Terminology

- **Hard real-time** — systems where it is absolutely imperative that responses occur within the required deadline. E.g. Flight control systems.

- **Soft real-time** — systems where deadlines are important but which will still function correctly if deadlines are occasionally missed. E.g. Data acquisition system.

- **Real real-time** — systems which are hard real-time and which the response times are very short. E.g. Missile guidance system.

- **Firm real-time** — systems which are soft real-time but in which there is no benefit from late delivery of service.
Characteristics of RTS

• Can be large and complex — vary from a few hundred lines of assembler or C to 20 million lines of Ada estimated for the Space Station Freedom

• Concurrent control of separate system components — devices operate in parallel in the real-world; better to model this parallelism by concurrent entities in the program

• Facilities to interact with special purpose hardware — need to be able to program devices in a reliable and abstract way
More Characteristics

• Extreme reliability and safe — embedded systems typically control the environment in which they operate; failure to control can result in loss of life, damage to environment or economic loss

• Guaranteed response times — we need to be able to predict with confidence the worst case response times for systems; efficiency is important but predictability is essential
Types of Real-time System

• Hard real-time
  – Absolute deadlines, careful scheduling analysis
  – Often safety critical, careful software engineering audit trail

• Soft real-time
  – Best effort; occasional missed deadlines OK
  – Eg Digital Video, but probably not Audio.
Sizes of real-time system

- Microcontroller
  - PIC, coded in assembler
- Digital signal processing
  - TMS320, programmed in C
- Factory automation
  - Compact PCI
- Large distributed
  - N M D (Star Wars)
The Microchip PIC

• See http://www.microchip.com

• A cheap and simple family of single chip microcontrollers

• Very low entry cost:
  – Free software: Microchip or http://www.gnupic.org/
  – Almost free programmer

• I don’t use them any longer: use an ARM Cortex or an Atmel AVR.
**High Performance RISC CPU Features:**
- Only 35 single word instructions to learn
- All instructions single-cycle except for program branches which are two-cycle
- Operating speed: DC - 20 MHz clock input  DC - 200 ns instruction cycle
- 1024 words of program memory
- 68 bytes of Data RAM
- 64 bytes of Data EEPROM
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 Special Function Hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt-on-change
  - Data EEPROM write complete

**Peripheral Features:**
- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler
A PIC programmer

See: http://www.techfreakz.org/oldstuff/picb.html
A PIC-based WWW server

See: http://www.mycal.net/wsweb/
Part of a PIC Frequency meter

main

; *** Initialize registers ***
  clrf     pclath
  movlw    00000000b
  movwf    intcon
  clrf     porta
  clrf     portb
  bsf      status, rp0
  movlw    00101000b
  movwf    option
  movlw    00001000b
  movwf    trisa
  movlw    00111111b
  movwf    trisb
  bcf      status, rp0
  movlw    11110000b
  andwf    porta, f
  movlw    001111111b
  andwf    portb, f

; *** Initialize variable ***
  clrf     CNT+0
  clrf     CNT+1
  clrf     CNT+2

; *** Initialize LCD ***
  call     init_LCD
  call     dsp_fixmsg

; Select program memory bank #0.
; Disabled all interrupt
; sources.
; Initialize I/O ports.

; Select bank 1.
; Pull-up PORTB, ToCS = EXT &
; set TMRO rate. (1:1)
; Set data directions.
; 1: as input.

; Select bank 0. (Default)
; Set all output ports to '0'.

; Clear counter for
; dummy display.

; Contains function set & clear screen.
; Display fixed message.
Programming a Microcontroller

• Historically usually in assembler:
  – Allows accurate timing by instruction counting
  – Makes more efficient use of the rather small program memory
  – C compilers available, but not widely used
C Help


- [Embedded C Quiz](#)