An Introduction to PIC32

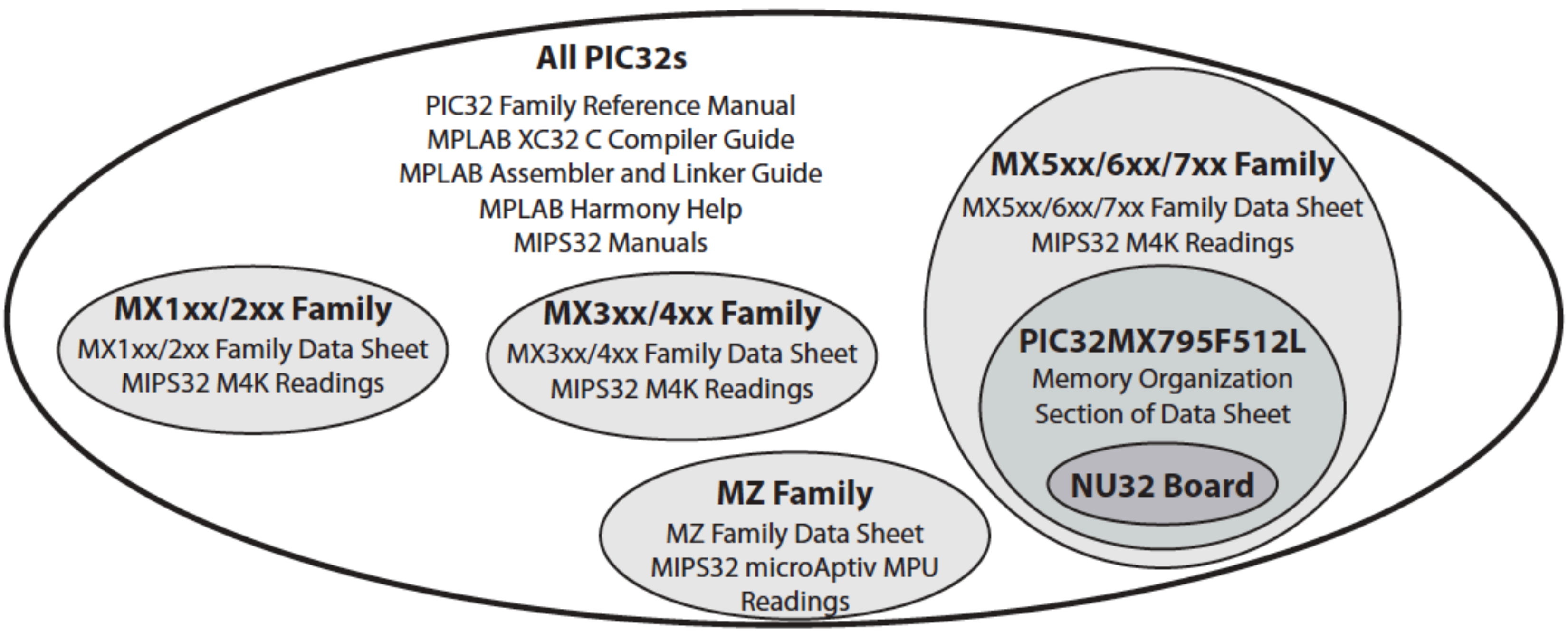
Dr. Farahmand

PIC32 Family

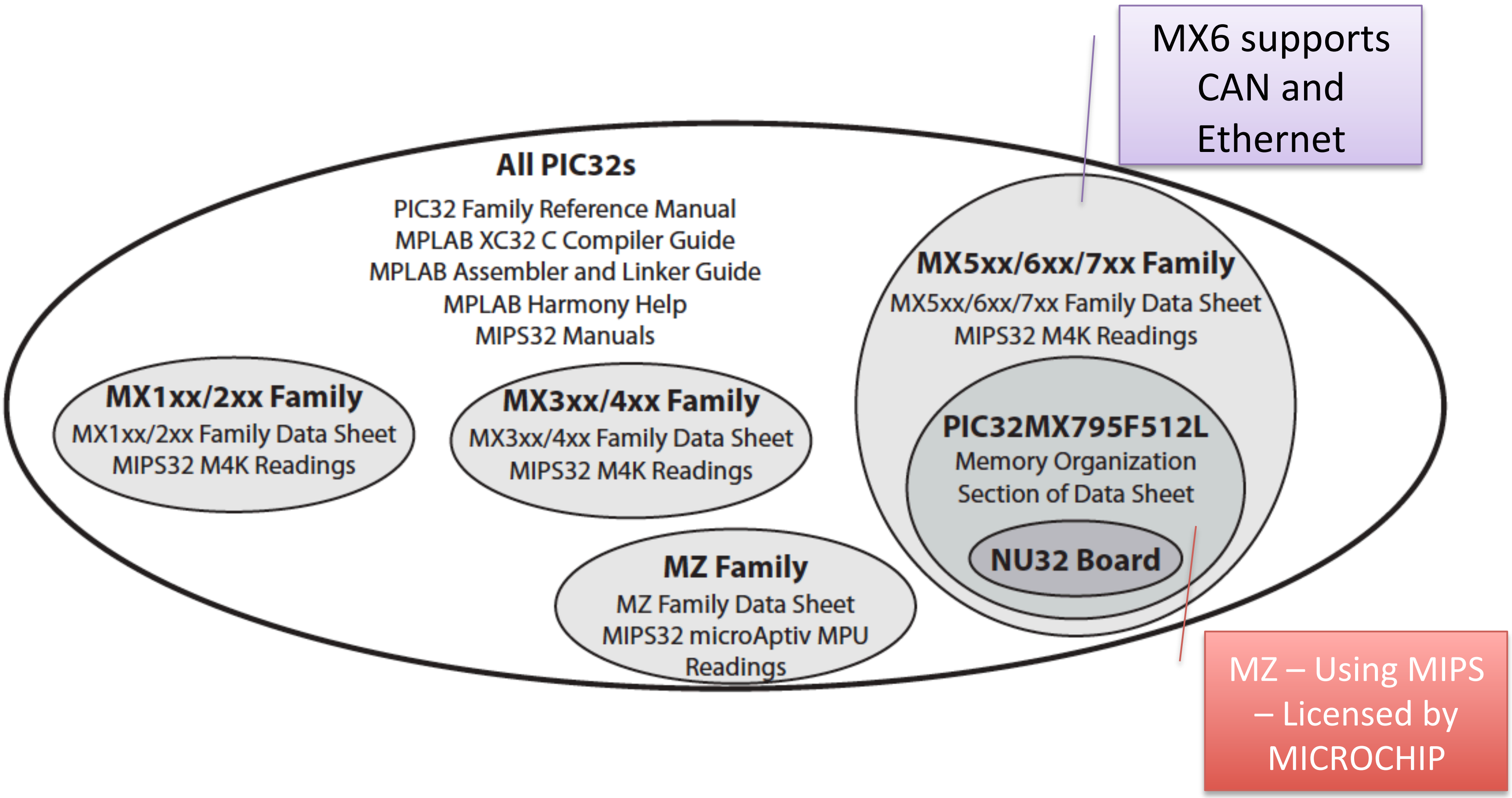
Different in terms of IO pins / RAM (data memory) / FLASH (program memory; non-volatile) / Peripherals

32-bit in terms of

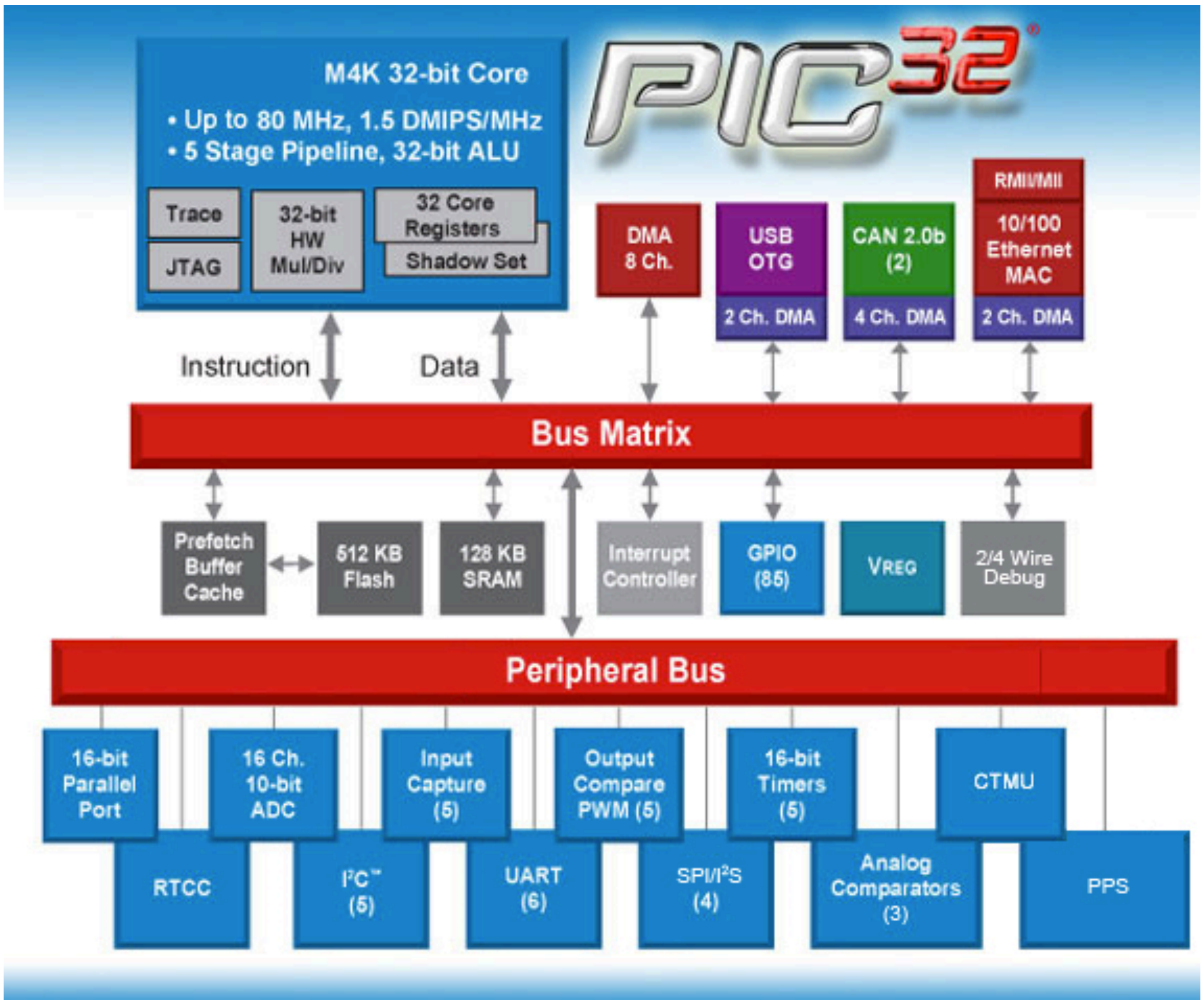
Instructions, register size, Instruction Bus, Data Bus



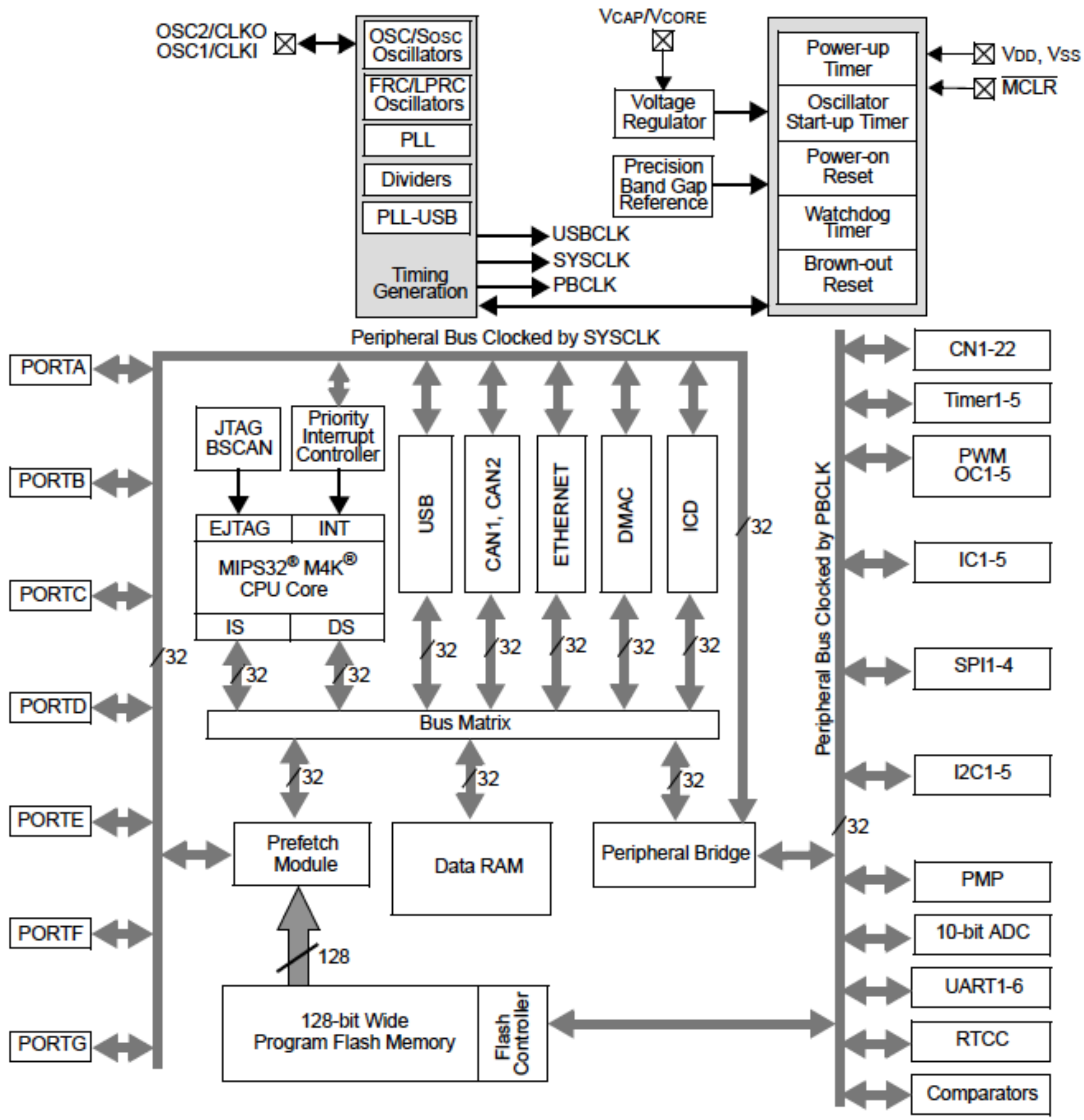
PIC32 Family



PIC32 Architecture



Architecture:



Example

Select Product Family: 32-bit PIC Microcontrollers - All

32-bit PIC Microcontrollers - All

View All Parametrics

Product	Pins	MHz	Flash KB	RAM	Temperature Range	Operation Voltage Range	USB	Ethernet
PIC32MX110F016B	28	40	16	4096	-40 to 105	2.3V - 3.6V	None	None
PIC32MX110F016C	36	40	16	4096	-40 to 105	2.3V - 3.6V	None	None
PIC32MX110F016D	44	40	16	4096	-40 to 105	2.3V - 3.6V	None	None
PIC32MX120F032B	28	50	32	8192	-40 to 105	2.3V - 3.6V	None	None
PIC32MX120F032C	36	50	32	8192	-40 to 105	2.3V - 3.6V	None	None
PIC32MX120F032D	44	50	32	8192	-40 to 105	2.3V - 3.6V	None	None
PIC32MX120F064H	64	50	64	8192	-40 to 105	2.3V - 3.6V	None	None
PIC32MX130F064B	28	40	64	16384	-40 to 105	2.3V - 3.6V	None	None
PIC32MX130F064C	36	40	64	16384	-40 to 105	2.3V - 3.6V	None	None
PIC32MX130F064D	44	40	64	16384	-40 to 105	2.3V - 3.6V	None	None

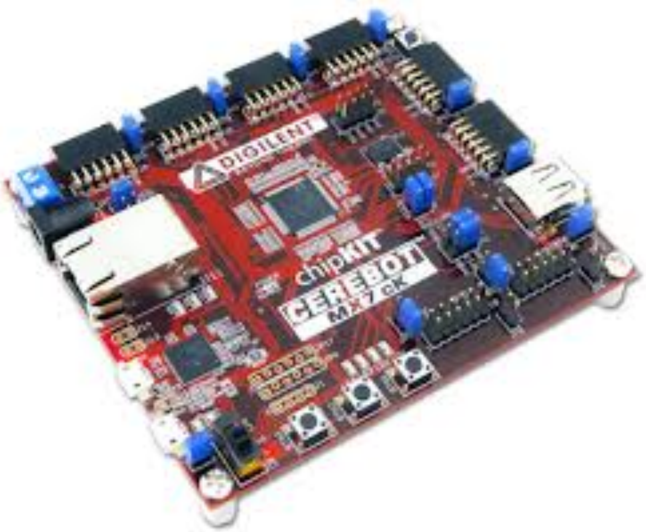
http://www.microchip.com/pagehandler/en-us/family/32bit/

Example

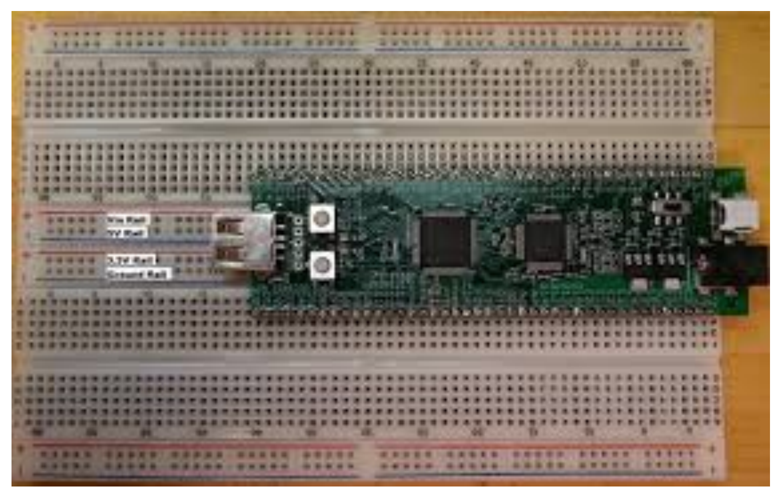
The PIC32MX795F512L is powered by a supply voltage in the range 2.3 to 3.6 V and features a max clock
frequency of 80 MHz,
512 KB program memory (Flash)
128 KB data memory (RAM)
1610-bit analog-to-digital input lines (multiplexed to a single analog-to-digital converter, or ADC),
USB 2.0,
Ethernet,
two CAN modules, I2C and four SPI synchronous serial communication modules, six UARTs for RS-232 or RS-485 asynchronous serial communication…….

Parameter Name	Value
Family	PIC32MX7xx
Max Speed MHz	80
Program Memory Size (KB)	512
RAM (KB)	128
Auxiliary Flash (KB)	12
Temperature Range (C)	-40 to 105
Operating Voltage Range (V)	2.3 to 3.6
DMA Channels	8
SPI™	4
I^2C™ Compatible	5
USB	FS Device/Host/OTG
USB (Channels, Speed, Compliance	1,FS Device/Host/OTG,USB 2.0 OTG
CAN	2
A/D channels	16
Max A/D Resolution	10
Max A/D Sample Rate (KSPS)	1000
Input Capture	5
Output Compare/Std. PWM	5
16-bit Digital Timers	5

Development Boards



ard



Development Boards

- Hardware
  - PIC32 USB Starter Kit – With on-board programmer
  - Development Board – Need PICKIT3
  - Development Board with a Boot Loader – Can be programmed using USB cable
- Software
  - XC32 Compiler
  - MPLABX

Sample Prog:

```
#include <plib.h>

// configuration bits are not set by a bootloader, so set here
#pragma config DEBUG = OFF          // Background Debugger disabled
#pragma config FPLLMUL = MUL_20     // PLL Multiplier: Multiply by 20
#pragma config FPLLIDIV = DIV_2     // PLL Input Divider:  Divide by 2
#pragma config FPLLODIV = DIV_1     // PLL Output Divider: Divide by 1
#pragma config FWDTEN = OFF         // WD timer: OFF
#pragma config POSCMOD = HS         // Primary Oscillator Mode: High Speed xtal
#pragma config FNOSC = PRIPLL       // Oscillator Selection: Primary oscillator w/ PLL
#pragma config FPBDIV = DIV_1       // Peripheral Bus Clock: Divide by 1
#pragma config BWP = OFF            // Boot write protect: OFF
#pragma config ICESEL = ICS_PGx2    // ICE pins configured on PGx2
#pragma config FSOSCEN = OFF        // Disable second osc to get pins back
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set for interrupt priority 7

#define SYS_FREQ 80000000           // 80 million Hz

void delay(void);

int main(void) {

  SYSTEMConfig(SYS_FREQ, SYS_CFG_ALL); // cache on, PBCLK setup, min flash wait
  DDPCONbits.JTAGEN = 0; // Disable JTAG, make pins 4 and 5 of Port A available.
  TRISA = 0xFFCF;        // Pins 4 and 5 of Port A are LED1 and LED2.  Clear
                         // bits 4/5 to zero, for output.  Others are inputs.
  LATAbits.LATA4 = 0;    // Turn LED1 on and LED2 off.  These pins sink ...
  LATAbits.LATA5 = 1;    // ... current on NU32, so "high" = "off."

  while(1) {
    delay();
    LATAINV = 0x0030;    // toggle the two lights
  }
  return 0;
}

void delay(void) {
  int j;
  for (j=0; j<1000000; j++) { // number is 1 million
    while(!PORTDbits.RD13);   // Pin D13 is the USER switch, low if pressed.
  }
}
```

http://www.johnloomis.org/microchip/pic32/resources.html

Introduction to MIPS

- Microprocessors without Interlocked Pipelines Stages
  - MIPS I, II, …V, 32, 64
- Developed by MIPS Technology
- RISC Instruction Sets
- 32-bit Instructions
  - R-type; I-type, & J-type instructions
- Applications:
  - Routers, Switches, Laser Printers, Sony Station, Nintendo 64, etc.
- Main Competitor is ARM
  - PDAs and Cellphones

MIPS Assembly

- Basic commands:
  - Arithmetic, Data Transfer, Logic, Bit operation, Branch, Jump

Add	add $d,$s,$t	$d = $s + $t	R	0	20_{16}	adds two registers, executes a trap on overflow `000000ss sssttttt ddddd--- --100000`
Add unsigned	addu $d,$s,$t	$d = $s + $t	R	0	21_{16}	as above but ignores an overflow `000000ss sssttttt ddddd--- --100001`
Subtract	sub $d,$s,$t	$d = $s - $t	R	0	22_{16}	subtracts two registers, executes a trap on overflow `000000ss sssttttt ddddd--- --100010`
Subtract unsigned	subu $d,$s,$t	$d = $s - $t	R	0	23_{16}	as above but ignores an overflow `000000ss sssttttt ddddd000 00100011`

MIPS Assembly

- Basic commands:
  - Arithmetic, Data Transfer, Logic, Bit operation, Branch, Jump

Store word	sw $t,C($s)	Memory[$s + C] = $t	I	$2B_{16}$	-
Store half	sh $t,C($s)	Memory[$s + C] = $t	I	29_{16}	-
Store byte	sb $t,C($s)	Memory[$s + C] = $t	I	28_{16}	-

stores a word into: MEM[$s+C] and the following 3 bytes. The order of the operands is a large source of confusion.
stores the least-significant 16-bit of a register (a halfword) into: MEM[$s+C].
stores the least-significant 8-bit of a register (a byte) into: MEM[$s+C].

https://en.wikipedia.org/wiki/MIPS_instruction_set

Assembly Programing Example

```
/* leds.S
   Written <date> by <your_name>@hmc.edu
   Test PIC by turning on LEDs        */

#include <P32xxxx.h>

#  Define constants
#define  LEDS  0xA5

#  Define functions
.global main

#  Compiler instructions
.text    # store the code in the main program section of RAM
.set noreorder # do not let the compiler reorganize your code

# Main program

.ent main      # Start function block
main:
   la    $t0, TRISD  # Load the address of TRISD into $t0
   addi  $t1, $0, 0xFF00
   sw    $t1, 0($t0) # TRISD = 0xF00 (bottom 8 bits outputs)
   addi  $t1, $zero, LEDS  # $t1 = LEDS (LEDS + 0)

write:   # This is a label you can jump to
   la    $t0, PORTD  # Load the address of PORTD into $t0
   sw    $t1, 0($t0) # PORTD = $t1
   j     write       # Jump back to write
   nop
.end main      # End function block
```

http://pages.hmc.edu/jspjut/class/f2014/e155/lab/lab04.pdf