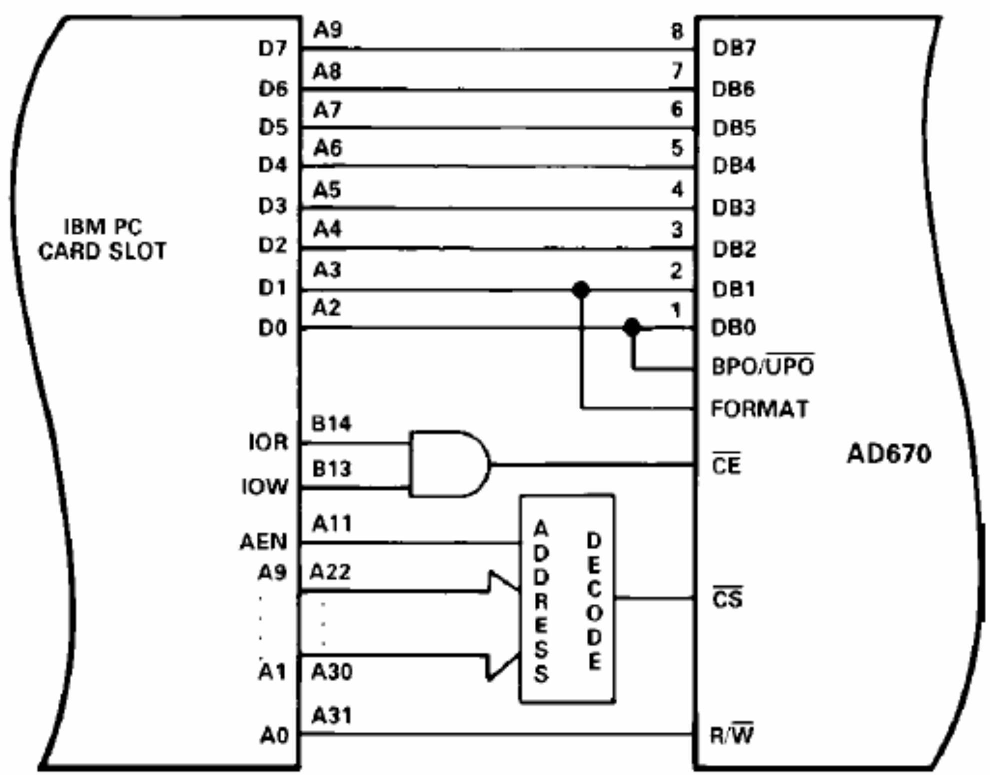


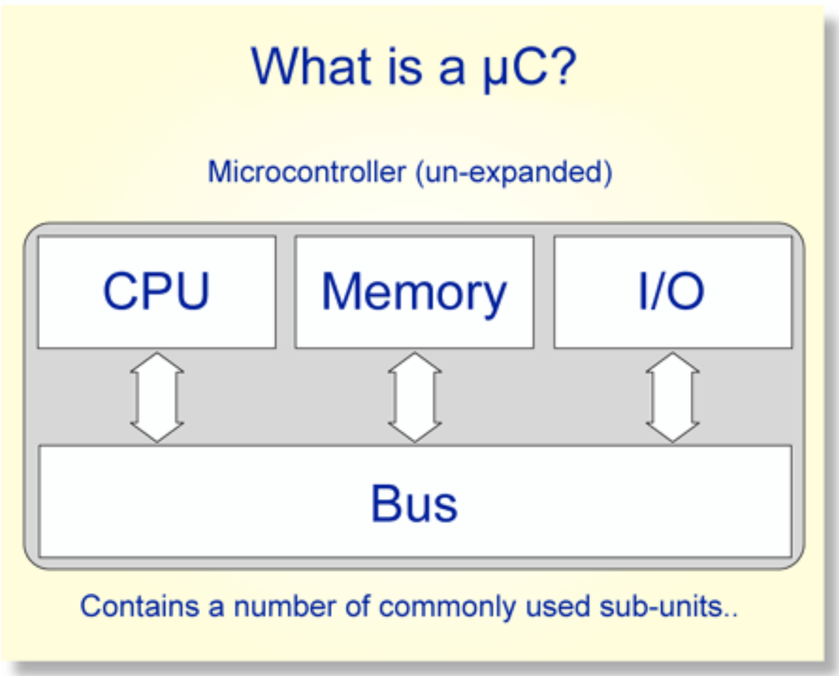

Figure 18. IBM PC Interface to AD670

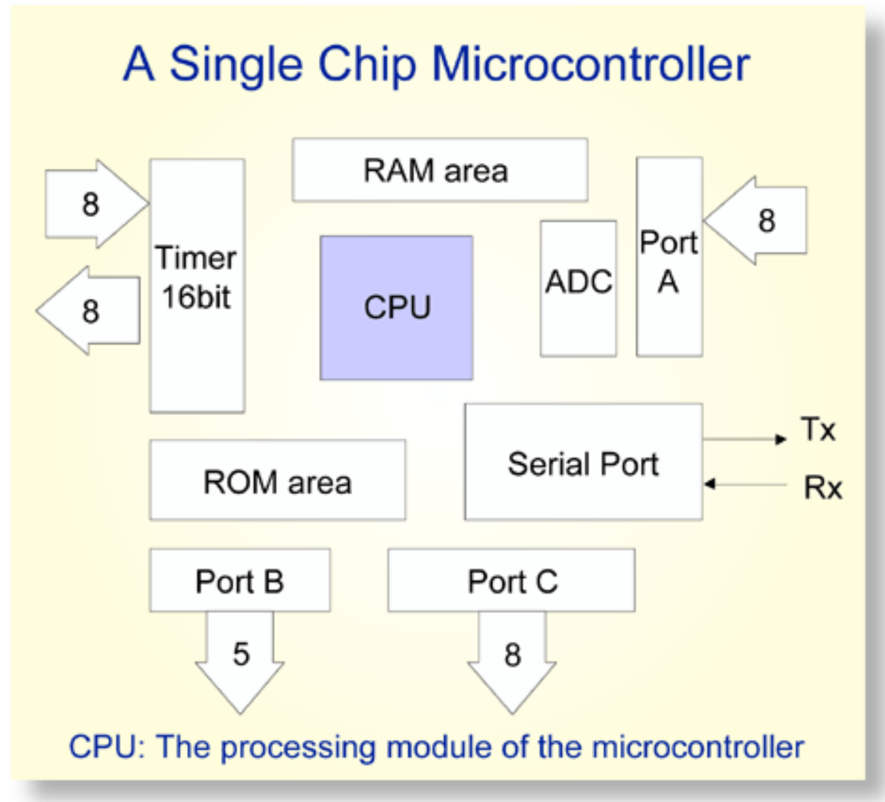
| Data | Input Format | Output Coding |
|---|---|---|
| 0 | Unipolar | Straight Binary |
| 1 | Bipolar | Offset Binary |
| 2 | Unipolar | 2s Complement |
| 3 | Bipolar | 2s Complement |

# Microcontrollers

- Low cost, low power
- System-on-chip
  - Small-size on-chip memory
  - variety of I/O (analog, digital)
  - ADC, timer, UART, USB, etc.
- Low interaction environment
  - sensor

# Architecture Overview of Microcontrollers



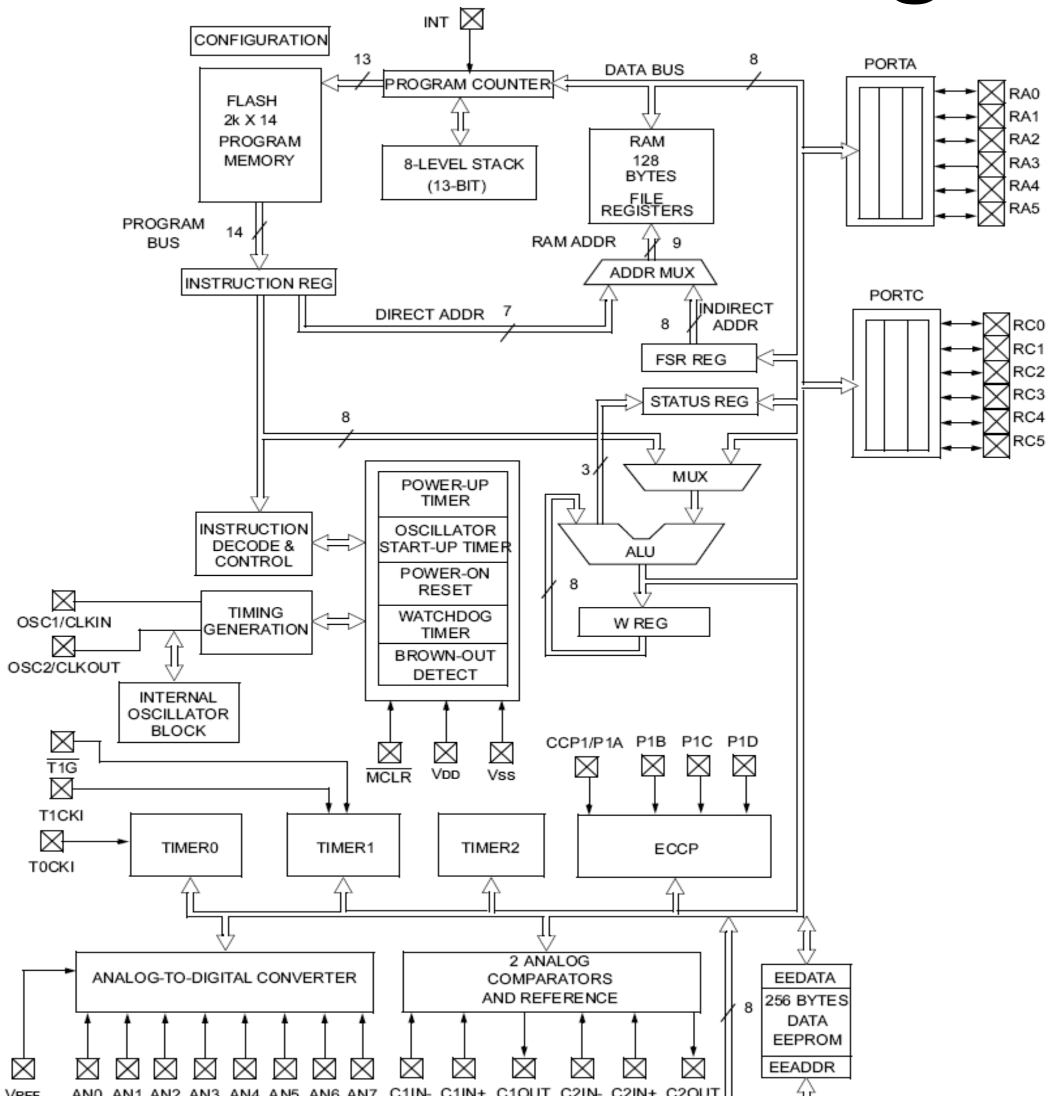

- Basically, a microcontroller is a device which integrates a number of the components of a microprocessor system onto a single microchip.

Reference: http://mic.unn.ac.uk/miclearning/modules/micros/ch1/micro01notes.html#1.4

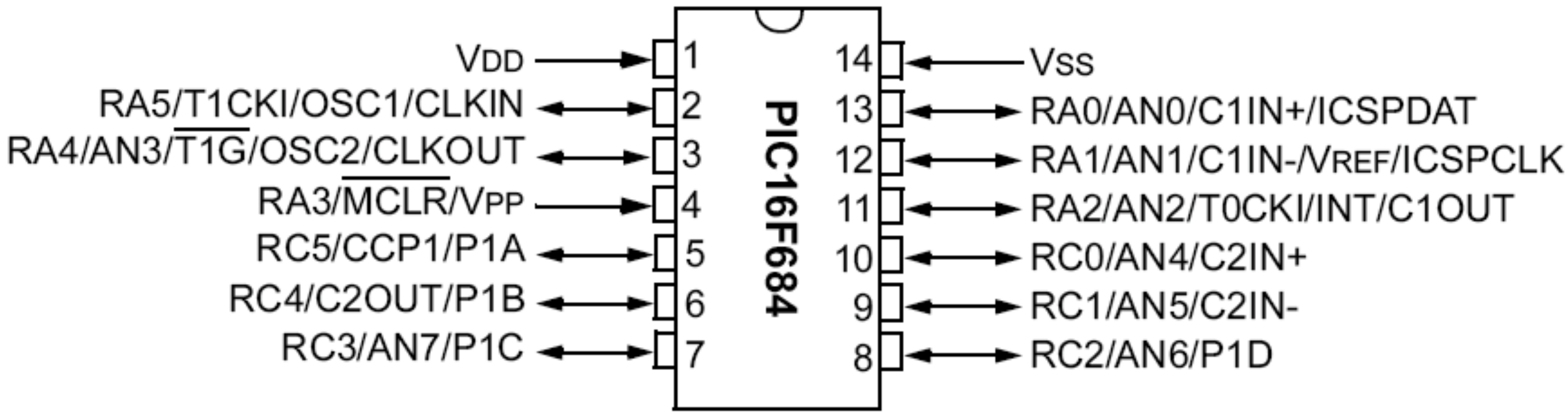
# PIC Microcontroller (PIC16F684)

- High performance, low cost, for embedded applications
  - Only 35 different instructions
  - Interrupt capability
  - Direct, indirect, relative addressing mode
- Low Power
  - 8.5uA @ 32KHz, 2.0V
- Peripheral Features
  - 12 I/O pins with individual direction control
  - 10-bit A/D converter
  - 8/16-bit timer/counter
- Special Microcontroller Features
  - Internal/external oscillator
  - Power saving sleep mode
  - High Endurance Flash/EEPROM cell
  - Etc.

# PIC16F684 Block Diagram



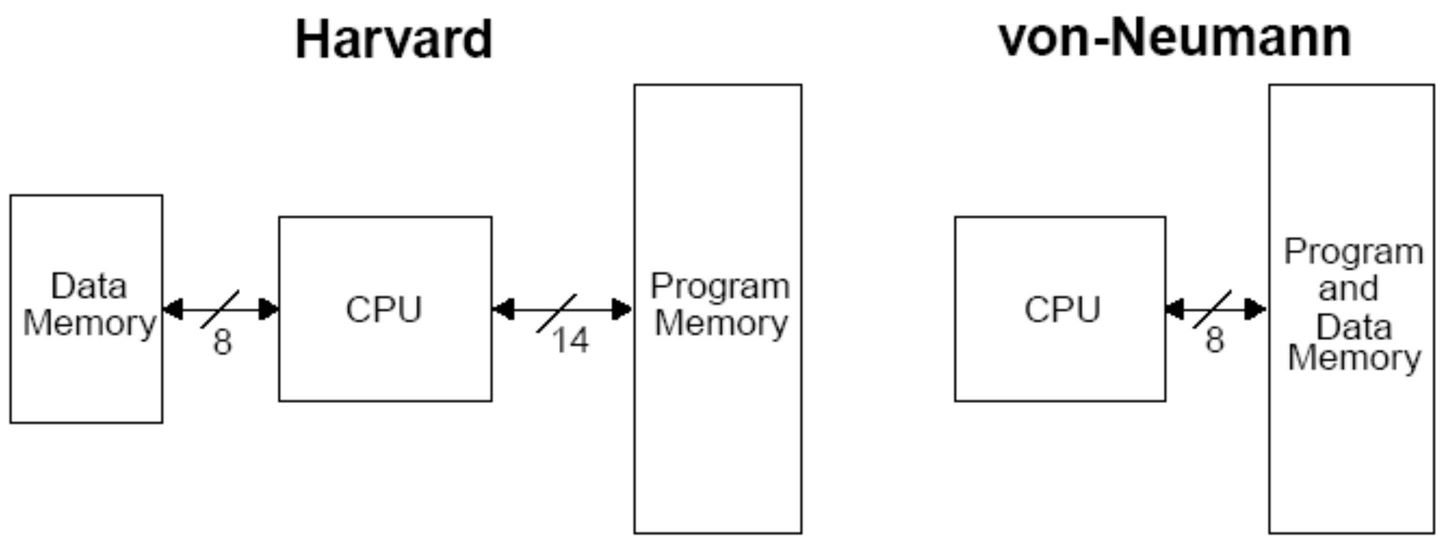
# PIC16F684




- 12 pins, 2048 instructions, 128 byte variable memory, ADC, comparator, Timers, ICD

# Harvard vs Von Neumann

- Organization of program and data memory



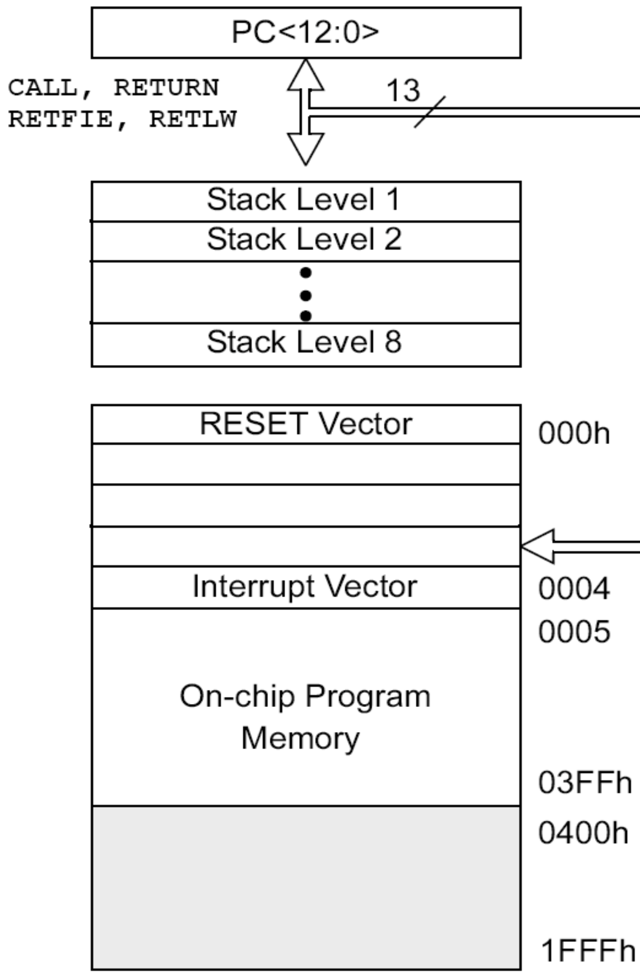
# Instruction Pipelining

- It takes one cycle to fetch the instruction and another cycle to decode and execute the instruction
- Each instruction effectively executes in one cycle
- An instruction that causes the PC to change requires two cycles.

| | TCY0 | TCY1 | TCY2 | TCY3 | TCY4 | TCY5 |
|---|---|---|---|---|---|---|
| 1. MOVLW 55h | Fetch 1 | Execute 1 | | | | |
| 2. MOVWF PORTB | | Fetch 2 | Execute 2 | | | |
| 3. CALL SUB_1 | | | Fetch 3 | Execute 3 | | |
| 4. BSF PORTA, BIT3 (Forced NOP) | | | | Fetch 4 | Flush | |
| 5. Instruction @ address SUB_1 | | | | | Fetch SUB_1 | Execute SUB_1 |
| | | | | | | Fetch SUB_1 + 1 |

All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

# Program Memory Space




- 13-bit program counter to address 8K locations (only first 2K is implemented in 16F684)
- PIC16F675 has only 1K (x14 bit) program memory - the upper bit is simply ignored during fetches from program memory
- Each location is 14-bit wide (instructions are 14 bits long)
- RESET vector is 0000h
  - When the CPU is reset, its PC is automatically cleared to zero.
- Interrupt Vector is 0004h
  - 0004h is automatically loaded into the program counter when an interrupt occurs

# Data Memory Space

| Bank 0 | File Address | Bank 1 | File Address |
|---|---|---|---|
| Indirect addr.[1] | 00h | Indirect addr.[1] | 80h |
| TMR0 | 01h | OPTION_REG | 81h |
| PCL | 02h | PCL | 82h |
| STATUS | 03h | STATUS | 83h |
| FSR | 04h | FSR | 84h |
| GPIO | 05h | TRISIO | 85h |
| | 06h | | 86h |
| | 07h | | 87h |
| | 08h | | 88h |
| | 09h | | 89h |
| PCLATH | 0Ah | PCLATH | 8Ah |
| INTCON | 0Bh | INTCON | 8Bh |
| PIR1 | 0Ch | PIE1 | 8Ch |
| | 0Dh | | 8Dh |
| TMR1L | 0Eh | PCON | 8Eh |
| TMR1H | 0Fh | | 8Fh |
| T1CON | 10h | OSCCAL | 90h |
| | 11h | | 91h |
| | 12h | | 92h |
| | 13h | | 93h |
| | 14h | | 94h |
| | 15h | WPU | 95h |
| | 16h | IOC | 96h |
| | 17h | | 97h |
| | 18h | | 98h |
| CMCON | 19h | VRCON | 99h |
| | 1Ah | EEDATA | 9Ah |
| | 1Bh | EEADR | 9Bh |
| | 1Ch | EECON1 | 9Ch |
| | 1Dh | EECON2[1] | 9Dh |
| ADRESH[2] | 1Eh | ADRESL[2] | 9Eh |
| ADCON0[2] | 1Fh | ANSEL[2] | 9Fh |
| General Purpose Registers 64 Bytes | 20h–5Fh | accesses 20h-5Fh | A0h–DFh |
| | 60h–7Fh | | E0h–FFh |
| Bank 0 | | Bank 1 | |

Unimplemented data memory locations, read as '0'.
1: Not a physical register.
2: PIC12F675 only.

# Two Special addresses

- Reset Vector Address (0000h)
  – When the CPU starts up from its reset state, its PC is automatically cleared to zero.
  – Assign ***goto Mainline*** instruction

```
Mainline
    call    Initial     ;Initialize everything
MainLoop
    call    Task1       ;Deal with task1
    call    Task2       ;Deal with task2
    …
    call    LoopTime        ;Force looptime to a fixed value
    goto    MainLoop        ;repeat
```

- Interrupt Vector Address (0004h)
  – 0004h is automatically loaded into the program counter when an interrupt occurs.
  – Assign ***goto IntService*** instruction there: cause the CPU to jump to the beginning of the interrupt service routine, located elsewhere in the memory space.

# Special Function Registers

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on POR, BOD |
|---|---|---|---|---|---|---|---|---|---|---|
| **Bank 0** | | | | | | | | | | |
| 00h | INDF[1] | Addressing this Location uses Contents of FSR to Address Data Memory | | | | | | | | 0000 0000 |
| 01h | TMR0 | Timer0 Module's Register | | | | | | | | xxxx xxxx |
| 02h | PCL | Program Counter's (PC) Least Significant Byte | | | | | | | | 0000 0000 |
| 03h | STATUS | IRP[2] | RP1[2] | RP0 | $\overline{\text{TO}}$ | $\overline{\text{PD}}$ | Z | DC | C | 0001 1xxx |
| 04h | FSR | Indirect Data Memory Address Pointer | | | | | | | | xxxx xxxx |
| 05h | GPIO | — | — | GPIO5 | GPIO4 | GPIO3 | GPIO2 | GPIO1 | GPIO0 | --xx xxxx |
| 0Ah | PCLATH | — | — | — | Write Buffer for Upper 5 bits of Program Counter | | | | | ---0 0000 |
| 0Bh | INTCON | GIE | PEIE | T0IE | INTE | GPIE | T0IF | INTF | GPIF | 0000 0000 |
| 0Ch | PIR1 | EEIF | ADIF | — | — | CMIF | — | — | TMR1IF | 00-- 0--0 |
| 0Eh | TMR1L | Holding Register for the Least Significant Byte of the 16-bit Timer1 | | | | | | | | xxxx xxxx |
| 0Fh | TMR1H | Holding Register for the Most Significant Byte of the 16-bit Timer1 | | | | | | | | xxxx xxxx |
| 10h | T1CON | — | TMR1GE | T1CKPS1 | T1CKPS0 | T1OSCEN | $\overline{\text{T1SYNC}}$ | TMR1CS | TMR1ON | -000 0000 |
| 19h | CMCON | — | COUT | — | CINV | CIS | CM2 | CM1 | CM0 | -0-0 0000 |
| 1Eh | ADRESH[3] | Most Significant 8 bits of the Left Shifted A/D Result or 2 bits of the Right Shifted Result | | | | | | | | xxxx xxxx |
| 1Fh | ADCON0[3] | ADFM | VCFG | — | — | CHS1 | CHS0 | GO/$\overline{\text{DONE}}$ | ADON | 00-- 0000 |

# Status Register

| Reserved | Reserved | R/W-0 | R-1 | R-1 | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| IRP | RP1 | RP0 | $\overline{\text{TO}}$ | $\overline{\text{PD}}$ | Z | DC | C |
| bit 7 | | | | | | | bit 0 |

bit 7 **IRP:** This bit is reserved and should be maintained as '0'

bit 6 **RP1:** This bit is reserved and should be maintained as '0'

bit 5 **RP0:** Register Bank Select bit (used for direct addressing)
0 = Bank 0 (00h - 7Fh)
1 = Bank 1 (80h - FFh)

bit 4 **$\overline{\text{TO}}$:** Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3 **$\overline{\text{PD}}$**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

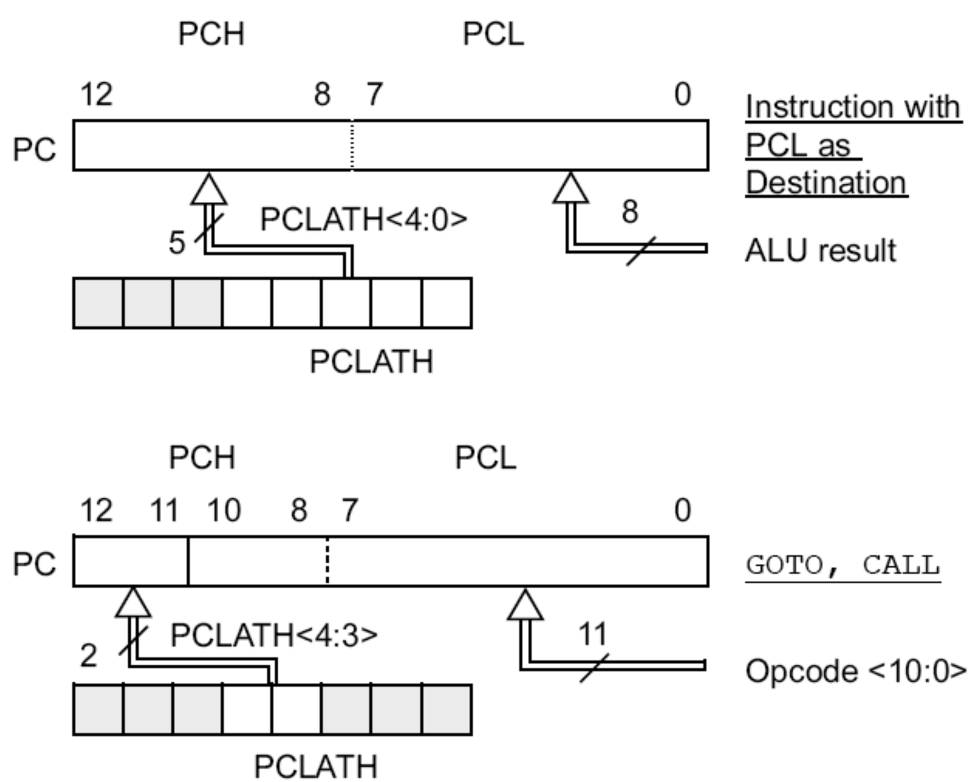
bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
For borrow, the polarity is reversed.
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

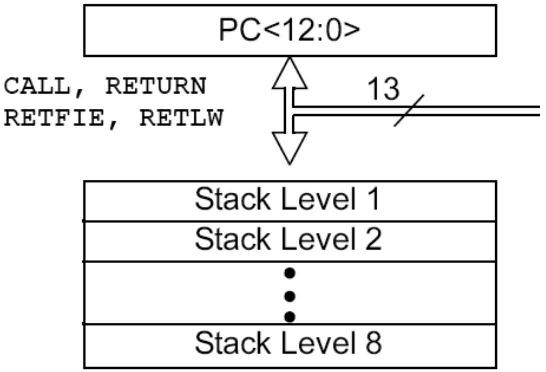
**Note:** For $\overline{\text{borrow}}$ the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register

# PCL and PCLATH




- PC: Program Counter, 13 bits
- PCL (02h): 8 bits, the lower 8 bits of PC
- PCLATH (0Ah): PC Latch, provides the upper 5 (or 2) bits of PC when PCL is written to
- 1st example: PC is loaded by writing to PCL
- 2nd example: PC is loaded during a CALL or GOTO instruction
- PCLATH is used to for jumping across 2K pages (thus, not needed in PIC16F684)

# Stack




- 8-level deep x 13-bit wide hardware stack
- The stack space is not part of either program or data space and the stackpointer is not readable or writable.
- The PC is "PUSHed" onto the stack when a CALL instruction is executed, or an interrupt causes a branch.
- The stack is "POPed" in the event of a RETURN, RETLW or a RETFIE instruction execution.
- However, NO PUSH or POP instructions !
- PCLATH is not affected by a "PUSH" or "POP" operation.
- The stack operates as a circular buffer:
  - after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push.

# Data Memory Map

- Data memory consists of
  - Special Function Registers (SFR) area
  - General Purpose Registers (GPR) area
- SFRs control the operation of the device
- GPRs are area for data storage and scratch pad operations
- GPRs are at higher address than SFRs in a bank
- Different PIC microcontrollers may have different number of GPRs

| BANK 0 | FILE ADDRESS | BANK 1 | FILE ADDRESS |
|---|---|---|---|
| INDIRECT ADDR.[1] | 00H | INDIRECT ADDR.[1] | 80H |
| TMR0 | 01H | OPTION_REG | 81H |
| PCL | 02H | PCL | 82H |
| STATUS | 03H | STATUS | 83H |
| FSR | 04H | FSR | 84H |
| PORTA | 05H | TRISA | 85H |
| | 06H | | 86H |
| PORTC | 07H | TRISC | 87H |
| | 08H | | 88H |
| | 09H | | 89H |
| PCLATH | 0AH | PCLATH | 8AH |
| INTCON | 0BH | INTCON | 8BH |
| PIR1 | 0CH | PIE1 | 8CH |
| | 0DH | | 8DH |
| TMR1L | 0EH | PCON | 8EH |
| TMR1H | 0FH | OSCCON | 8FH |
| T1CON | 10H | OSCTUNE | 90H |
| TMR2 | 11H | ANSEL | 91H |
| T2CON | 12H | PIR2 | 92H |
| CCPR1L | 13H | | 93H |
| CCPR1H | 14H | | 94H |
| CCP1CON | 15H | WPUA | 95H |
| PWM1CON | 16H | IOCA | 96H |
| ECCPAS | 17H | | 97H |
| WDTCON | 18H | | 98H |
| CMCON0 | 19H | VRCON | 99H |
| CMCON1 | 1AH | EEDAT | 9AH |
| | 1BH | EEADR | 9BH |
| | 1CH | EECON1 | 9CH |
| | 1DH | EECON2[1] | 9DH |
| ADRESH | 1EH | ADRESL | 9EH |
| ADCON0 | 1FH | ADCON1 | 9FH |
| GENERAL PURPOSE REGISTERS 96 BYTES | 20H | GENERAL PURPOSE REGISTERS 32 BYTES | A0H – BFH |
| | 70H | ACCESSES 70H-7FH | F0H |
| | 7FH | | FFH |

# Banking

- Data memory is partitioned into banks
- Each bank extends up to 7Fh (128) bytes
  - 4 banks : 4*128 bytes = 512 bytes
  - 2 banks : 2*128 bytes = 256 bytes
- Lower locations of each bank are reserved for SFRs. Above the SFRs are GPRs.
- Implemented as Static RAM
- Some “high use” SFRs from bank0 are mirrored in the other banks (e.g., INDF, PCL, STATUS, FSR, PCLATH, INTCON)
- RP0 and RP1 bits in the STATUS register selects the bank when using direct addressing mode.

# What are the two/four banks for?

- 14-bit instructions use 7 bits to address a location
- Memory space is organized in 128Byte banks.
- PIC 16F684 has two banks - Bank 0 and Bank 1.
- Bank 1 is used to control the actual operation of the PIC
  - for example to tell the PIC which bits of Port A are input and which are output.
- Bank 0 is used to manipulate the data.
- An example is as follows: Let us say we want to make one bit on Port A high.
  - First we need to go to Bank 1 to set the particular bit, or pin, on Port A as an output.
  - We then come back to Bank 0 and send a logic 1 (bit 1) to that pin.

# Special Function Registers (1)

- W, the working register
  - To move values from one register to another register, the value must pass through the W register.
- FSR (04h,84h,104h,184h), File Select Register
  - Indirect data memory addressing pointer
- INDF (00h,80h,100h,180h)
  - accessing INDF accesses the location pointed by IRP+FSR
- PC, the Program Counter, PCL (02h, 82h, 102h, 182h) and PCLATH (0Ah, 8Ah, 10Ah, 18Ah)

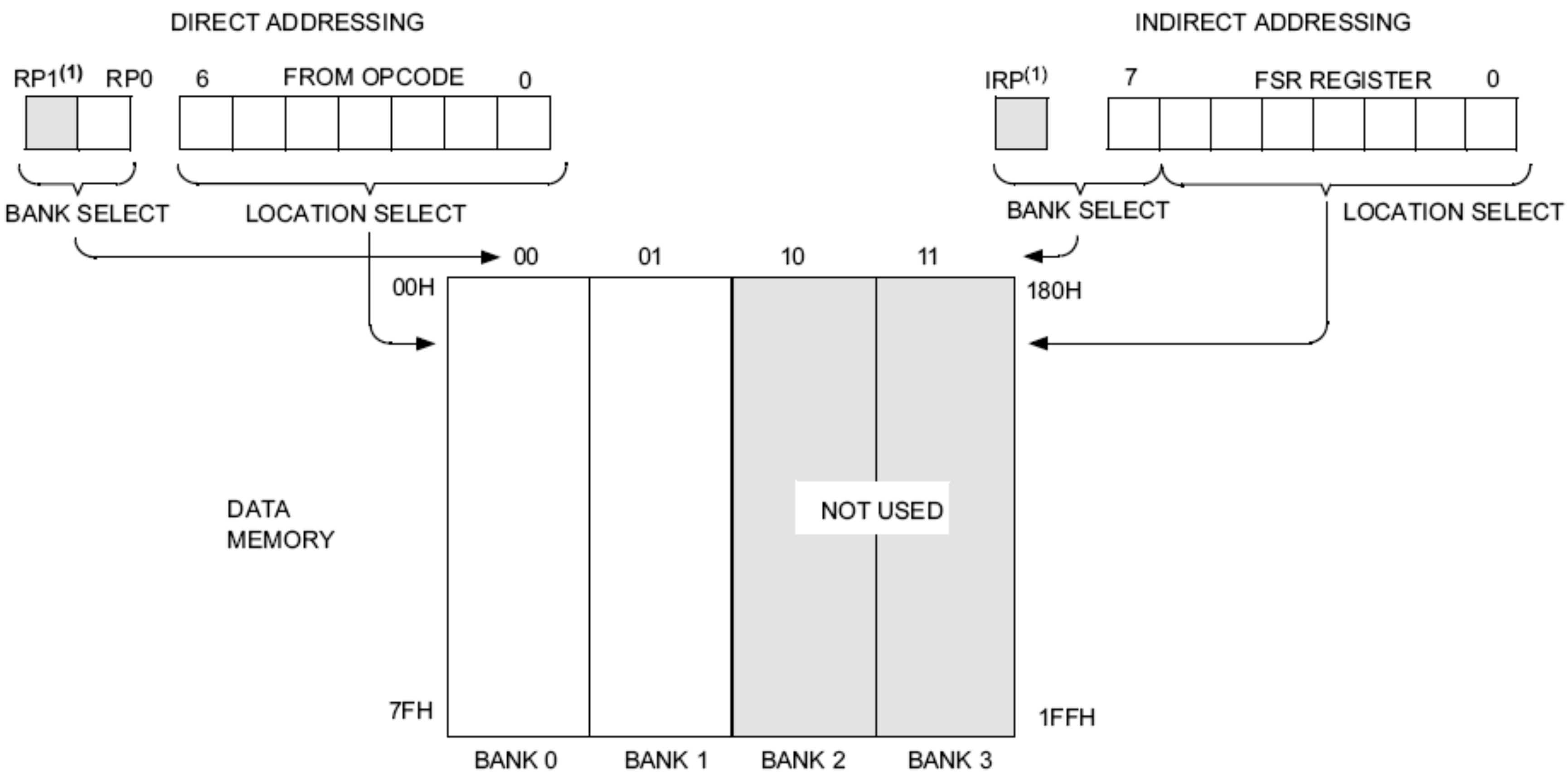
# Special Function Registers (2)

- STATUS (03h, 83h, 103h, 183h)

| R/W-0 | R/W-0 | R/W-0 | R-1 | R-1 | R/W-x | R/W-x | R/W-x |
|---|---|---|---|---|---|---|---|
| IRP | RP1 | RP0 | $\overline{TO}$ | $\overline{PD}$ | Z | DC | C |
| bit 7 | | | | | | | bit 0 |

  - IRP: Register bank select bit (indirect addressing)
  - RP1:RP0 – Register bank select bits (direct addressing)
  - NOT_TO: Time Out bit, reset status bit
  - NOT_PD: Powel-Down bit, reset status bit
  - Z: Zero bit ~ ZF in x86
  - DC: Digital Carry bit ~ AF in x86
  - C: Carry bit ~ CF in x86 (note: for subtraction, borrow is opposite)

# Direct/Indirect Addressing




For memory map detail see Figure 2-2.

**Note 1:** The RP1 and IRP bits are reserved; always maintain these bits clear.

# Direct Addressing

- Use only 7 bits of instruction to identify a register file address
- The other two bits of register address come from RP0 and RP1 bits in the STATUS register

| Accessed Bank | Direct (RP1:RP0) | Indirect (IRP) |
|---|---|---|
| 0 | 0 0 | 0 |
| 1 | 0 1 | |
| 2 | 1 0 | 1 |
| 3 | 1 1 | |

Example: Bank switching (Note: case of 4 banks)

```
CLRF     STATUS        ; Clear STATUS register (Bank0)
: ;
BSF      STATUS, RP0   ; Bank1
: ;
BCF      STATUS, RP0   ; Bank0
: ;
MOVLW    0x60       ; Set RP0 and RP1 in STATUS register, other
XORWF    STATUS, F           ; bits unchanged (Bank3)
: ;
BCF      STATUS, RP0   ; Bank2
: ;
BCF      STATUS, RP1   ; Bank0
```

# Indirect Addressing

- The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.
- Any instruction using the INDF register actually access the register pointed to by the File Select Register (FSR).
- The effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit in STATUS register.

Example

```
          MOVLW     0x20  ;initialize pointer
          MOVWF     FSR   ;to RAM
NEXT:          CLRF INDF  ;clear INDF register
          INCF  FSR,F ;inc pointer
          BTFSS     FSR,4 ;all done?  (to 0x2F)
          GOTO      NEXT  ;no clear next
CONTINUE
          :         ;yes continue
```

# I/O Ports

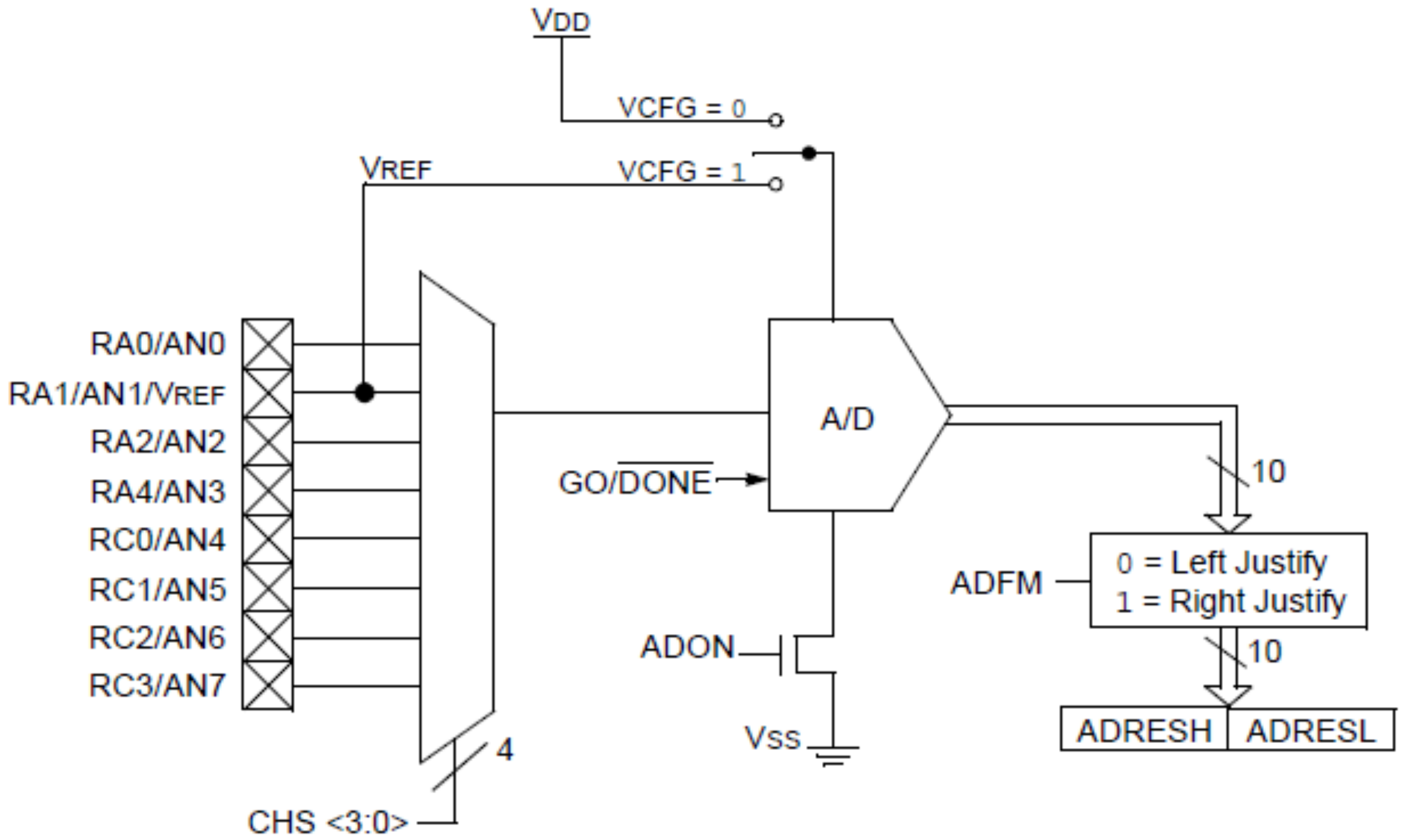
- General I/O pins are the simplest of peripherals used to monitor and control other devices.
- For most ports, the I/O pin's direction (input or output) is controlled by the data direction register **TRISx** (x=A,B,C,D,E): a '1' in the TRIS bit corresponds to that pin being an input, while a '0' corresponds to that pin being an output
- The **PORTx** register is the latch for the data to be output. Reading PORTx register read the status of the pins, whereas writing to it will write to the port latch.
- Example: Initializing PORTD (PORTD is an 8-bit port. Each pin is individually configurable as an input or output).

```
bcf     STATUS, RP0      ; bank0
bcf     STATUS, RP1
clrf    PORTD   ; initializing PORTD by clearing output data latches
bsf     STATUS, RP0   ; select bank1
movlw   0xCF    ; value used to initialize data direction
movwf   TRISD   ; PORTD<7:6>=inputs, PORTD<5:4>=outputs,
                ; PORTD<3:0>=inputs
```

# PIC based Sensor Data Acquisition

# Block Diagram of ADC in PIC16F684

- 10bit resolution
- Reference voltage software-selectable
- Can generate an interrupt upon the completion of a conversion



# ADC Configuration

- Port configuration
  - TRISx and ANSEL
- Channel selection
  - ADCON0: CHS
- ADC voltage reference selection
  - ADCON0: VCFG
- ADC conversion clock source
  - ADCON1: ADCS
- Interrupt control (optional)
- Result formatting
  - ADCON0: ADFM

# Registers related to ADC

**TABLE 9-2: SUMMARY OF ASSOCIATED ADC REGISTERS**

| Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 | Value on: POR, BOR | Value on all other Resets |
|---|---|---|---|---|---|---|---|---|---|---|
| ADCON0 | ADFM | VCFG | — | CHS2 | CHS1 | CHS0 | GO/$\overline{\text{DONE}}$ | ADON | 0000 0000 | 0000 0000 |
| ADCON1 | — | ADCS2 | ADCS1 | ADCS0 | — | — | — | — | -000 ---- | -000 ---- |
| ANSEL | ANS7 | ANS6 | ANS5 | ANS4 | ANS3 | ANS2 | ANS1 | ANS0 | 1111 1111 | 1111 1111 |
| ADRESH | A/D Result Register High Byte | | | | | | | | xxxx xxxx | uuuu uuuu |
| ADRESL | A/D Result Register Low Byte | | | | | | | | xxxx xxxx | uuuu uuuu |
| INTCON | GIE | PEIE | T0IE | INTE | RAIE | T0IF | INTF | RAIF | 0000 0000 | 0000 0000 |
| PIE1 | EEIE | ADIE | CCPIE | C2IE | C1IE | OSFIE | TMR2IE | TMR1IE | 0000 0000 | 0000 0000 |
| PIR1 | EEIF | ADIF | CCPIF | C2IF | C1IF | OSFIF | TMR2IF | TMR1IF | 0000 0000 | 0000 0000 |
| PORTA | — | — | RA5 | RA4 | RA3 | RA2 | RA1 | RA0 | --x0 x000 | --uu uuuu |
| PORTC | — | — | RC5 | RC4 | RC3 | RC2 | RC1 | RC0 | --xx 0000 | --uu uuuu |
| TRISA | — | — | TRISA5 | TRISA4 | TRISA3 | TRISA2 | TRISA1 | TRISA0 | --11 1111 | --11 1111 |
| TRISC | — | — | TRISC5 | TRISC4 | TRISC3 | TRISC2 | TRISC1 | TRISC0 | --11 1111 | --11 1111 |

**Legend:** x = unknown, u = unchanged, – = unimplemented read as '0'. Shaded cells are not used for ADC module.

# Port Configuration (I)

- TRISx controls the direction of the PORTx pins

**REGISTER 4-2: TRISA: PORTA TRI-STATE REGISTER**

| U-0 | U-0 | R/W-1 | R/W-1 | R-1 | R/W-1 | R/W-1 | R/W-1 |
|---|---|---|---|---|---|---|---|
| — | — | TRISA5 | TRISA4 | TRISA3 | TRISA2 | TRISA1 | TRISA0 |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 7-6 **Unimplemented**: Read as ‘0’

bit 5-0 **TRISA<5:0>:** PORTA Tri-State Control bit
1 = PORTA pin configured as an input (tri-stated)
0 = PORTA pin configured as an output

**Note 1:** TRISA<3> always reads ‘1’.
**2:** TRISA<5:4> always reads ‘1’ in XT, HS and LP OSC modes.

# Port Configuration (II)

- Configure the Input mode of an I/O pin to analog

**REGISTER 4-3: ANSEL: ANALOG SELECT REGISTER**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 | R/W-1 |
| ANS7 | ANS6 | ANS5 | ANS4 | ANS3 | ANS2 | ANS1 | ANS0 |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as '0' | |
| -n = Value at POR | '1' = Bit is set | '0' = Bit is cleared | x = Bit is unknown |

bit 7-0 **ANS<7:0>**: Analog Select bits

Analog select between analog or digital function on pins AN<7:0>, respectively.

1 = Analog input. Pin is assigned as analog input[(1)].

0 = Digital I/O. Pin is assigned to port or special function.

# Ch. Selection, Ref. V Selection and Format

**REGISTER 9-1: ADCON0: A/D CONTROL REGISTER 0**

| R/W-0 | R/W-0 | U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| ADFM | VCFG | — | CHS2 | CHS1 | CHS0 | GO/$\overline{\text{DONE}}$ | ADON |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

bit 7 **ADFM:** A/D Conversion Result Format Select bit
1 = Right justified
0 = Left justified

bit 6 **VCFG:** Voltage Reference bit
1 = $V_{REF}$ pin
0 = $V_{DD}$

bit 5 **Unimplemented:** Read as ‘0’

bit 4-2 **CHS<2:0>: Analog Channel Select bits**
000 = AN0
001 = AN1
010 = AN2
011 = AN3
100 = AN4
101 = AN5
110 = AN6
111 = AN7

bit 1 **GO/$\overline{\textbf{DONE}}$:** A/D Conversion Status bit
1 = A/D conversion cycle in progress. Setting this bit starts an A/D conversion cycle.
This bit is automatically cleared by hardware when the A/D conversion has completed.
0 = A/D conversion completed/not in progress

bit 0 **ADON:** ADC Enable bit
1 = ADC is enabled
0 = ADC is disabled and consumes no operating current

# Conversion Clock Source

**REGISTER 9-2: ADCON1: A/D CONTROL REGISTER 1**

| U-0 | R/W-0 | R/W-0 | R/W-0 | U-0 | U-0 | U-0 | U-0 |
|---|---|---|---|---|---|---|---|
| — | ADCS2 | ADCS1 | ADCS0 | — | — | — | — |
| bit 7 | | | | | | | bit 0 |

| **Legend:** | | | |
|---|---|---|---|
| R = Readable bit | W = Writable bit | U = Unimplemented bit, read as ‘0’ | |
| -n = Value at POR | ‘1’ = Bit is set | ‘0’ = Bit is cleared | x = Bit is unknown |

| | |
|---|---|
| bit 7 | **Unimplemented:** Read as ‘0’ |
| bit 6-4 | **ADCS<2:0>:** A/D Conversion Clock Select bits<br>000 = Fosc/2<br>001 = Fosc/8<br>010 = Fosc/32<br>x11 = FRC (clock derived from a dedicated internal oscillator = 500 kHz max)<br>100 = Fosc/4<br>101 = Fosc/16<br>110 = Fosc/64 |
| bit 3-0 | **Unimplemented:** Read as ‘0’ |

# ADC Operation

- Start
  - ADCON0:ADON <= ‘1’    ; turn on ADC module
  - ADCON0:GO/DONE* <= ‘1’   ;start conversion
  - Have to use two separate instructions
- Completion
  - ADC module clears GO/DONE* bit
  - Set ADIF flag bit
  - Update ADRESH:ADRESL with conversion result
- Termination
  - ADCON0:GO/DONE* <= ‘0’

# Review

### 9.2.6 A/D CONVERSION PROCEDURE

This is an example procedure for using the ADC to perform an Analog-to-Digital conversion:

1. Configure Port:
   - Disable pin output driver (See TRIS register)
   - Configure pin as analog
2. Configure the ADC module:
   - Select ADC conversion clock
   - Configure voltage reference
   - Select ADC input channel
   - Select result format
   - Turn on ADC module
3. Configure ADC interrupt (optional):
   - Clear ADC interrupt flag
   - Enable ADC interrupt
   - Enable peripheral interrupt
   - Enable global interrupt(1)
4. Wait the required acquisition time(2).
5. Start conversion by setting the GO/$\overline{\text{DONE}}$ bit.
6. Wait for ADC conversion to complete by one of the following:
   - Polling the GO/$\overline{\text{DONE}}$ bit
   - Waiting for the ADC interrupt (interrupts enabled)
7. Read ADC Result
8. Clear the ADC interrupt flag (required if interrupt is enabled).