# Customisation in cross-assembler design for the PIC16F872

**M. Collier**

*Department of Electronic Engineering*

*National University of Science and Technology*

*Bulawayo, Zimbabwe*

*collier1942@yahoo.co.uk*

**Abstract**

The paper describes the development of a cross-assembler known as MCASM which enables programming of the PIC16F872 in a language similar in style to that of the Intel ASM51. The work was instigated by the needs of students in the National University of Science and Technology for a cheap and versatile means of assembly for this processor. Customised features of the software are described.

## 1. INTRODUCTION

The background to this development is the nature of the embedded systems curriculum for Electronic Engineering students at NUST. Two types of processors are taught and used, these being the Intel 8051 family [1, 2] and the Microchip PIC series. To avoid the confusion caused by learning different assembly languages a policy of programming in a style broadly compatible with Intel syntax has been adopted [3].

However the native assembly language utilised by Microchip has a unique form with a different logic from that of Intel. The PIC picocontrollers use instruction pipelining to speed operation, resulting in a RISC instruction set and overlapped instruction execution. Each instruction is stored as a single word in Microchip language and normally takes one machine cycle to execute. Thus a cross-assembler has been developed which uses op-codes similar to Intel while generating hexadecimal code suitable for PIC devices.

## 2. THE NEED FOR A CROSS-ASSEMBLER

### 2.1 The Assembly Process

The action of an assembler is shown in Figure 1 where direct conversion between assembly language words and object code hex takes place.

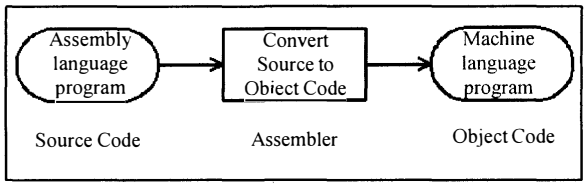


Fig. 1: The assembly operation

### 2.2 The cross-assembly algorithm

When the source program is not written in the native assembly language of the microprocessor the operation entails language conversion, and therefore involves a two-step process, as shown in Figure 2.

The MCASM cross-assembler takes the source file and then codes each instruction line into the 14-bit number used by the picocontroller. To achieve this, the MCASM program correlates source instructions with Microchip opcodes, if necessary creating more than one word of object code to represent a single source instruction.

In addition to the code conversion process, MCASM has also been customised to provide features that experience has shown will improve the speed and efficiency of programming.

### 2.3 Intel and Microchip opcodes

Intel [4] followed the basic language syntax which has evolved from early devices such as the Z80 and Intel 8080. Fundamentally its form is:

**<opcode> <destination address> <source address>**

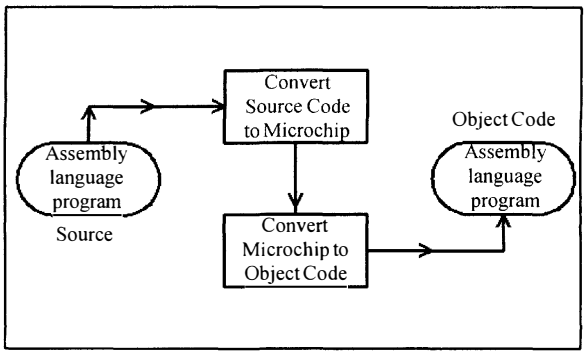


Fig. 2: The Cross-Assembly operation

Microchip devised an instruction set largely incompatible with the Intel format, and taking the form:

**<opcode> <source or destination address> <mode number>**

### 2.4 Building on Parallax

Two decades ago the Parallax company produced an assembler for the PIC16C84 [5] which is an earlier and smaller version of the PIC16F872. The syntax was similar to Intel, and as a result was used in NUST for programming of 16F84 picocontrollers. This chip was housed in an 18-pin package and provided 13 bits of digital I/O. The newer offering, the PIC16F872 [6], has 28 pins and provides 22 lines of digital I/O and 5 analogue-to-digital ports. However when the larger PIC was introduced, there was no cheap, suitable assembler available for it. Thus MCASM was developed, following in general terms the style of the Parallax software.

## 3. CUSTOMISED FEATURES OF THE MCASM ASSEMBLER

### 3.1 Preset equates

The PIC16F872 has a much larger range of special function registers than its smaller brother. Thus the MCASM assembler has been configured to recognise the standard mnemonics both for byte registers and individual bits.

### 3.2 Automatic page switching

One unusual feature of the RAM memory of the PIC is that it is divided into pages of 128 bytes. Some of the Special Function Registers (SFRs) are reflected into other pages, but not all of them. Therefore it is necessary to select the page before executing each instruction, and this is done by writing two bits into one of the SFRs. In the smaller PIC16F84 there are only two pages (0 or 1), and most instructions use only the first page, thus making it unnecessary for the programmer to change pages frequently. However, the PIC16F872 uses four pages (0 - 3), shown in Figure 3, which implies a considerable programming overhead in switching between pages.

For example, the code to write successively to two registers in different pages could take the following form:-

```
clrb rp0          ;  Set to page 1
setb rp1
mov eecon2, #0f5h  ; EECON2 in page 1
clrb rp0          ;  Set to page 0
```

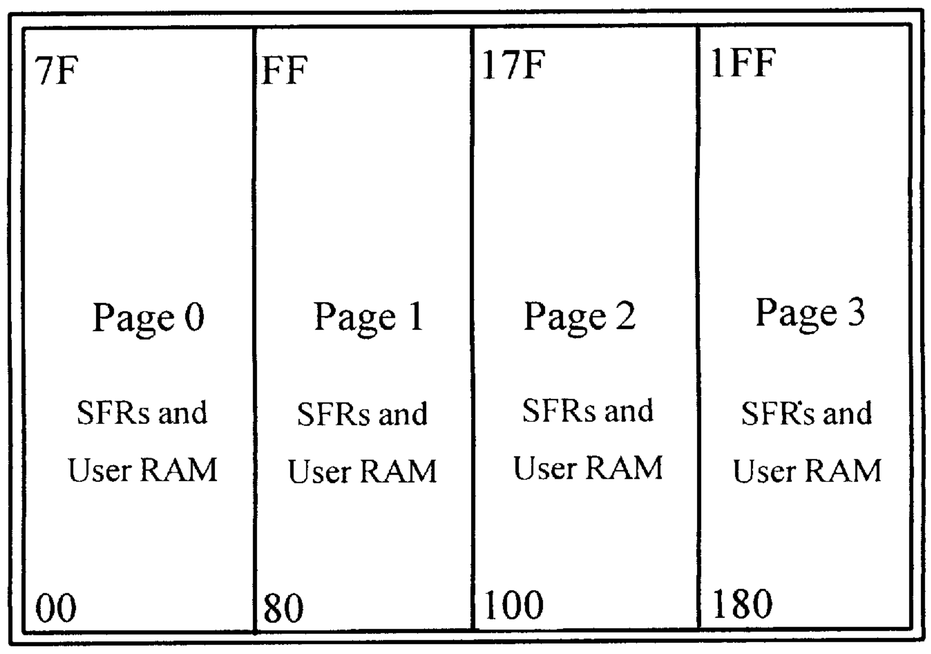


Fig. 3: RAM pages in the PIC16F872

```
clrb rp1
mov status, #5dh  ; STATUS in page 0
```

Thus a programmer needs to remember the location of all the registers in the page structure. To overcome this impediment to fast coding, MCASM has an option to generate the page switching instructions automatically and insert them into the code before assembly. This facility is enabled by a switch on the command line.

### 3.3 Port direction setting

The PIC sets data directions by reading the bits written into a set of registers called TRISA, TRISB, TRISC (meaning "tristate"). However Microchip has peristently warned that the use of such registers may be withdrawn in future versions of their chips. Therefore a standard instruction has been utilised in the MCASM instruction set which can be tailored in future versions of the cross-assembler to meet hardware changes.

The use of "!" to indicate the port direction function enables a unique form of instruction to be used. For example, the following will set Port A so that the low nibble is an input while the high nibble is an output:-

```
mov !PortA, #0fh
            ; Set low nibble I/P; high nibble O/P
```

### 3.4 Operations with double direct addressing

Most of the native PIC operations require the working register as either the source or destination of the data. Thus the movement of data from one memory location to another cannot be achieved with a single Microchip instruction. For this reason the cross-assembler will, if necessary, convert a single source instruction into two machine code operations.

## 4. PITFALLS OF CROSS-ASSEMBLING

### 4.1 Increase of operation execution time

It is traditional to program delay subroutines by calculation of loop duration based on instruction times. The PIC executes all native operations within one machine cycle, with the exception of skips and jumps. When a program is cross-assembled it is no longer possible to assume that instructions execute in a single cycle, and care must be taken in making assumptions about timing.

### 4.2 Invisible use of Working Register

When an MCASM instruction is actually executed as two Microchip instructions it is possible that one of the latter is using the working register (accumulator) for temporary storage between the operations. The programmer must therefore be aware of invisible changes to the value in the register.

For example, the following code fragment will not produce the obvious result, since the final contents of W will be the contents of location 32h, and not the number 56h. This is illustrated in Figure 4.

```
mov w, #55h     ; Move 55h into W
mov 12h, 32h    ; Move 32h to 12h
inc w           ; Increment W - NOW RANDOM!
```

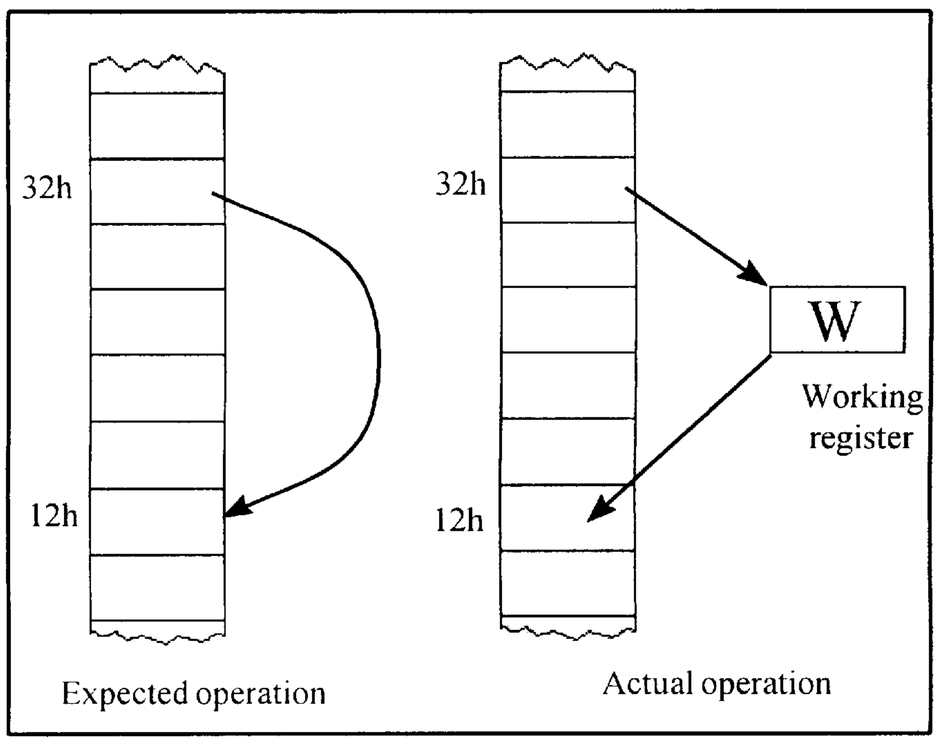


Fig. 4: Use of W in direct move instructions

### 4.3 Dangers of the skip instruction

Microchip has incorporated a skip instruction into its native set. However when a program is cross-assembled it may be that a single MCASM instruction may assemble to a pair of machine code instructions. Thus a skip instruction may only jump over one of these and then execute the other, causing confusion in program control. For this reason the skip instruction has been eliminated from the instruction set for MCASM.

## 5. OPERATIONAL TESTING

The MCASM assembler has passed through various stages of beta testing, in which a considerable number of bugs and improvements have been handled. Student projects in the department of Electronic Engineering at NUST make extensive use of picocontrollers for a wide range of applications [7]. The feedback from this activity has been invaluable in detecting and eliminating errors. These students have also suggested further "wouldn't it be nice" features.

The assembler is written in C language [8] to provide a high level of portability between machines and operating systems. This is partly due to the range of equipment and hardware in use in Zimbabwe, and consequently allows the product to be run on systems from MS-DOS to Windows 2000.

## 6. CONCLUSION

Advantages of a locally-designed product are that it can be easily maintained and improved, import-substitution can be achieved, and local skills can be honed. The MCASM assembler described here is a small example of this principle.

## 7. REFERENCES

[1] Yeralan S., & Ahluwalia A., Programming and Interfacing the 8051 Microcontroller, Addison Wesley, USA, 1995.

[2] Ayala K.J., The 8051 Microcontroller - Architecture, Programming and Applications, West Publishing Co., USA, 1991.

[3] Collier M., An Introduction to Microcontrollers and Picocontrollers, NUST Lecture Notes Series, No.6, NUST, Zimbabwe.

[4] 8-bit Embedded Microcontrollers, Intel Corporation, USA, 1990.

[5] PIC16F84 Data Sheet, Microchip Corporation, USA, 2000.

[6] PIC16F872 Data Sheet, Microchip Corporation, USA, 2000.

[7] Sibanda M. & Collier M., "The opportunities afforded by embedded computer systems for monitoring and control of industrial processes in less industrialised countries", International Journal of Engineering, Iran, Vol. 15, No. 4, pp.379-384, 2002

[8] Power C, Mix Software, USA, 1988.