Knowledge of baseline

Switching a source from one PIC to another

The aim of the exercise is to better understand how it is possible to switch the source from one PIC to another.

STOP Obviously, it **is NOT POSSIBLE to pass a program that uses certain resources of a microcontroller to another that does not have them!**

For example, a program written for 12F508 can be switched to 12F509 without any changes other than in the processor definition line, but the reverse is not necessarily possible if the program, written for 12F509, uses more memory than 12F508 offers.
And you certainly won't be able to run a program that requires the ADC module on a chip that doesn't have one, or to control 10 lines of I/O directly from a microcontroller in an 8-pin package. Or use the internal 8MHz oscillator for a chip that only has the frequency of 4MHz. And so on.

Switching a program from one chip to another is only possible if the two chips have the same resources used by the program!

Not the same resources at all, but the ones that the program uses.
Thus, it will be possible to switch the source from a Midrange, equipped with interrupts and maybe UART and many k of memory, but which are not used, to a Baseline that itself has neither interrupts, nor large memory, nor UART. And a source written for a Baseline will be able to move to a higher PIC without difficulty, since all the resources present in the first will also be present in the second.

By "passing the source" we mean transporting as it is, with the minimum possible modifications, the text written for one processor to another with a different acronym. It is not, therefore, a complete rewrite, but an adaptation of the source to the characteristics of the new target, without changing the structure of the instructions that make up the core of the program.

In this sense, there is still a stumbling block in the transition between chips of different families: that of instruction sets that are different, increasing the number of opcodes available as the level of the family advances. The 4 levels of 8-bit PICs are characterized by increasing instruction sets, with a gradually larger instruction bus:

	Baseline	Midrange	Enhanced Mid.	PIC18F
Core	12-bit	14-bit	14-bit	16-bit
Guidelines	33	35	49	83

Of course, it will not be possible to pass a source written for PIC18F and that uses the specific instructions present only in this type of components, for example for table management or hardware multiplication, on a Baseline, unless major changes (which go beyond this writing). But the opposite will be possible, since the opcodes of one set are present in the upper one. The same goes for the special instructions of the Enhanced Midrange set (some of the

which aren't even in the PIC18F set).
Of course, Baselines, as we have seen, use obsolete opcodes, such as `option` and `tris`, but when they are corrected, the MPASM compiler comes to the rescue, which on the one hand correctly compiles these instructions for Midranges, where they are still possible to apply (even if not recommended by Microchip), indicating the action with a message, or sending an error where compilation is not possible.

What we are interested in making clear here is that more often than you think it is possible to pass a source from one PIC to another and, in particular, how to "recover" the huge amount of examples written for obsolete chips, but at their time very widespread, such as 16C54, 16F84, etc. and put them back into operation on more current PICs. In the case of these tutorials, we explicitly refer to the Baseline PCIs we are covering.
And, remaining within the same family, especially for simple components like these, the switching of the source from one chip to another is just as simple. .

Switching the source from Baseline to Baseline

In the various examples presented in the previous tutorials we have the same compileable source for different members of the family and it should be clear that the differences between one and the other version of the text refer only to the following fields:

1. **definition of the processor**, with the inclusion of the ***.inc file*** suitable for the chosen chip
2. **initial config**, due to the slight differences in resources between the chips (oscillator and EEPROM memory), but mainly to "oddities" of the labels in the ***.inc files***, different from each other for the same resources
3. **I/O setup according to the number of pins** (GPIO for 6-pin and 8-pin chips and PORT for larger chips)
4. **disabling functions that by default overlap with digital I/O** (where necessary) and that are chip-specific, but also common to groups of chips, which makes work easier.

The logic and instructions of the actual program remain unchanged; They only change if we want to perform different actions, such as taking advantage of a greater number of available pins.

This is possible because the members of the same family are designed around a single basic idea, use similar hardware structures and the same set of instructions.

To the above we can add that for the transition it is necessary to observe the integrated modules available and the size of the RAM and Flash, since, as mentioned, it will not be possible to pass a program that uses the resources of one chip into another that does not have them to the same extent. Among these, of course, is the impossibility of passing a program that uses, for example, the ADC module to one that does not have it.
However, in general, Baselines have more or less the same structure and only where you use a lot of memory or modules such as comparators, ADCs or EEPROMs you need to be more careful.
Here is a table that collects various models with the salient features.

PIC	Pin	Flash [kB]	RAM [bytes]	Internal Clock [MHz]	Timer0	Comp.	ADC	EEPROM
10F200	6	0.375	16	4	1			
10F202		0.75	24	4	1			
10F204		0.375	16	4	1	1		
10F206		0.75	24	4	1	1		
10F220		0.375	16	4/8	1		2	
10F222		0.75	23	4/8	1		2	
12F508	8	0.75	25	4	1			
12F509		1.5	41	4	1			
12F510		1.5	38	4/8	1	1	3	
12F519		1.5	41	4/8	1			Yes
16F505	14	1.5	72	4	1			
16F506		1.5	67	4/8	1	2	3	
16F526		1.5	67	4/8	1	2	3	Yes
16F54	18	0.75	25	no	1			
16F527	20	1.5	68	4/8	1	2	8	Yes
16F57	28	3	72	no	1			
16F570	28	3	64	8	1	2	8	Yes
16F59	40	3	134	no	1			
16F707	40	14	368	16	3		14	Yes

If we exclude the 16F527/570 and 707, which are a bit peculiar and equipped with more peripherals, even unusual for the family, such as op-amps (16F527), 16bit timers (16F707), UART/SPI/I2C (16F707), and which are of uncommon use, and the very old 16F5x, direct heirs of the 16C5x, with limited resources and without internal oscillator, the others differ little from each other.

If we analyze some sources, we can quickly see the confirmation of what has been said so far.

We exclude the introductory comments, which do not become part of the compilation, although, as should be clear, they constitute documentation that should never be overlooked, and we highlight the differences. For example, let's look at a step 12F519->12F508:

LIST p=12F519 #include <p12F519.inc> radix dec ;============================== ; CONFIGURATION = ;============================== ; ; Internal oscillator, no WDT, no CP, pin4=MCLR ; __config _IntRC_OSC & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF & _MCLRE_ON ;============================== ; = RAM MEMORY = ; general purpose RAM CBLOCK 0x07 d1,d2,d3 ENDC ;============================== ;= LOCAL MACROS = ; Controls for the LEDf_ON MACRO LED bsf LEDf ENDM LED_ON MACRO bsf LEDl ENDM LED_OFF MACRO bcf LEDl ENDM ;============================== ; = RESET ENTRY = ; Reset Vector RESET_VECTOR ORG 0x00 ; Calibration of the internal oscillator movwf OSCCAL ;============================== MAIN: ; Reset Initializations ; OPTION default 11111111 ; 1------- done ; -1------ done ; --0----- Disable T0CKI ; ---1---- done ; ----1--- done ; -----111 done movlw b'11010111'	LIST p=12F508 #include <p12F508.inc> radix dec ;============================== ; CONFIGURATION = ;============================== ; Internal oscillator, no WDT, no CP, pin4=MCLR ; __config _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_ON ;============================== ; = RAM MEMORY = ; general purpose RAM CBLOCK 0x07 d1,d2,d3 ENDC ;============================== ; = LOCAL MACROS = ; Controls for the LEDf_ON MACRO LED bsf LEDf ENDM LED_ON MACRO bsf LEDl ENDM LED_OFF MACRO bcf LEDl ENDM ;============================== ; = RESET ENTRY = ; Reset Vector RESET_VECTOR ORG 0x00 ; Calibration of the internal oscillator movwf OSCCAL ;============================== MAIN: ; Reset Initializations ; OPTION default 11111111 ; 1------- done ; -1------ done ; --0----- Disable T0CKI ; ---1---- done ; ----1--- done ; -----111 done movlw b'11010111'

option clrf GPIO ; GPIO preset latch to 0 ; TRISGPIO --111010 GP0/2 out movlw b'11111010' triplets GPIO ; To the Management Register	option clrf GPIO ; GPIO preset latch to 0 ; TRISGPIO --111010 GP0/2 out movlw B'11111010' Tris GPIO
; Lights solid LED LEDf_ON ; flashing cycle according to goto mainloop LED ; table jumps to subroutines Delay05s goto Dly05s ================================= Mainloop: ; lights up LED LED_ON call Delay05s LED_OFF call Delay05s goto mainloop	; Lights solid LED LEDf_ON ; flashing cycle according to goto mainloop LED ; table jumps to subroutines Delay05s goto Dly05s ================================= Mainloop: ; Lights up LEDs LED_ON call Delay05s LED_OFF call Delay05s goto mainloop
;================================= ; = SUBROUTINES = ; Delay = 1/2 second @4 MHz Dly05s: movlw 0x03 movwf d1 movlw 0x18 movwf d2 movlw 0x02 movwf d3 Dly05s_0: decfsz d1, f goto $+2 ; decfsz d2, f goto $+2 ; decfsz d3, f Goto Dly05s_0 Goto $+1 retlw 0	;================================= ; = SUBROUTINES = ; Delay = 1/2 second @4MHz Dly05s: movlw 0x03 movwf d1 movlw 0x18 movwf d2 movlw 0x02 movwf d3 Dly05s_0: decfsz d1, f goto $+2 ; decfsz d2, f goto $+2 ; decfsz d3, f Goto Dly05s_0 Goto $+1 retlw 0

Notice how the difference between the source for the 12F519 and 12F508 is reduced to the fields identified at the beginning (for the config, for example, 519 has the 4/8MHz option of the internal oscillator and 508 does not), while for the rest they are completely identical.
Out of the entire list, only 3 lines require variations!

Moreover, we have seen how the same source can be adapted to several processors that have similar resources, only with the variation of the processor definition: 12F508 and 12F509, for example However, even for seemingly different processors, it is likely that the changes to be made to the source are limited. For example, let's see two incipits for a source suitable for **12F508/509** and **16F505/526**:

```
 ; choice of processor
 #ifdef __12F509
 LIST p=12F509 ;
 #include <p12F509.inc>
 #endif
 #ifdef __12F508
 LIST p=12F508
 #include <p12F508.inc> #endif

  radix dec


; ##############################
; CONFIGURATION
;
; Internal Oscillator, No WDT, No
CP, MCLR

__config _IntRC_OSC & _WDT_OFF &
_CP_OFF & _MCLRE_ON


;
##############################
; RAM
; general purpose RAM
     CBLOCK 0x07
 Counter
 D1, D2, D3
     ENDC
```

```
; choice of #ifdef 16F526
 processor

 LIST p=16F526
 #include <p16F526.inc>
 #endif __
 #ifdef   16F505
 LIST p=16F505
 #include <p16F505.inc> #endif
   radix dec

; #######################
; CONFIGURATION
 #ifdef __16F526
; Internal Oscillator, 4MHz, No WDT,
No CP, RB3=MCLR
   config _IntRC_OSC_RB4 && _CP_OFF &
__
 IOSCFS_4MHz & _WDTE_OFF
_CPDF_OFF & _MCLRE_ON
 #endif

 #ifdef __16F505
 __config _IntRC_OSC & _WDT_OFF &
_CP_OFF & _MCLRE_ON
 #endif

; #################################
; RAM
; general purpose RAM
   CBLOCK 0x10
 counter
 d1,d2,d3
   ENDC
```

We see, in addition to the need to adjust definitions and configs, the different beginning of RAM (a detail that, using the modular structure and **UDATA** instead of **CBLOCK**, we could easily eliminate). But let's observe how the config of 16F505 is completely "compatible" with that of its 8-pin relatives!

Other differences we've seen may be that the chip with multiple pins will have the I/O collected in PORT instead of GPIO, but these are other details that can be easily adjusted.

However, it should always be remembered that pins share access to different internal resources, and multiple functions are competing on the same pin. And, in general, chips with a higher number of pins integrate more functional modules than chips with a low pin count.

Then, perhaps more subtle, is the need to disable the resources present on the pins in the default to the POR, resources that do not allow access to the functions of simple digital I/O pins if they are not disabled. This happens, of course, only for chips that have these resources:

; disable T0CKI to have GP2 as digital I/O ; b'11010111' ; 1------- GPWU disabled ; -1------ GPPU disabled ; --0----- clock internal ; ---1---- falling ; ----1--- prescaler al WDT ; -----111 1:256 movlw b'11011111' OPTION	; inizializzazioni al reset #ifdef 16F526 ; Disable Analog Inputs clrf ADCON0 ; Disable comparators to free up the digital function bcf CM1CON0, C1ON bcf CM2CON0, C2ON #endif ; disable T0CKI to have GP2 as digital I/O ; b'11010111' ; 1------- GPWU disabled ; -1------ GPPU disabled ; --0----- clock internal ; ---1---- falling ; ----1--- prescaler al WDT ; -----111 1:256 movlw b'11011111' OPTION

We can see that in the listing on the left, suitable for 12F508/9 and 16F505, no special actions are required to get to the digital I/O, while in the one on the right, for 16F526, it is necessary to disable comparators and analog.
The configuration section of the **OPTION** register, being in both cases Baseline, is completely identical.

By comparing the various listings offered for the first exercises, the differences will be immediately evident and therefore it will be possible to implement them in any other circumstance.

We should have outlined that switching from one chip to another within the Baseline family is nothing complicated. Again: of course you won't be able to run tutorial 5 on a 10F200, but only because it doesn't have enough pins. If there were no such limit, the changes in the source would only be those indicated so far.

So, from now on, the exercises will be centered, in the practical part, on a particular chip, the most appropriate or significant one, leaving it to you to put in place the necessary modifications to adapt it to other PCIs.

16F84 -> Baseline Conversion

The transfer of the source to other families of PICs follows very similar procedures, but it can be more complex, since the members of the higher families are much more numerous, have different structures (and, sometimes, small specific "oddities"), as well as a number of peripherals that can be very large and therefore require more attention in the initial phases.
What will certainly remain constant is the set of instructions and their logic, since the 14-bit and 16-bit sets are practically an extension of the 12-bit Baseline set.
Where the program uses instructions that are not present in the original, it is often possible to replace them with simple actions.

Here we want to see how many of the things written for the most famous ICP can be replicated on other PICs.

There are thousands of examples based on 16F84/A on WEB . It is a very old midrange and now largely obsolete, but, at the time, it was the first microcontroller chip with Flash memory, which allowed the component to be reused numerous times (previously PICs were xxCxxx, where C indicates a one-time writable program memory. It is evident that the novelty was welcomed with great enthusiasm by the experimenters (and not only), decreeing the success of the component. Unfortunately, using this chip now, which Microchip is trying to dismiss, has a considerable cost: as of today (Microchip Direct) 16F84A is available for €3.12 (+VAT and shipping) while a 16F505 costs only €0.65 and a current 18F2321 with 28 pins, 10MIPS costs €2.48.
There is, therefore, no reason to buy the obsolete chip now, other than the desire to try one of the many circuits that have been, over time, posted on the Internet.

However, if we consider some points, we see that it is possible in many cases to pass a program written for 16F84 to a Baseline, even if 16F84 is a Midrange.

We are based on two considerations.
The first is that it is, yes, a Midrange, but very old and therefore rather "primitive" compared to other more recent chips. We won't go into the details of the Midrange family now, which is the subject of another section of the course. Let's limit ourselves here to outline a few elements: like all Midrange, 16F84 has:

- **Larger instruction set** - compared to Baselines, there are 35 instructions, but if the program doesn't use the two that don't exist in the 12-bit core, there's no problem with conversion. And, even in case there are these two opcodes, they can be replaced quite easily.

- **interrupt** - only if the program uses this function is the conversion not possible, since the Baselines have no interrupts

- **EEPROM** - only some Baselines have this option. If the program makes use of it, transposition is only possible for these few chips.

- **SFR on two banks** - Baselines basically have only one for the most common registers, and the conversion involves deleting the source lines that command the bank switch.

- **RA4 is an open drain output** - this "oddity" was inserted by Microchip in some products of that period, but then seems to have been completely abandoned. This situation limits the use of the pin for 16F84. In Baselines, all pins are totem poles.

- **It's an 18-pin** - but there are only 13 I/O. A conversion to a 14-pin chip baseline is only possible if 12 I/Os are used. If you use a lot of pins, there can be difficulties in maintaining the relevant source parts since 14-pin chips have the I/O organized in two 6-bit ports each, while 16F84 has a full 8-bit port + one 4-bit port. We have seen, and we will see, that it is still possible to organize the two 6 ports to have one of 8, but this requires more work than a simple transcription of the text

- **MCLR is a separate pin** and is not excludable. This is not a problem.
- **It does not have an internal oscillator** - only external oscillators are possible, but these are headed by two OSC1/OSC2 pins that have only and exclusively this function (in the most recent PICs it is shared with digital I/O). This creates problems only if you have to use a frequency other than that of the internal oscillator, since, if an external oscillator is required, one or two I/Os are lost on the Baselines. However, even in this circumstance, it is probably possible to revise the tempo routines and adjust them to 4/8MHz.
- **RAM starts at 0Ch** - you need to move the reference to the start point of the Baseline RAM, or use the UDATA statement.
- **The RAM is 68 bytes and the program memory is 1k** - for the conversion you need to use no more than the Baseline on which you intend to run it can offer.

There are other differences, such as, for example, fewer limitations in the placement of subroutines and lookup tables in memory, and, in this case, a more thorough revision would be required, dictated by an equally thorough knowledge of the characteristics of the components. However, in the case of small programs, you usually don't have to deal with these elements.

And, on the other hand, **it is highly advisable, if you want to try your hand at source conversions, to start with simple examples and, only after understanding the key mechanisms, move on to more challenging listings.**

Otherwise, the structure of the **`STATUS`, `OPTION`, `PORT`**, **Timer0** and **WDT registers** is very similar to that of the Baselines. This similarity is made even more pronounced by the fact that the 16F84 has no analog functions or other integrated modules.

Warning

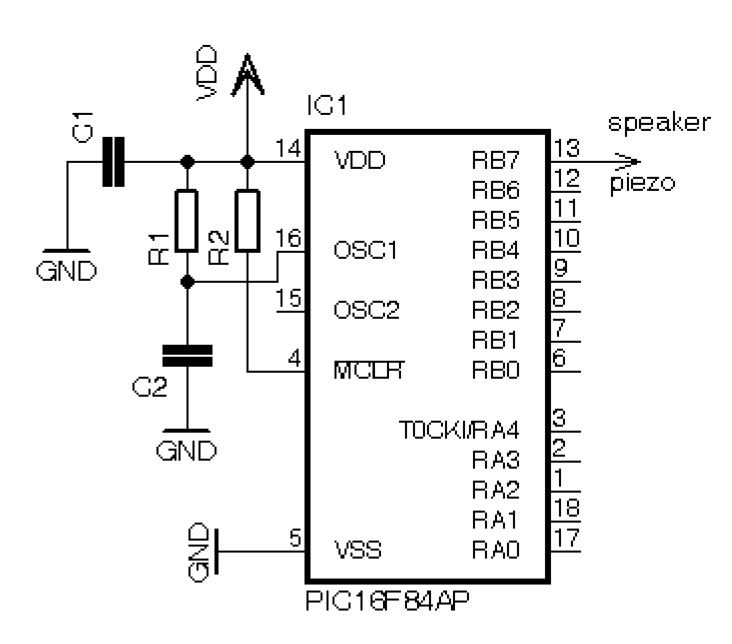
You're not claiming that any application written for 16F84 can pass on a Baseline:

- 16F84, as simple as it is, is a Midrange

and if interrupts are used, for example, the pass can only be on another midrange, but not on baselines that do not have interrupts.
As mentioned at the beginning, **it is NOT POSSIBLE to pass a program that uses certain resources of a microcontroller to another that does not have them!**
Fortunately, many applications written for 16F84 are very simple and suitable for this kind of conversion.
So, we can try to take something from the WEB, written for 16F84 or 16F84A and pass it to a Baseline. Here are a couple of examples.

Generate a French type siren tone.

Let's take the example from an interesting site, Talking Electronics, and in particular experiment 7a.
The circuit referred to is this:



The circuit, around the 16F84A 18-pin is reduced to the bare minimum:

1. C1 is the always indispensable filter on the Vdd.
2. The oscillator is an external RC (R1/C2), for economy. With 4k7 and 22pF the frequency obtained is about 4MHz.
3. The MCLR pin is pull up with the classic 10k (R2).
4. The RB7 output is directed to a piezo speaker.

Also on our LPCuB there is a miniature speaker with its driver that lends itself well to the function.

The program alternately generates two square wave trains, at different frequencies, a higher one (Hee) and a lower one (Haw), switching an I/O pin configured as a digital output; The pin controls a piezo buzzer or speaker.

The form of the original source is not the best, as it is dated, and should not be used as an example, but the comment lines help to understand how it works.

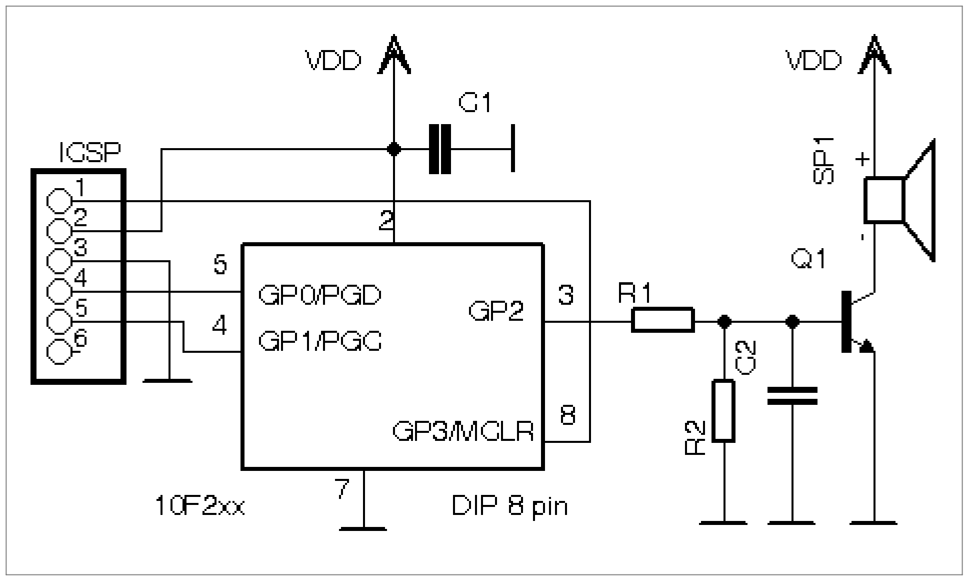
```
;Expt7a.asm
;Project: Hee Haw Sound
    List P = 16F84
   #include <p16F84.inc>
  __CONFIG 1Bh     ;_CP_OFF & _PWRTE_ON & _WDT_OFF & _RC_OSC

  ORG 0

     BSF 03,5      ;Go to Bank 1
     CLRF 06       ;Make all port B output
     BCF 03,5      ;Go to Bank 0 - the program memory area.
     CLRF 06       ;Clear display
     GOTO Hee1

Hee1 MOVLW 0FFh    ;Number of loops
     MOVWF 14h     ;The loop file
Hee2 MOVLW 0C0h    ;Duration of HIGH
     BSF 06,7      ;Turn on piezo
Hee3 NOP
     DECFSZ 15h,1 ;Create the HIGH time
     GOTO Hee3
     MOVLW 0C0h    ;Duration of the LOW
     MOVWF 15h
     BCF 06,7      ;Turn off piezo
Hee4 NOP
     DECFSZ 15h,1 ;Create the LOW time
     GOTO Hee4
     DECFSZ 14h,1 ;Decrement the loop file
```

```
     GOTO Hee2

     MOVLW 0C0h    ;Number of loops
     MOVWF 14h     ;The loop file
Haw1 MOVLW 0FFh
     MOVWF 15h
     BSF 06,7      ;Turn on piezo
Haw2 NOP
     DECFSZ 15h,1 ;Create the HIGH time
     GOTO Haw2
     MOVLW 0FFh    ;Duration of the LOW
     MOVWF 15h
     BCF 06,7      ;Turn off piezo
Haw3 NOP
     DECFSZ 15h,1 ;Create the LOW time
     GOTO Haw3
     DECFSZ 14h,1 ;Decrement the loop file
     GOTO Haw1     ;Do more cycles
     GOTO Hee1

  END
```

Looking at the source, you can see that:

- **opcodes that do not exist in the Baseline set are not used**. So there is no need for changes in this regard
- **It doesn't use interrupts** , so it doesn't hinder conversion
- **does not use EEPROM** : you can use any Baseline
- **bank switches are used**, but Baselines do without them, and the conversion only deletes the relevant source lines.
- **RA4 is not used** as an open drain.
- **only one I/O pin is used** , so the PIC10F2xx can also be used.
- **An RC oscillator is used**, but, as mentioned, it is about 4MHz, adequate to be replaced by the internal oscillator of the Baseline.
- **it only makes use of very little RAM and program memory** . No subroutines or other more complex programming elements are used.

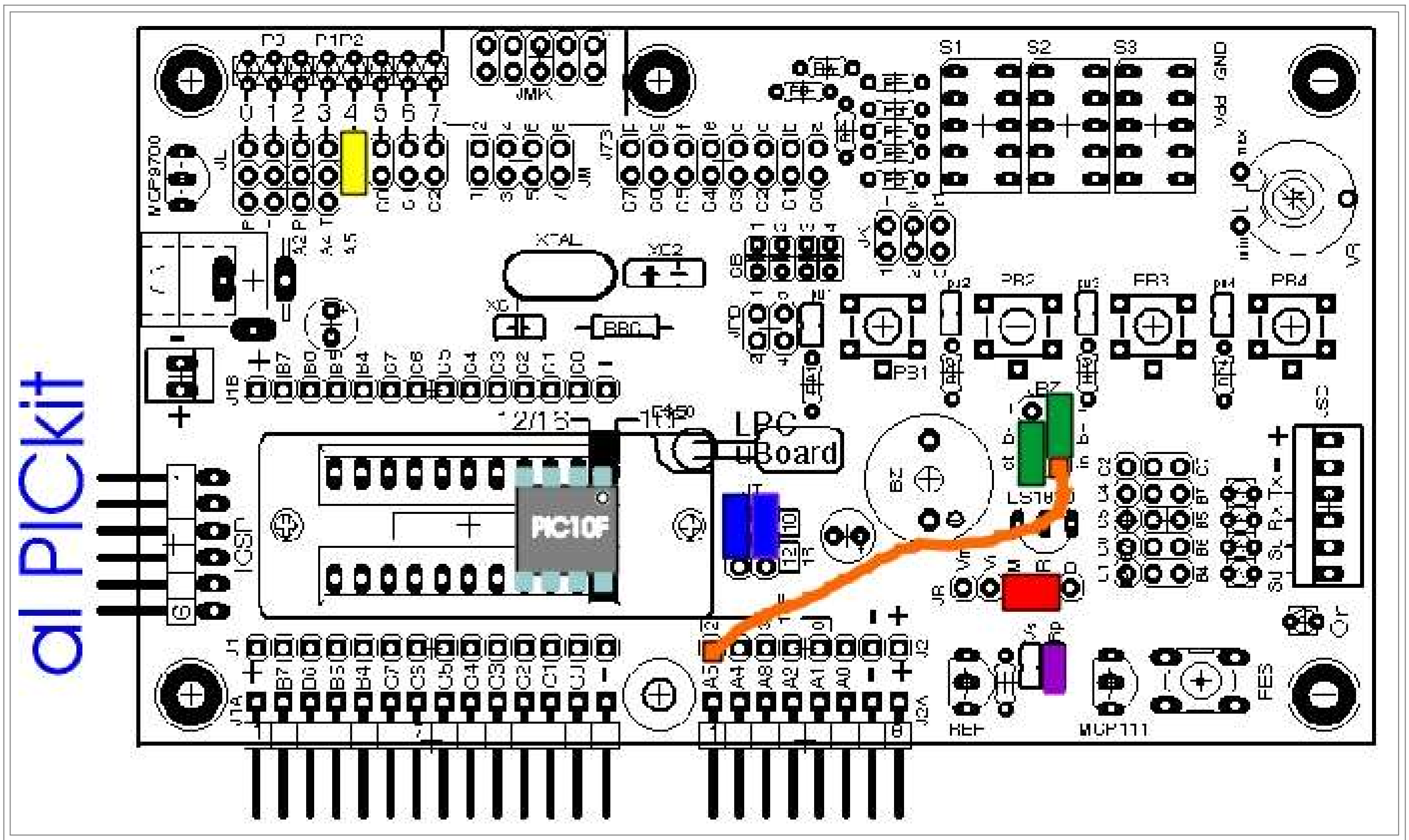
On the other hand, in 16F84, **the RAM starts at 0Ch** and you need to move the reference to the start point of the Baseline RAM, or use the UDATA statement. So we can think of switching the source to a 10F200, which is more than adequate. The wiring diagram looks like this:



R1, R2, C2, Q1, and SP1 are already installed on the **LPCuB** board. Q1 is a generic NPN, SP1 is a miniature PCB speaker, R1= 15k, R2=10k and C2=10-22nF.

As for the sound produced, it is square waves, which, added to the poor performance of the miniaturized speaker, makes the acoustic quality typically "harsh". R1/C2 are a low-pass filter that, by eliminating the steepest wavefronts, slightly improves the quality of the sound produced, reducing the highest frequencies.

Connections on the **LPCuB**:



The "green" jumpers configure the speaker area as shown in the diagram. The "orange" flying jumper connects **GP2** to the base of the transistor.
The "yellow" jumper is optional: connect an LED on the **GP2** output to display the signal. The "red" and "purple" jumpers are related to Reset, which is not used in the tutorial. Always observe the insertion position of the 10F2xx and the "blue" jumpers.

We use a 10F20x since only 1 pin is required in output, but you can use any other PIC, adjusting the resources.

The original source, seen above, is certainly written some time ago, with a lot of absolutes, for a compiler different from the one of the MPLAB environment we are using; This shape makes it almost illegible. The switch, however, is very easy because the resources of the 16F84, as absolute addresses, are the same as those of the Baselines.

Address	16F84 Function	10F20x Function
03	STATUS	STATUS
05	DOOR	OSCCAL
06	PORTB	GPIO
07	-	COMCON0
0Ch-4Fh	RAM	10h(08)-1Fh

Essentially, it is just a matter of adjusting the RAM area. So, leaving absolutes galore, without varying the shape, we have:

```
;Expt7a.asm
;Project: Hee Haw Sound
;     List P = 16F84
;    #include <p16F84.inc>
     LIST P=10F204
    #include <10f204.inc>
;   __CONFIG 1Bh    ;_CP_OFF & _PWRTE_ON & _WDT_OFF & _RC_OSC
; Internal Oscillator, cp, WDT e MCLR off
 __config _CP_OFF & _MCLRE_OFF & _WDT_OFF

  ORG 0
;---> Add Internal Oscillator Calibration ANDLW
     0xFE             ; Ensures non Fosc/4
     MOVWF 05h        ; OSCCAL

;---> Add Disable Comparator per 10F204/6 MOVLW 0xF7  ;
     no comparator
     MOVWF 07h        ; CMCON0

;---> Add edit to log in a GP2 come I/O
; Turn off T0CKI MOVLW
     b'11010111'
     OPTION

     ; BSF 03,5      ;Go to Bank 1
     ; CLRF 06       ;Make all port B output
     MOVLW 0        ;Make all port B output
     TRIS GPIO
     ; BCF 03,5      ;Go to Bank 0 - the program memory area.
     ; CLRF 06       ;Clear display
```

```
     CLRF 06      ; clear GPIO
     GOTO Hee1

Hee1 MOVLW 0FFh   ;Number of loops
     MOVWF 14h    ;The loop file
Hee2 MOVLW 0C0h   ;Duration of HIGH
     ; BSF 06,7     ;Turn on piezo
     BSF 06,2     ;Turn on piezo
Hee3 NOP
     DECFSZ 15h,1 ;Create the HIGH time
     GOTO Hee3
     MOVLW 0C0h   ;Duration of the LOW
     MOVWF 15h
     ; BCF 06,7     ;Turn off piezo
     BCF 06,2      ;Turn off piezo
Hee4 NOP
     DECFSZ 15h,1 ;Create the LOW time
     GOTO Hee4
     DECFSZ 14h,1 ;Decrement the loop file
     GOTO Hee2

     MOVLW 0C0h   ;Number of loops
     MOVWF 14h    ;The loop file
Haw1 MOVLW 0FFh
     MOVWF 15h
     ; BSF 06,7     ;Turn on piezo
     BSF 06,2     ;Turn on piezo
Haw2 NOP
     DECFSZ 15h,1 ;Create the HIGH time
     GOTO Haw2
     MOVLW 0FFh   ;Duration of the LOW
     MOVWF 15h
     ; BCF 06,7     ;Turn off piezo
     BCF 06,2     ;Turn off piezo
Haw3 NOP
     DECFSZ 15h,1 ;Create the LOW time
     GOTO Haw3
     DECFSZ 14h,1 ;Decrement the loop file
     GOTO Haw1    ;Do more cycles
     GOTO Hee1

  END
```

In clearer fonts, we've highlighted the changes we've made. These are only ever actions to adjust the I/O and characteristics of the processor.
We observe that they derive from the different structure of the processor:

- Changing the Processor Definition
- Changing the config (as well as in symbolic form)
- Added instructions for calibrating the internal oscillator (optional, but why not add them?). In particular, the pin used for the output, **GP2**, also has the function of ***Fosc/4***: for this purpose the oscillator calibration bit 7 is reset to ensure that the function is not activated by mistake in the value loaded in **OSCCAL**
- added instructions to access **digital GP2** : **GP2** is also **T0CKI,** so we have to act on the **OPTION** to disable the choice and finally get to the digital I/O function.

- elimination of the main switches, which are not needed, having the 10F200 all the SFRs in a single bank.

Adaptation to the TRIS `instruction` of the Baselines

As for the addition of the `ANDLW line 0xFE ; ensures that Fosc/4 is not` assured, it must be said that, if the factory calibration value is intact, it is not necessary at all, since the 0 bit of this value is always and in any case at 0. However, it can happen that, during experiments, the chip is erased and rewritten without attention to the calibration byte, which can also take on an inadequate value. Hence the precaution.
It must be said, however, that access to a chip with the wrong calibration byte is indicated in the control windows of the Pickit and, through them, it is possible to restore the correct value at any time.

The use of absolutes has been retained, which does not help to clarify the source, only because it allows us to clearly see that, even in using them, the differences in the sources are minimal, since the 8-bit PICs are born from the same design philosophy and therefore have a very similar structure, including SFR addresses or bit positions in the registers.

The RAM memory used consists of two locations at 14h and 15h; These locations are also available in the 10F20x memory map, so you don't need to change them.

Note for the perplexed: in the valid data RAM area, one cell is as good as the other. There is no reason to use them consecutively, starting with the first one available, except for a matter of order. You could use any other, starting from the bottom or center.
So, if the memory of the 10F200 starts at 10h and ends at 1Fh, the 14h and 15h locations are perfectly usable, as would 10h and 11h or 1Eh and 1Fh.
If you want to strictly adapt the memory allocation to the characteristics of the processor, you can simply use the UDATA statement, which is linked to the file *processorname.inc*.

As you can see, this is a very simple operation, since the structure of the program is only marginally modified.

If, on the other hand, we do a little revision and extend the compilation to 4 possible chips, in addition to using symbols and not absolutes, we get a much more readable, clear and pleasant source, to which we add an initial comment area that allows the source text to be a self-documentation.

```
;**************************************************************************
;---------------------------------------------------------------------------
;
; Title          : Assembly & C Course - Tutorial 6A_hh
;
; Project Conversion from http://www.talkingelectronics.com/projects/
; PIC_LAB-1/PicLab1_experiments-P4.html from 16F84 to 10F200
; This experiment creates a Hee Haw sound suitable for an alarm.
; Higher frequency cycles (HEE) are generated, and for a
; set time. The frequency of the HAW is lower.
; The program therefore consists of two sections: HEE and HAW. Every
; section has two nested loops: the inner loop creates the
; Time for the high and low level of a single cycle and the loop
```

```
; external creates the number of cycles.
; In the original the oscillator is an external RC with 4k7 and 22pF for
; a calculated frequency of about 4MHz. Here it is replaced by the
; 4Mhz internal oscillator.

; PIC            : 10F200/2/4/6
; Support        : MPASM
; Version        : 1.0
; Date           : 01-05-2013
; Hardware ref. :
; Author         :Afg
;*********************************************************************
; Pin use :
; ----------------
; 10F200/2/4/6 @ 8     DIP   10F200/2/4/6 @ 6 pin SOT-23
;                      pins
;       |¯¯\/¯¯|                    *______|
; NC  -|1      8|- GP3        GP0  -|1     6|- GP3
; Vdd -|2      7|- Vss        Vss  -|2     5|- Vdd
; GP2 -|3      6|- NC         GP1  -|3     4|- GP2
; GP1 -|4      5|- GP0              |______|
;       |______|
;
;                        DIP SOT
; NC                      1: Nc
; Vdd                     2: 5: ++
; GP2/T0CKI/FOSC4/[COUT] 3: 4: Out speaker
; GP1/ICSPCLK/[CIN-]      4: 3:
; GP0/ICSPDAT/[CIN+]      5: 1:
; NC                      6: Nc
; Vss                     7: 2: --
; GP3/MCLR/VPP            8: 6:
;
; [] only 10F204/6
;*********************************************************************
;=====================================================================
; DEFINITION OF PORT USE
;=====================================================================
; GPIO map

; | 3 | 2 | 1 | 0 |
; |----|-----|---|---|
; | in | Spkr|   |   |
;
;#define GPIO,GP0 ;
;#define GPIO,GP1 ;
#define soundpin GPIO,GP2 ; out speaker


;#define GPIO,GP3          ; Input only

;*********************************************************************
;---> Replacing the processor definition
; List P = 16F84
;#include <p16F84.inc>
; ####################################################################
; Choice of processor
 #ifdef __10F200
    LIST p=10F200
    #include <p10F200.inc>
 #endif
```

```
 #ifdef___10F202
    LIST p=10F202
    #include <p10F202.inc>
 #endif
 #ifdef___10F204
    LIST p=10F204
    #include <p10F204.inc>
#define proccomp
 #endif
 #ifdef___10F206
    LIST p=10F206
    #include <p10F206.inc>
#define proccomp
 #endif

;---> Configuration Line Replacement
; abolishing the hexadecimal form
;  CONFIG 1Bh ;_CP_OFF & _PWRTE_ON & _WDT_OFF & _RC_OSC

; #####################################################################
; CONFIGURATION
;
; No WDT, no CP, pin4=GP3
  __config _CP_OFF & _MCLRE_OFF & _WDT_OFF


;---> defines RAM memory in symbolic
   CBLOCK 0x10          ; RAM Area
  cnttime
  cntloop
   END

     ORG 0

;---> add internal oscillator calibration andlw
                0xFE        ; ensures not FOSC/4
     movf       OSCCAL

;---> add comparator disable for 10F204/6 #ifdef
 proccomp
     movlw      0xF7        ; No Comparator
     movf       CMCON0
 #endif

;---> add edit to access GP2 as I/O
; disable T0CKI
     movlw      b'11010111'
     OPTION

;---> Delete Bank Switches
; BSF 03.5 ; Go to Bank 1

;---> replace absolutes with labels, adjust instructions
; to the Baseline set and fix incorrect comment
; CLRF 06 ; Make all port B output
     movlw    0             ; GPIO = out
     TRIS     GPIO

;---> Delete Bank Switches
; BCF 03.5 ; Go to Bank 0 - the program memory are
```

```
;---> replace absolutes with labels
; CLRF 06
     CLRF     GPIO          ;Lactch presets to 0

;---> line  useless
;      Goto     Hee1

Hee1 movlw    0FFh          ;       of loops
     movwf    cntloop       Number
Hee2 movlw    0C0h                  of the high
                            ;Durati level
                            on

; Replace Absolutes with Label
; BSF 06.7 ; Turn on piezo
     Bsf      soundpin      ; pin on

Hee3 nop
     decfsz cnttime,f ; time on-1 if 0 skips
      Goto    Hee3
     movlw    0C0h          ;D low level uration movwf
              cnttime

;---> replace absolutes with labels
; BCF 06.7 ; Turn off piezo
     Bcf     soundpin       ; Pin OFF

Hee4 nop
     decfsz cnttime,f       ; Time Off
      Goto Hee4
     decfsz cntloop,f       ; loop-1 - if 0
      jumps Goto Hee2

     movlw 0C0h             ; Number of
     loops movwf cntloop
Haw1 movlw 0FFh             ;
     duration movwf cnttime

; Replace Absolutes with Label
; BSF 06.7 ; Turn on piezo
     Bsf     soundpin ;p in
     on

Haw2 nop
     decfsz cnttime,f       ; time on -1 - if 0 jumps
      Goto Haw2
     movlw 0FFh             ;D Out
     movwf cnttime

;---> replace absolutes with labels
; BCF 06.7 ; Turn off piezo
     Bcf     soundpin       ;p in off

Haw3 nop
     decfsz cnttime,f       ; Time Off
      Goto Haw3
     decfsz cntloop,f       ; loop - 1 if 0
      jumps Goto Haw1       ; Other Haw Cycles
     Goto    Hee1           ; Resume Hee Cycles

   END
```

The choice of processors has gone a bit wider, including all 10F2xx, since **PIC10F200/2/4/6** are pretty much the same as far as this application is concerned. In fact, **10F204/6 has a comparator** and, if selected, this should be disabled. This is done by associating the definition of these processors with a `proccomp label` that will be used immediately afterwards to insert where necessary the instructions to disable the analogue.
Obviously, if you use only one specific chip, these adaptation lines can very well be eliminated.

Still on the subject of RAM, for what was said before, we see that it is initialized starting from 0x10. This is a suitable address for all definable chips, even if the 10F204/6 starts at 08h: for both at 10h it allows you not to have two different RAM declarations to select with the #ifdef. In the "reconstructed" form of the source, the use of absolute addresses is still excluded with the creation of appropriate labels, which allows for greater understanding (and portability, ease of maintenance, etc.).

Once this is done, the logic of the program is not changed in the slightest, so much so that it is not even necessary to discuss its structure.

Note that programs that run on Baseline without the use of special resources can run on any other PIC, since the higher instruction sets include all those of the 12-bit core. So, something like the one written above for a small 10F200 will work just as well on a Midrange as it does on an 18F.

Generate musical notes

On a hardware basis identical to the previous one, we see how the instructions that create a delay between the high and low level applied to the output produce square wave cycles of defined frequency and duration. When the output pin is connected to a speaker, the square wave is reproduced as an audible frequency, a note. A series of notes and rests is the basis of a piece of music.
On a hardware basis identical to the previous one, we see how instructions that create a delay between the high/low switching of the output pin can produce square wave cycles of defined frequency and duration.
Let's take from: http://www.talkingelectronics.com/projects/PIC_LAB-1/PicLab1_experiments-P3.html example 6:

```
                    ;Expt6.asm
                    ;Project: Creating a tune
List P = 16F84
#include <p16F84.inc>
__CONFIG 1Bh    ;_CP_OFF & _PWRTE_ON & _WDT_OFF & _RC_OSC

      ORG 0           ;This is the start of memory for the program.
SetUp       BSF 03,5        ;Go to Bank 1
      CLRF 06         ;Make all port B output
      BCF 03,5        ;Go to Bank 0 - the program memory area.
      CLRF 06         ;Clear outputs of junk
SetUp1  MOVLW 01
      MOVWF 0Ch
      GOTO Main

table       ADDWF 02h,1
      RETLW 0A8h  ;duration - 168 loops
      RETLW 5Bh   ;G - 392Hz 1.27mS HIGH,LOW - 91 loops
      RETLW 0FAh  ;duration - 250 loops
      RETLW 6Bh   ;E - 330Hz 1.51mS HIGH,LOW - 107 loops
      RETLW 46h   ;duration - 70 loops
      RETLW 6Bh   ;E - 330Hz
      RETLW 54h   ;duration - 84 loops
      RETLW 5Bh   ;G - 392Hz
      RETLW 5Eh   ;duration - 94 loops
      RETLW 51h   ;A - 440Hz - 1.13mS HIGH,LOW - 81 loops
      RETLW 0FCh  ;duration - 252 loops
      RETLW 7Ah   ;D - 292Hz - 1.71mS HIGH,LOW - 122 loops
      RETLW 0FFh ;End of table
      RETLW 0FFh  ;End of table

delay   NOP       ;Create 10uS delay
      NOP
      NOP
      NOP
      NOP
      NOP
      RETURN

Delay2  CALL Delay      ;Create 3mS delay
```

```
      DECFSZ 1A,1
      GOTO Delay2
      RETURN

Delay3  NOP            ; 250mS
      delay DECFSZ 1A,1
      GOTO  Delay3
      DECFSZ 1B,1
      GOTO  Delay3
      RETURN

Main DECF 0C,1        ;D ec jump value to re-look at
      values MOVF 0Ch,0       ;Copy jump-value into W
      CALL Table      ; Return with table-value in
      W MOVWF 0F      ; Length of note into file 0F
      INCF 0Ch,1      ;Increment the table-value
      MOVF 0Ch,0      ; Copy jump-value into W
      CALL Table      ; Return with table-value in W
      MOVWF 0D        ; Frequency of note into file
      0D MOVWF 0E     ; Frequency of note into file
      0E BSF 06.7
      CALL  Delay  ;  Create  HIGH  time
      DECFSZ  0D,1  ;  Each  loop  =  14uS
      GOTO Main2
      BCF 06.7
      CALL Delay      ; Create LOW
      time DECFSZ 0E,1
      GOTO Main3
      DECFSZ 0F,1     ; Length of
      note GOTO Main1
      BCF 06.7
      CALL  Delay2  ;  3mS  between  notes
      CALL  Delay2  ;  3mS  between  notes
      CALL Delay2 ; 3mS between notes
      INCF 0Ch,1      ;Increment pointer to next value in table
      MOVF 0Ch,0      ; Copy jump-value into W
      CALL Table      ; Return with table-value in W
      MOVWF 10h       ;P ut "end of table" into file
      10h MOVLW 0FFh  ; Check for 'end of table'
      XORWF 10h,0     ; Compare file 10h with FF (result in
      W) BTFSC 03,2   ; Look at Zero flag in status file
      GOTO Main4      ; Start again
      INCF 0Ch,1      ;Increment the table-value
      GOTO Main       ; Go to next note
      Main4 CALL Delay3
      CALL Delay3
      CALL Delay3
      GOTO SetUp1

      END
```

The text suffers even more than the previous one from the problems we have already seen, which arise from the age of the text and the lack of more advanced tools for compiling it at the time. We see how the use of absolutes makes it difficult to follow what the instructions are doing, while some comments are really not very useful. However, it is a good example to use to verify how it is possible to transform the source from one PIC to another.

First, however, a necessary note on the comments.

Comments

One of the topics of the course is the understanding of the rules that allow you to write a good program. Comments are not a minor part of this, not only in Assembly, but also (and sometimes more) in C or BASIC.
Comments are meant to clarify what the program is doing, especially where it's not immediately comprehensible. So, let's look at some key points:

- **The comment that merely reiterates the function of the opcode is a useless comment**: this is usually one of the most annoying comments. For example,:

```
    nop         ; No Operation
```

It is evident that such a comment is completely useless, it adds text to the source and complicates its reading without providing any useful indication as to why that opcode is put there. This is different:

```
    nop         ; 1us
```

- **Repeating a comment several times is also useless**:

```
      CALL Delay2       ;3mS between notes
      CALL Delay2       ;3mS between notes
      CALL Delay2       ;3mS between notes
```

better certainly, and more useless:

```
      CALL Delay2       ; 3 x 3mS = 9ms pause between notes
      CALL Delay2
      CALL Delay2
```

- **A comment that doesn't clarify anything is worse than useless.**

```
      CLRF 06           ;Clear outputs of junk
```

It's nonsense. 06 is not the ecological island! Better, certainly, and more useful:

```
      CLRF 06           ;Reset all outputs
```

- **A comment that does not relate to the situation does not clarify anything .**

```
      MOVWF 10h         ;Put "end of table" into file 10h
```

It's nonsense: what if the byte taken from the table is not the one at the end of the table? Better, certainly, and more useful:

```
      MOVWF 10h         ;Save the value taken from the table for
                        ; Compare it with the end-of-table indicator
```

- **A comment that is missing at a key point does not allow the reader to understand the logic of the action .**

```
        GOTO SetUp1
```

What does it refer to? Better, certainly, and more useful:

```
        GOTO SetUp1      ;Repeat the song all over again
```

- **A logical block needs a comment in order to be clear about its function on first reading .**

```
        MOVWF 10h        ;Put "end of table" into file 10h
        MOVLW 0FFh       ;Check for 'end of table'
        XORWF 10h,0      ;Compare file 10h with FF (result in W)
        BTFSC 03,2       ;Look at Zero flag in status file
        GOTO Main4       ;Start again
        INCF 0Ch,1       ;Increment the table-value
        GOTO Main        ;Go to next note
```

Apart from the difference in language, better and more useful:

```
; End-of-table verification
; if the value taken is FFh, end of the track
        MOVWF 10h         ; save value taken from the
        MOVLW 0FFh table ; compare with the end of the table value
        FFh XORWF 10h,0   ; through an XOR
        BTFSC 03.2        ; if different, Z = clear flag - skip next
        GOTO Main4 statement    ; if equal, end of track
        INCF    0Ch,1     ; different - increment counters
        for GOTO          Main    ; Withdraw next note
```

That is: comments are an essential part of the source and should not be neglected, otherwise it will be difficult, if not impossible, for others to read the source, but also, and unfortunately, to themselves after some time.

Take care of the comments, their quality, and their real usefulness.

Conversion

Let's switch the original source, like the previous one, to PIC10F200/2/4/6 (but you can easily switch it to any other Baseline).

In the meantime, let's see how, also here:

- Interrupts are not used
- no resources outside the capabilities of the Baselines are used
- instructions that are not in the baseline set are not used. Even the opcode **`return`**, as we saw earlier, is allowed by MPASM, with a warning message:

```
Warning[227] full path : Substituting RETLW 0 for RETURN pseudo-op
```

that it does not prevent the success of the compilation; MPASM generates the right binary code for `retlw 0 for us`.

So the operation is feasible without any problems.

The operation is quite linear: after initializing the I/O, a step counter is reset and the duration of the note is taken from the table, which is saved in a RAM location.
Then you increment the step counter to access the next row in the table, which contains the note value. This is saved in two counters that are used to time level 1 and level 0 of the square wave. This is repeated for the indicated duration, after which the counter is incremented to pick up a new duration/frequency pair.
The program checks the value taken from the table and if it is equal to FFh, this indicates the end of the song. A longer pause and then the cycle starts again. If the value is different from FFh, a short pause and you move on to playing the note.

If we don't want to "embellish" the source, this is a pure translation of the original (for 10F204 only). It is necessary to:

- Change the processor definition
- Changing the config
- Delete the bank switch
- add the necessary instructions to access the digital I/O
- modify the access to the TRIS register, which with the Baselines can only be dealt with in writing with the special TRIS instruction
- modify the addresses of Ram that starts from 10h and not from 0Ch (and, since they are all absolute, you have to change them one by one... 'na crap...)
- change the output pin (this too absolutely...)

```
                    ;Expt6.asm
                    ;Project: Creating a tune
  ;List P = 16F84
  ;#include <p16F84.inc>
 ;__CONFIG 1Bh     ;_CP_OFF & _PWRTE_ON & _WDT_OFF & _RC_OSC
  List P = 10F204
  #include <p10F204.inc>
 __config  _CP_OFF & _MCLRE_ON  & _WDT_OFF

      ORG 0          ;This is the start of memory for the program.
SetUp ;BSF 03,5       ;Go to Bank 1
      ;CLRF 06        ;Make all port B output
      ;BCF 03,5       ;Go to Bank 0 - the program memory area.
;----------------------------------------------------------
; Add for adapt the processor
;---> Add Internal Oscillator Calibration
andlw   0xFE    ; assure no Fosc/4
movwf   OSCCAL
;---> add comparator disable for 10F204/6
        movlw   0xF7          ; no comparator
        movwf   CMCON0
;---> Add edit to log in a GP2 come I/O
;     Turn off T0CKI
movlw b'11010111'
```

```
        option
;---> Direction accessible only with TRIS
        movlw 0
        TRIS  06
;-------------------------------------------------------
        CLRF 06         ; Clear outputs of
junk SetUp1             MOVLW 01
      MOVWF 10h ; 0Ch
      GOTO Main

table ADDWF 02h,1
      RETLW 0A8h  ; Duration - 168 loops
      RETLW 5Bh   ; G - 392Hz 1.27mS HIGH,LOW - 91 loops
      RETLW 0FAh ; Duration - 250 loops
      RETLW 6Bh   ; E - 330Hz 1.51mS HIGH,LOW - 107 loops
      RETLW 46h   ; duration - 70
      loops RETLW 6Bh ; E - 330Hz
      RETLW 54h   ; duration - 84
      loops RETLW 5Bh ; G - 392Hz
      RETLW 5Eh   ; Duration - 94 loops
      RETLW 51h   ; A - 440Hz - 1.13mS HIGH,LOW - 81 loops
      RETLW 0FCh ; Duration - 252 loops
      RETLW 7Ah   ;D - 292Hz - 1.71mS HIGH,LOW - 122 loops
      RETLW 0FFh ; End of
      table RETLW 0FFh ; End
      of table

Delay  NOP        ; Create 10uS
      delay NOP
      NOP
      NOP
      NOP
      NOP
      RETURN

Delay2  CALL Delay      ; Create 3mS
      delay DECFSZ 1A,1
      GOTO Delay2
      RETURN

Delay3  NOP            ; 250mS
      delay DECFSZ 1A,1
      GOTO  Delay3
      DECFSZ 1B,1
      GOTO  Delay3
      RETURN

Main DECF 10.1 ;0C,1        ;D ec jump value to re-look at
      values MOVF 10.0 ;0Ch,0           ; Copy jump-value into
      W
      CALL Table      ; Return with table-value in W
      MOVWF 13 ;0F         ; Length of note into file 0F
      INCF 10h,1 ;0Ch,1       ;Increment the table-value
      MOVF 10h,0 ;0Ch,0       ; Copy jump-value into W
      CALL Table      ; Return with table-value in W
      MOVWF  11 ; 0D    ; Frequency of note into file 0D
      MOVWF 12 ; 0E   ; Frequency of note into file 0E BSF
      06.2 ; 7
      CALL Delay      ; Create HIGH time
      DECFSZ 11.1 ;0D,1     ; Each loop =
      14uS
      GOTO Main2
      BCF 06.2 ; 7
```

```
CALL Delay      ; Create LOW time
```

```
DECFSZ 12,1 ;0E,1
GOTO Main3
DECFSZ 13,1 ;0F,1      ;Length of note
GOTO Main1
BCF 06,2 ;7
CALL  Delay2   ;3mS  between  notes
CALL  Delay2   ;3mS  between  notes
CALL Delay2     ;3mS between notes
INCF 10h,1 ;0Ch,1      ;Increment pointer to next value in table
MOVF 10h,0 ;0Ch,0        ;Copy jump-value into W
CALL Table      ;Return with table-value in W
MOVWF 14h ;10h         ;Put "end of table" into file 10h
MOVLW 0FFh      ;Check for 'end of table'
XORWF 14h,0 ;10h,0      ;Compare file 10h with FF (result in W)
BTFSC 03,2      ;Look at Zero flag in status file
GOTO Main4      ;Start again
INCF 10h,1 ;0Ch,1       ;Increment the table-value
GOTO Main       ;Go to next note
Main4 CALL Delay3
CALL Delay3
CALL Delay3
GOTO SetUp1
END
```

It should be established by now that using absolutes is a very impractical thing, since, as we have seen, a good quality compiler allows you to use symbols, with enormous advantages:

- Simplicity of writing
- Easy to read and understand
- Ease of editing

The use of absolutes was necessary in primitive compilers, but with programs like MPASM they are certainly not the right way. So, if we do a little make-up, we get not only something prettier, but also much more understandable and self-documenting, as well as easy to maintain and modify.

```
;**********************************************************************
;-----------------------------------------------------------------------
;
;    Title          :  course Assembly & C - Exercise 6A_heyj
;
; Project Conversion from http://www.talkingelectronics.com/projects/
; PIC_LAB-1/PicLab1_experiments-P6.html from 16F84 to 10F200
; This experiment creates a series of notes.
; A table contains pairs of successive values: the first is the duration
; of the note, the second is the tone.
; These values are used to create loops to generate square waves of the
; desired frequency for the indicated time (based on 4MHz oscillator).
; The table ends with the FFh value, so it can only be used
; for this purpose.
; The table in the example contains the first 6 notes of "Hey Jude" by
; Beatles.
; The series of notes is played continuously.
;
;    PIC            :  10F200/2/4/6
;    Support        :  MPASM
;    Version        :  1.0
```

```
;    Date           :  01-05-2013
;    Hardware ref. :
;    Author         :  Afg
;
; ########################################################################
;
;  Pin use :
;  ---------------
;     10F200/2/4/6 @ 8 pin DIP      10F200/2/4/6 @ 6 pin SOT-23
;
;                  __  __                   * ______
;                 |  \/  |                    |      |
;          NC -|1     8|- GP3        GP0 -|1     6|- GP3
;         Vdd -|2     7|- Vss        Vss -|2     5|- Vdd
;         GP2 -|3     6|- NC         GP1 -|3     4|- GP2
;         GP1 -|4     5|- GP0             |______|
;                 |______|
;
;                            DIP  SOT
;    NC                       1:   Nc
;    Vdd                      2:   5: ++
;    GP2/T0CKI/FOSC4/[COUT]   3:   4: Out speaker
;    GP1/ICSPCLK/[CIN-]       4:   3:
;    GP0/ICSPDAT/[CIN+]       5:   1:
;    NC                       6:   Nc
;    Vss                      7:   2: --
;    GP3/MCLR/VPP             8:   6: MCLR
;
;   [] only 10F204/6
;*********************************************************************
;=====================================================================
;             DEFINITION OF PORT USE
;=====================================================================
; GPIO map
; |  3  |  2  |  1  |  0  |
; |-----|-----|-----| ----|
; | MCLR| Spkr|     |     |
;
;#define     GPIO,GP0      ;
;#define     GPIO,GP1      ;
#define     soundpin GPIO,GP2     ; Out speaker
;#define     GPIO,GP3             ; MCLR

; ####################################################################
; Choice of processor
 #ifdef __10F200
        LIST       p=10F200
        #include <p10F200.inc>
 #endif
 #ifdef __10F202
        LIST       p=10F202
        #include <p10F202.inc>
 #endif
 #ifdef __10F204
        LIST       p=10F204
        #include <p10F204.inc>
#define proccomp
 #endif
 #ifdef __10F206
        LIST       p=10F206
        #include <p10F206.inc>
#define Proccomp
```

```
 #endif

; #################################################################
;                          CONFIGURATION
;
; No WDT, no CP, MCLR
  __config _CP_OFF & _MCLRE_ON   & _WDT_OFF

; #################################################################
;                          RAM
;---> defines RAM memory in symbolic
  CBLOCK 0x10                ; RAM Area
  stepcntr                   ; Step Counter
  noteH                      ; Output Time Counter at Level 1
  Note                       ; Output Time Counter at Level 0
  durcntr                    ; Note Counter
  Temp                       ; temporary
  D1                         ; Temporary for Delay
  D2
  ENDC

; #################################################################
; Reset Vector
        ORG 0

SetUp
; #################################################################
; Additions to match the processor
;---> add internal oscillator calibration
        andlw    0xFE          ; ensures no Fosc/4 also in terms of
        value movwf           OSCCAL; incorrect calibration
;---> add comparator disable for 10F204/6 #ifdef
  proccomp
        movlw    0xF7          ; No MovWF
        Comparator            CMCON0
  #endif
;---> add edit to access GP2 as I/O
;     disable T0CKI movlw
                b'11010111'
        option
;---> Direction accessible only with special TRIS opcode
        movlw    0
        TRIS     GPIO
; #################################################################

        CLRF     GPIO           ; Clear Out

 SetUp1 movlw    0xFF           ;p Step Counter Reset
        movwf    stepcntr
        Goto     Main

; #################################################################
;            TABLES AND SUBROUTINES (in the first 256 bytes)
; Lookup table by
duration/note Table    addwf
        PCL,f
        retlw    0xA8           ; Duration - 168 loops
        retlw    0x5B           ; G - 392Hz 1.27mS HIGH,LOW - 91
        loops retlw            0xFA  ; Duration - 250 loops
        retlw    0x6B           ; E - 330Hz 1.51mS HIGH,LOW - 107
        loops retlw            0x46  ; Duration - 70 loops
        retlw    0x6B           ; E - 330Hz
```

```
        retlw   0x54        ; Duration - 84
        retlw   0x5B        loops
        retlw   0x5E        ; G - 392Hz
        retlw   0x51        ; Duration - 94     HIGH,LOW - 81 loops
        retlw   0xFC        loops
        retlw   0x7A        ; A - 440Hz -       HIGH,LOW - 122 loops
        retlw   0xFF        1.13mS
        retlw   0xFF        ; Duration - 252
                            loops
                            ;D - 292Hz - 1.71mS
                            ; End of table
                            ; End of table

; Ritardo 10us
Delay   Nop                 ; 1uS
        nop                 ; 1uS
        nop                 ; 1uS
        nop                 ; 1uS
        nop                 ; 1uS
        nop                 ; 1uS
        retlw   0           ; 2uS + 2us call

; 3ms Delay
Delay2 Call     Delay
        decfsz D1,F
         Goto   Delay2
        retlw   0

; delay 250ms
Delay3 nop
        decfsz d1.1
         Goto   Delay3
        decfsz d2.1
         Goto   Delay3
        retlw   0

;------------------------------------------------------------------------
Main    incf    stepcntr,f  ; Step Counter+1
; Load Duration from Table
        movf    stepcntr,w
        Call    Table       ; Table by duration
        movwf   durcntr     ; Save to Counter
; if =FF is fine
        movwf   Temp
        movlw   0xFFh
        XORWF   temp,w
        skpnz               ; If you are
         different, continue Goto Main4 ; if
         FFh end of song

; Load Note from Table
        incf    stepcntr,f ;
stepcounter+1 Main1   movf   stepcntr,w
        Call    Table       ; Load Note
        movwf   noteH       ; and save in
        counters movwf      Note

; Play Note
Main2   Bsf     soundpin
        Call    Delay       ; Time at High Level
        decfsz noteH,f      ;each loop = 14us
         Goto   Main2
Main3   Bcf     soundpin
```

```
Call     Delay       ; Time at low level
decfsz noteL,1
 Goto    Main3
```

```
        decfsz durcntr,f    ; End of Known Duration?
         goto   Main1       ; No - Charging Frequency Value
        bcf     soundpin    ; Yes - End of Note

; End Note - Short Pause
        call    Delay2      ; 3x3ms between
        call    Delay2
        call    Delay2
        goto    Main        ; next note

; End of Song - Long Pause
Main4   call    Delay3      ; 3x250ms prima della
        call    Delay3      ; ripetizione del brano
        call    Delay3
; riavvia brano
        goto    SetUp1

        END
```

Again, apart from the use of labels instead of absolutes, the changes are minimal and essentially concern the position of the end-of-track test in the loop, a position that is anticipated and saves instructions, as well as being more linear and understandable.
Comments have been changed or added to really clarify what the instructions do. Find this version in the 6A_heyj project folder.

Conclusions

It should be clear that switching a simple source from one PIC to another can be a very simple thing, even if we are dealing with members of different families.

On this site, and many others, there are a number of simple examples for older processors that can be transferred to newer components.

You can try some of them and see how difficult it is. In any case, we reiterate what has been said:

It is highly advisable, if you want to try your hand at source conversions, to start with simple examples and, only after understanding the key mechanisms, move on to more challenging listings.

In order to avoid disappointment and stress...

A helping hand

And a help certainly comes from the drafting of orderly, functional and documented sources. To achieve this, avoid these mistakes:

1. Never use absolute values: use labels. They make the source understandable and portable.
2. Don't enter SFR definitions: everything you need to define the microcontroller you're using is in the *processorname.inc file* you include at the beginning.
3. Don't insert macros for the actions that the Assembler offers you ready-made: inserting macro definitions such as Bank1, Bank0, etc. is useless: there are already banksel, pagesel, etc.
4. Don't pack the lines: give the text a shape that makes it pleasant and easy to read.
5. Don't omit comments: it's better to abound.
6. Decide which format to use for the number bases. A **`leading radix de`**c gives you the decimal as the default.
7. Don't mix the various possible forms to define numbers in different bases: for example, for hexadecimals and binaries, use only ***`0xnumerohex`*** and ***`b'binarynumber'`***.
8. Don't calculate numerical values manually, let the Assembler do it for you, just as you don't spend time manually calculating parameters for timings and the like: use the applets you find on the net.
9. Do not choose abstruse ways to achieve a goal: the simplest way, although not elegant, is often the most effective.
10. Don't neglect to draw a flowchart: it's not an extra, since it allows you to clarify ideas, facilitates transposition into instructions and allows you to understand what has been written after some time.

Instead:

1. Use a template, of any nature, but one that allows you to work tidily.
2. Use self-explanatory labels and variable names – they make your life so much easier.
3. Prefer to write relocatable or modular and reusable code: spend a little more time at the moment, but you will largely recover it later.
4. Comment on the code, especially where there was a "brilliant idea", otherwise after some time you will be in difficulty with your own work.
5. Make sure that your comments are sensible and functional: they help you understand what the program does.
6. Use plenty of space, align the line starts of similar elements, use blank lines to lighten text and make logical blocks more obvious.

6A_hh.asm

```
;*******************************************************************
; 6A_hh.asm
;-------------------------------------------------------------------
;
;    Title         : Assembly & C Course - Tutorial 6A_hh
;
; Project Conversion from http://www.talkingelectronics.com/projects/
; PIC_LAB-1/PicLab1_experiments-P4.html 16F84 to 10F200
; This experiment creates a Hee Haw sound suitable for an alarm.
; Higher frequency cycles (HEE) are generated, and for a
; set time. The frequency of the HAW is lower and the number of
; cycles must be processed so that the time for the HAW is even
; at the time for HEE.
; The program consists of two sections: HEE and HAW. Each section has
; Two nested loops: the inner loop creates the time period for the
; high and low level of a single loop, and the outer loop creates the
; number of cycles.
; The oscillator used is an external RC with 4k7 and 22pF for a
; calculated of about 4MHz

;    PIC            :  10F200/2/4/6
;    Support        :  MPASM
;    Version        :  1.0
;    Date           :  01-05-2013
;    Hardware Ref. :
;    Author         :  Afg
;
;
;  Pin usage :
;  ---------------
;     10F200/2/4/6 @ 8 PIN DIP      10F200/2/4/6 @ 6 pin SOT-23
;
;               |¯¯\/¯¯|                   *¯¯¯¯¯¯|
;          NC -|1    8|- GP3        GP0 -|1    6|- GP3
;         Vdd -|2    7|- Vss        Vss -|2    5|- Vdd
;         GP2 -|3    6|- NC         GP1 -|3    4|- GP2
;         GP1 -|4    5|- GP0              |______|
;               |______|
;
;                            DIP  SOT
;    NC                       1:  Nc
;    Vdd                      2:  5: ++
;    GP2/T0CKI/FOSC4/[COUT]   3:  4: Out speaker
;    GP1/ICSPCLK/[CIN-]       4:  3:
;    GP0/ICSPDAT/[CIN+]       5:  1:
;    NC                       6:  NC
;    Vss                      7:  2: --
;    GP3/MCLR/VPP             8:  6:
;
;  [] only 10F204/6
;*******************************************************************
;===================================================================
;            DEFINITION OF PORT USE
;===================================================================
```

```
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-----|-----|-----|--------|
; | in | Spkr|     |     |
;
;#define    GPIO,GP0    ;
;#define    GPIO,GP1    ;
#define    soundpin GPIO,GP2    ; Out speaker
;#define    GPIO,GP3            ; Input only

;*******************************************************************
;---> Replacing the processor definition
 ; List P = 16F84
 ;#include <p16F84.inc>
; ####################################################################
; Choice of processor
 #ifdef __10F200
        LIST      p=10F200     ; Processor Definition
        #include <p10F200.inc>
 #endif
 #ifdef __10F202
        LIST      p=10F202     ; Processor Definition
        #include <p10F202.inc>
 #endif
 #ifdef __10F204
        LIST      p=10F204     ; Processor Definition
        #include <p10F204.inc>
#define proccomp
 #endif
 #ifdef __10F206
        LIST      p=10F206     ; Processor Definition
        #include <p10F206.inc>
#define proccomp
 #endif

;---> Configuration Line Replacement
; abolishing the hexadecimal form
; __CONFIG 1Bh    ;_CP_OFF & _PWRTE_ON & _WDT_OFF & _RC_OSC

; ####################################################################
;                          CONFIGURATION
;
; No WDT, no CP, GP3
  __config _CP_OFF & _MCLRE_OFF & _WDT_OFF


;---> defines RAM memory as CBLOCK
  symbolic 0x10               ; RAM Area
 cnttime
 cntloop
  ENDC

  ORG 0

;---> add andlw internal oscillator calibration
           0xFE              ; Ensures no Fosc/4
    movwf   OSCCAL
```

```
;---> add comparator disable for 10F204/6 #ifdef proccomp
     movlw   0xF7               ; No MOVF
     Comparator                 CMCON0
 #endif

;---> add edit to access GP2 as I/O
;     disable T0CKI
     movlw   b'11010111'
     option

;---> Delete Bank Switches
;     BSF    03,5    ; Go to Bank 1

;---> replace absolutes with labels, adjust instructions
; to the Baseline set and fix incorrect comment
;     CLRF   06      ; Make all port B
     output movlw    0    ; GPIO = out
     TRIS    GPIO

;---> Delete Bank Switches
;     BCF    03,5    ; Go to Bank 0 - the program memory are

;---> replace absolutes with labels
;     CLRF   06
     CLRF    GPIO        ;p reset lactch to 0

;---> useless line
;     Goto    Hee1

Hee1 movlw   0FFh        ; Number of loops
     movwf   cntloop
Hee2 movlw   0C0h       ;Duration of the high
                                level

; Replace Absolutes with Label
;     BSF    06,7    ; Turn on
     piezo bsf       soundpin  ;
     pin on

Hee3 nop
     decfsz cnttempo,1 ; Time on-1 if 0 skip
      goto   Hee3
     movlw   0C0h        ;D low level uration of
     movwf   cnttime

;---> replace absolutes with labels
;     BCF   06,7    ; Turn off
     piezo bcf           soundpin
      ; Pin OFF

Hee4 nop
     decfsz cnttempo,1 ; Time Off
      Goto   Hee4
     decfsz cntloop,1    ; Loop-1 - If 0 Skip
      Goto   Hee2

     movlw   0C0h        ; Number of
```

```
movwf loops          cntloop
```

```
Haw1 movlw   0FFh         ; MovWF
     Durationcnttime

; Replace Absolutes with Label
;     BSF    06,7    ; Turn on
     piezo bsf       soundpin  ;p
     in on

Haw2 nop
     decfsz cnttempo,1 ; Time On -1 - SE 0 Skip
      Goto   Haw2
     movlw   0FFh        ;D urata
     off movwf           cnttime

;---> replace absolutes with labels
;     BCF   06,7     ; Turn off
     piezo bcf           soundpin
      ;p in off

Haw3 nop
     decfsz cnttempo,1 ; Time Off
      Goto   Haw3
     decfsz cntloop,1    ; Loop - 1 if 0 Jump
      Goto   Haw1        ; Other Haw Cycles
     Goto    Hee1        ; Resume Hee END

     Cycles
```

6A_heyj.asm

```
;*******************************************************************
; 6A_heyj.asm
;-------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 6A_heyj
;
; Project Conversion from http://www.talkingelectronics.com/projects/
; PIC_LAB-1/PicLab1_experiments-P6.html 16F84 to 10F200
; This experiment creates a series of notes.
; A table contains pairs of successive values: the first is the duration
; of the note, the second is the tone.
; These values are used to create loops to generate square waves of the
; desired frequency for the indicated time (based on 4MHz oscillator).
; The table ends with the FFh value, so it can only be used
; for this purpose.
; The table in the example contains the first 6 notes of "Hey Jude" by
; Beatles.
; The series of notes is played continuously.
;
;    PIC            : 10F200/2/4/6
;    Support        : MPASM
;    Version        : 1.0
;    Date           : 01-05-2013
;    Hardware ref. :
;    Author         :Afg
;
; ######################################################################
;
;  Pin use :
;  ----------------
;     10F200/2/4/6 @ 8 pin DIP       10F200/2/4/6 @ 6 pin SOT-23
;
;              |¯¯\/¯¯|                   *¯¯¯¯¯¯|
;           NC -|1    8|- GP3        GP0 -|1    6|- GP3
;          Vdd -|2    7|- Vss        Vss -|2    5|- Vdd
;          GP2 -|3    6|- NC         GP1 -|3    4|- GP2
;          GP1 -|4    5|- GP0             |______|
;              |______|
;
;                               DIP  SOT
;    NC                          1:  Nc
;    Vdd                         2:  5: ++
;    GP2/T0CKI/FOSC4/[COUT]      3:  4: Out speaker
;    GP1/ICSPCLK/[CIN-]          4:  3:
;    GP0/ICSPDAT/[CIN+]          5:  1:
;    NC                          6:  Nc
;    Vss                         7:  2: --
;    GP3/MCLR/VPP                8:  6: MCLR
;
;  [] only 10F204/6
;*******************************************************************
;===================================================================
;           DEFINITION OF PORT USE
;===================================================================
```

```
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-----|-----|-----|--------|
; | MCLR| Spkr|     |        |
;
;#define    GPIO,GP0    ;
;#define    GPIO,GP1    ;
#define    soundpin GPIO,GP2    ; Out speaker
;#define    GPIO,GP3            ; MCLR

;*******************************************************************
;---> Replacing the processor definition
 ; List P = 16F84
 ;#include <p16F84.inc>
; ####################################################################
; Choice of processor
 #ifdef __10F200
        LIST      p=10F200     ; Processor Definition
        #include <p10F200.inc>
 #endif
 #ifdef __10F202
        LIST      p=10F202     ; Processor Definition
        #include <p10F202.inc>
 #endif
 #ifdef __10F204
        LIST      p=10F204     ; Processor Definition
        #include <p10F204.inc>
#define proccomp
 #endif
 #ifdef __10F206
        LIST      p=10F206     ; Processor Definition
        #include <p10F206.inc>
#define proccomp
 #endif

;---> Configuration Line Replacement
; abolishing the hexadecimal form
; __CONFIG 1Bh    ;_CP_OFF & _PWRTE_ON & _WDT_OFF & _RC_OSC

; ####################################################################
;                          CONFIGURATION
;
; No WDT, no CP, MCLR
  __config _CP_OFF & _MCLRE_ON & _WDT_OFF


;---> defines RAM memory in symbolic
  CBLOCK   0x10           ; RAM Area
  stepcntr                ; Step Counter
  noteH                   ; Output Time Counter at Level 1
  Note                    ; Output Time Counter at Level 0
  durcntr                 ; Note Counter
  Temp                    ; temporary
  D1                      ; Temporary for Delay
  D2
  ENDC
```

```
; #####################################################################
; Reset Vector
        ORG 0

SetUp
; #####################################################################
; Additions to match the processor
;---> add internal oscillator calibration
        andlw   0xFE         ; ensures no Fosc/4 also for movwf
        value   OSCCAL       ; incorrect calibration
;---> add comparator disable for 10F204/6 #ifdef proccomp
        movlw   0xF7         ; No MovWF
        Comparator           CMCON0
  #endif
;---> add edit to access GP2 as I/O
;     disable T0CKI movlw
                b'11010111'
        option
;---> Direction accessible only with special TRIS movlw
        opcode  0
        TRIS    GPIO
; #####################################################################

        CLRF    GPIO          ; Clear Out

SetUp1  movlw   0xFF          ;p Step Counter Reset
        movwf   stepcntr
        Goto    Main

; #####################################################################
;            TABLES AND SUBROUTINES (in the first 256 bytes)
; Lookup table by duration/note
Table   addwf   02h,1
        retlw   0xA8          ; Duration - 168 loops
        retlw   0x5B          ; G - 392Hz 1.27mS HIGH,LOW  91 loops
        retlw   0xFA          -
        retlw   0x6B          ; Duration - 250 loops       107 loops
        retlw   0x46          ; E - 330Hz 1.51mS HIGH,LOW
        retlw   0x6B          -
        retlw   0x54          ; Duration - 70 loops
        retlw   0x5B          ; E - 330Hz
        retlw   0x5E          ; Duration - 84 loops
        retlw   0x51          ; G - 392Hz                  - 81 loops
        retlw   0xFC          ; Duration - 94 loops
        retlw   0x7A          ; A - 440Hz - 1.13mS         - 122 loops
        retlw   0xFF          HIGH,LOW
        retlw   0xFF          ; Duration - 252 loops
        retlw                 ;D - 292Hz - 1.71mS HIGH,LOW
        retlw                 ; End of table
        retlw                 ; End of table
        retlw
        retlw
        retl
        retlw

; Ritardo 10us
Delay   Nop                  ; 1uS
```

```
nop                     ; 1uS
nop                     ; 1uS
nop                     ; 1uS
nop                     ; 1uS
nop                     ; 1uS
retlw   0               ; 2uS + 2us call
```

```
; 3ms Delay
Delay2 call     Delay
        decfsz d1,f
         goto   Delay2
        retlw   0

; 250ms delay
Delay3 nop
        decfsz d1.1
         goto   Delay3
        decfsz d2.1
         goto   Delay3
        retlw   0

;------------------------------------------------------------------------
Main    incf    stepcntr,f ; Step Counter+1
; Load Duration from Movf
        Table   stepcntr,w
        Call    Table       ; Table by Movwf
        Duration            durcntr   ; Save to
        Counter
; if =FF is fine
        movwf   Temp
        movlw   0xFFh
        XORWF   temp,w
        skpnz               ; If you are
                            different, continue
         Goto   Main4       ; if FFh end of song

; Load Note from Table
        incf    stepcntr,f ;
stepcounter+1 Main1 movf  stepcntr,w
        Call    Table       ; Load Note
        movwf   noteH       ; and save in movwf
        counters            Note

; Play Note
Main2   Bsf     soundpin
        Call    Delay       ; time at high level
        decfsz noteH,f      ;each loop = 14us
         Goto   Main2
Main3   Bcf     soundpin
        Call    Delay       ; time at low decfsz
        noteL,1
         Goto   Main3
        decfsz durcntr,f ; End of Known Duration?
         Goto   Main1       ; No - Charging BCF Frequency
        Value   soundpin    ; Yes - End of Note

; End Note - Short Pause
        Call    Delay2      ; 3x3ms between
        call notes          Delay2
        Call    Delay2
        Goto    Main        ; next note

; End of Song - Long Pause
Main4   Call    Delay3      ; 3x250ms before the
```

```
call    Delay3     ; Repeating the Song
```

```
        Call     Delay3
; Restart Song
        Goto     SetUp1

        END
```