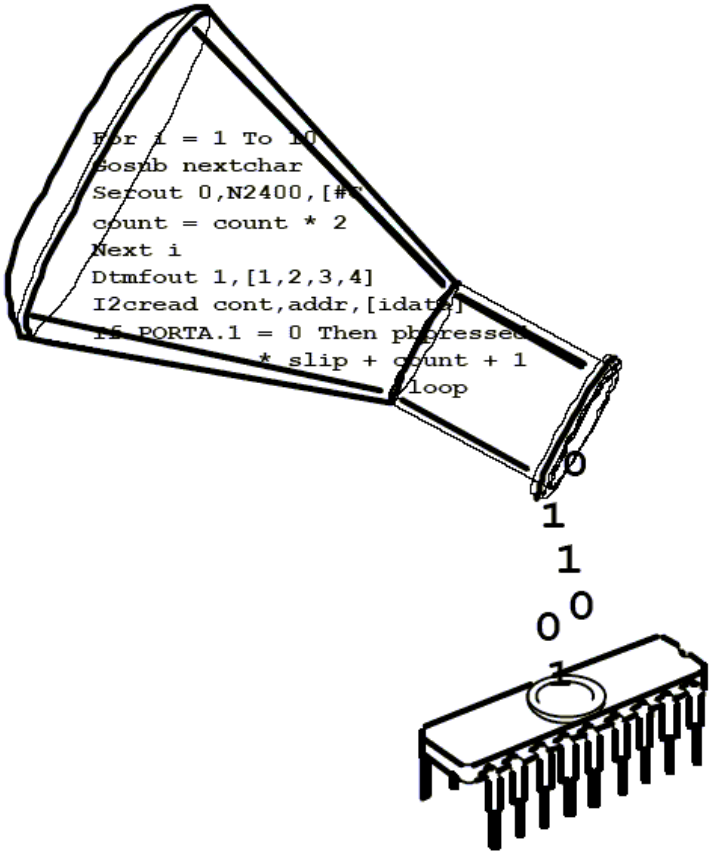
PicBasic Pro Compiler



micro Engineering Labs, Inc.

TABLE OF CONTENTS

1. 도입

PicBasic Pro 컴파일러(PBP) 는 Microchip Technology 사의 강력한 PIC 마이크로 컨트롤러(MCU)를 빠르고 쉽게 프로그램 할 수 있도록한 컴파일러입니다. 영어표기와 유사한 BASIC 언어는 Microchip 의 어셈블리 언어 보다도 읽고 쓰기가 쉽습니다.

PicBasic Pro 컴파일러는 "BASIC Stamp II" 과 비슷하며 BASIC Stamp I 과 II 의 많은 라이브러리와 함수를 가지고 있습니다. PicBASIC Pro 는 컴파일러 언어이므로 BASIC Stamp 보다도 매우 빠르게 동작하며 더 큰 프로그램 용량이 가능합니다.

PicBASIC Pro 컴파일러의 초기버젼은 BS1 을 기초로하여 제작되었으나 BASIC Stamp 와 정확한 호환성을 유지하지 않습니다. 언어의 전반적인 성능향상을 결정하여 조건 판단문이 개선되었습니다. BASIC STAMP 는 IF..THEN (GOTO)만 지원하지만 PicBASIC Pro 에서 IF..THEN ..ELSE .. ENDIF 가 가능하게 되었습니다. 이것은 C 언어와 같은 구조적인 IF 문 사용을 가능하게 한것입니다.

PicBASIC Pro 는 4MHz 로 동작하는 PIC16F84-04/P 가 디폴트(지정하지 않았을때 사용함)로 설정되어 있습니다 2 개의 22pF 세라믹 캐패시터, 4MHz 크리스탈, /MCLR 핀을 +5V 에 연결하는 풀업 저항등 최소한의 부품이 필요합니다. 16F84 와 다른 많은 PIC 마이크로 프로세서는 4MHz 와 다른 오실레이터 주파수를 사용할 수 있으며 PICBASIC Pro 컴파일러를 사용할 수 있습니다.

1.1. PIC 마이크로 MCU

PicBasic Pro 컴파일러는 8 핀 부터 68 핀까지 그리고 A/D 컨버터, 하드웨어 타이머, 시리얼 포트등 다양한 PIC 컨트롤러의 내장 기능을 지원합니다.

현재 PicBASIC Pro 컴파일러는 12 비트 코어, 14 비트 코어 그리고 PIC17Cxxx 와 PIC18Xxxx 의 16 비트 코어 시리즈를 지원합니다. 초기의

12-비트 코어 PIC 마이크로 MCU 는 작은 스택 깊이와 작은 코드 페이지 사이즈 때문에 제한된 기능을 지원합니다. 최신 PIC 마이크로 MCU 의 사용 여부는 README.TXT 를 참고합니다.

PicBASIC Pro 컴파일러와 함께 일반적인 용도의 Pic 마이크로 MCU 시스템을 개발하기 위하여 PIC12F675, 16F628, 16F84, 16F876, 16F877, 18F252 그리고 18F452 를 선택할 수 있습니다. 이들 마이크로 컨트롤러는 플래쉬 기술을 사용하여 빨리 지워지며 재 프로그램 기능으로 빠른 프로그램과 디버깅이 가능합니다. 프로그램 소프트웨어에서 마우스 클릭으로 Pic 마이크로 MCU 를 여러번 지우거나 프로그램 하는것이 가능합니다. 다른 Pic 마이크로 MCU 인 16C55x, 16Cxx, 16C7xx, 16C9xx, 17Cxxx 그리고 18Cxxx 시리즈는 원타임 프로그래머블(OTP)이거나 수정 윈도우가 부착된 Pic 마이크로 MCU 는(JW) 자외선을 수십분 쪼이면 지워지므로 재 프로그램이 가능합니다.

PIC12F6xx, 16F62x, 16F8x, 16F87x 그리고 18Fxx 디바이스는 64 에서 1024 바이트까지 비 휘발성 메모리를 가지고 있으며 프로그램 데이터 또는 다른 파라메터를 Power 가 Off 되었을 때에도 저장할 수 있습니다. 이 데이터 영역은 PicBASIC Pro 컴파일러의 READ 와 WRITE 커맨드에 의하여 간단히 사용할수 있습니다.(프로그램 코드는 Power On 또는 Off 에 관계없이 영구적으로 코드 영역에 저장되어 있습니다.)

초기 프로그램 테스트용으로 플래쉬 Pic 마이크로 MCU 를 사용하면 디버깅 과정이 빠르게 됩니다. 프로그램의 메인루틴의 동작이 안정화 되면 PIC 마이크로 MCU 가 더 많은 기능을 가지며 컴파일러에 의하여 광범위하게 사용될 것입니다.

이 매뉴얼에서 많은 PIC 마이크로 MCU 에 대하여 설명하지만 PIC 마이크로 MCU 데이터 시트나 Microchip Technology 의 CD-ROM 으로 부터 정보를 얻는 것이 필요합니다.

1.2. 본 설명서에 대하여

이 매뉴얼은 BASIC 언어의 모든 사항을 설명하지 않습니다. PicBsic Pro 컴파일러의 명령에 대하여 예제와 함께 어떻게 이용하는 것인가를 설명합니다. BASIC 언어를 사용해본 경험이 없다면 다른 관련 서적을 참고하시기 바랍니다. BASIC 은 사용하기 쉽도록 만들어진 언어 입니다. 어떻게 동작하는지 알기 위하여 간단한 명령을 실험해 보기 바랍니다.

다음 섹션에서 Pi Basic Pro 컴파일러의 설치하는 것과 처음으로 프로그램을 기술하는 것에 대하여 설명합니다. 그리고 컴파이러의 여러가지 선택사양에 대하여 설명할 것입니다. 기본적인 프로그램 규칙과 PicBasic Pro 컴파일러의 명령어를 자세히 설명한 레퍼런스가 있습니다. 레퍼런스에서는 모든 명령의 기본 형식과 예제와 함께 설명됩니다.. 큐어리 브라켓 { } 은 선태적인 파라메터임을 표시합니다.

마지막 부분에서 고급 프로그래머를 위한 컴파일러의 내부동작에 관련된 정보가 있습니다.

1.3. 샘플 프로그램

예제 프로그램은 SAMPLES 디렉토리에 저장되어 있습니다. 더 많은 예제프로그램이 microEngineering Labs, Inc 웹사이트에 있습니다.

2. 시작

2.1 소프트웨어설치

Pic Basic Pro 파일은 자기-압축해지형 파일로 디스켓에 들어 있습니다. 사용하기 전에 하드디스크에서 압축을 풀어야 합니다. 이 것은 DOS 또는 Windows 에서 가능합니다. DOS 에서 압축 해제하면 하드디스크에는 PBP 서브 디렉토리가 생깁니다. 또는 다른 이름으로 할 수 있습니다.

md PBP

DOS 프롬프트에서 디렉토리를 변경합니다.

cd PBP

PicBASIC Pro 프로그램이 담긴 배포 디스켓이 드라이브 A: 에 있다고 가정할 때 PBP 서브디렉토리로 압축 해제 합니다.

A: \pbpxxx - d

여기서 xxx 는 디스켓의 적힌 컴파이러 버전 번호입니다. 옵션 - d 를 커맨드에 마지막에 첨부하는 것을 잊지 말야야 합니다.

다른 방법으로 INSTALL.BAT 을 비슷한 스텝으로 실행할 수 있습니다. 만약 PBP 디렉토리가 이미 있다면 에러 메시지가 나오며 설치를 계속합니다. FILES 와 BUFFERS 가 CONFIG.SYS 파일에서 최소 50 이상으로 설정하여야 합니다. 얼마나 많은 FILES 와 BUFFERS 가 이미 시스템에 사용되어 있는 가에 따라서 더 큰 수를 배정하는 것이 필요할 것입니다.
PicBASIC Pro 프로그램을 Windows 에서 실행하려면 Windows 의 Start 메뉴의 Run Box 에서 A:\INSTALL.BAT 을 실행합니다.
압축 해제하는 파일에 대하여 더 자세한 정보는 README.TXT 를 참고합니다.

2.2 첫번째 프로그램

PicBASIC Pro 컴파일러를 사용하기 위하여는 소스 프로그램을 만들기 위한 텍스트 에디터나 워드 프로세서가 필요합니다. EPIC Plus PICmicro 프로그래머와 같은 PICmicro MCU 프로그래머와 PicBASIC Pro 컴파일러가 필요합니다. 물론 이 모든 것이 잘 동작되는 PC 가 있어야 합니다.
먼저 BASIC 소스 프로그램을 만듭니다. 일반적인 Text 편집기면 모두 가능합니다. DOS 에서 EDIT 명령으로 편집하거나 Window 에서 NotePad 를 사용하면 됩니다. 소스 프로그람은 확장자가 .BAS 가 되도록 합니다.(그렇게 하지 않아도 좋습니다) 소스 프로그램은 순수한 ASCII 문자이어야 합니다. 어떠한 특수코드나 양식용 제어문자를 포함하면 않됩니다. 대부분의 편집기에서 Pure DOS 나 ASCII Text 모드로 저장하면 됩니다. 다음의 예제프로그램은 PIC 마이크로 컨트롤러를 시험하기위한 좋은 예제입니다. 직접 타이핑하거나 PicBASIC Pro 컴파일러에 포함된 SAMPLES 서브 디렉토리로부터 복사하여 사용해도 됩니다. 이 파일은 BLINK.BAS 입니다. BASIC 소스 파일은 PBP.EXE 가 있는 동일한 디렉토리에서 만들어지거나 옮겨야 합니다.

```
' Example Program to blink an LED connected to PORTB.0
' about once a second

loop:   High PORTB.0          ' Turn on LED
        Puse 500              ' Delay for .5 seconds
        Low PORTB.0           ' Turn off LED
        Pause 500             ' Delay for .5 seconds
Goto loop                     ' Go back to loop and blink
                              ' LED forever
End
```

프로그램 작성이 완료되었으면 DOS 프롬프트에서 PBP 다음에 파일이름을 타이핑한 후 엔터키를 누르면 됩니다.

PBP blink

컴퓨터는 초기 메세지를 출력할 것이며 진행상황이 표시될 것입니다. 어셈블리 파일(BLINK.ASM)이 먼저 만들어지고 다음으로 에셈블러 프로그램(PM)이 실행되어 최종적인 PIC 마이크로 컨트롤러용 코드(이 경우 BLINK.HEX)가 만들어집니다. 만일 컴파일 에라가 발생하면 소스 프로그램을 수정하여 컴파일을 반복하여야 합니다. 좋은 프로그램을 개발하는 방법으로 전체 프로그램을 작은 부분으로 나누어서 테스트 하는 것입니다. Pic Basic Pro 컴파일러는 특별히 지정하지 않으면 PIC16F84 용 코드를 만듭니다. 다른 PIC 마이크로 컨트롤러를 사용하려면 -p 옵션 선택 스위치를 사용합니다. 예를들어 타겟 컨트롤러가 PIC16F877 일때 다음과 같이 입력합니다.
PBP -p16f877 blink

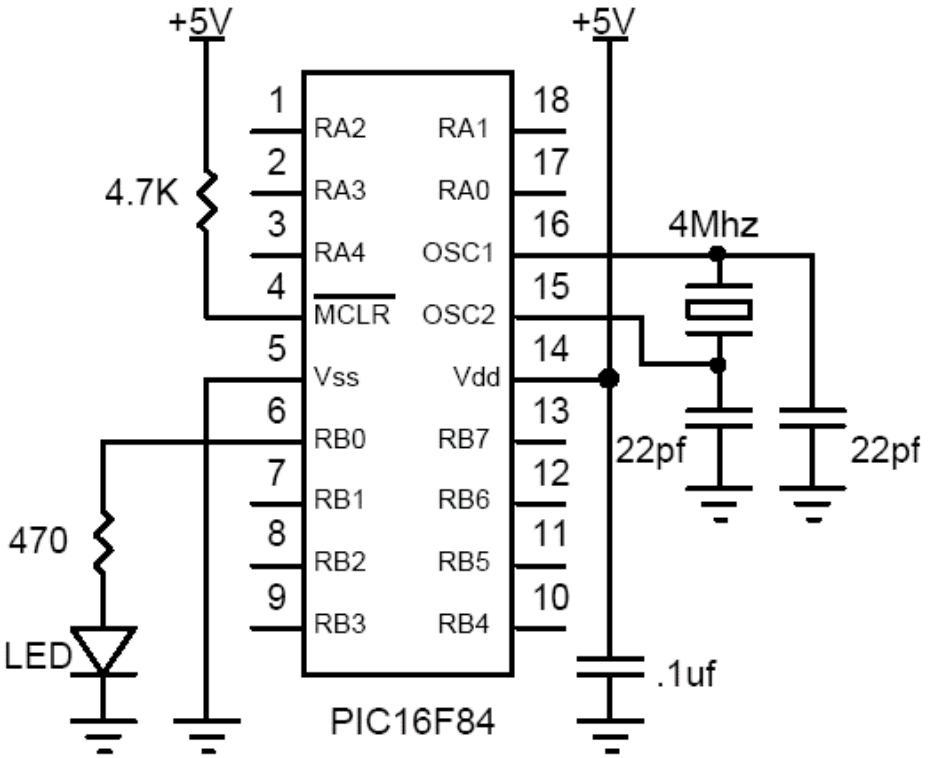
2.3. Program That MCU

컴파일하여 출력된 HEX 파일을 PIC 마이크로 컨트롤러에 프로그램 하는것과 테스트하는 2 단계가 남아 있습니다.
PicBASIC Pro 컴파일러는 모든 PIC 마이크로 컨트롤러 프로그래머에서 사용할수 있는 표준 Intel HEX 파일을 만듭니다. Pic BASIC Pro 는 HEX 파일을 출력하므로 PIC 를 라이팅할 수있는 모든 프로그래머를 사용할 수 있습니다.

2.4. 동작확인

아래에 기본적인 예제 회로가 있으며 Pic 마이크로 MCU 가 정상적으로 동작하기 위한 최소구성입니다. 기본적으로 /MCLR 라인에 풀업저항, 2 개의 캐패시터가 연결된 4MHz 크리스탈 그리고 5V 파워서플라이가 필요합니다. BLINK 프로그램이 동작하는가를 확인해보기 위하여 LED 와 전류제한 저항을 추가합니다. 이 회로를 브레드보드에 구성하였다면 결선을 잘 확인하고

PIC 마이크로 MCU 를 실장합니다. 파워서플라이를 연결하면 Pic 마이크로 MCU 가 동작하며 1 초에 한번씩 반짝거릴 것입니다. 반짝거리지 않는다면 모든 결선상황을 다시 확인하여야 합니다. .



2.5. 동작하지 않을때 고려사항

Pic 마이크로 프로세서를 동작할때 적절한 외부 부품과 치수를 사용하여야 합니다. 다음은 약간의 도움이 될것입니다. /MCLR 핀이 5Volts 에 직접 연결되거나 4.7K 로 5V 에 연결하여야 합니다. 만약 이핀을 연결하지 않으면 레벨이 떠있으므로 제대로 동작하지 않습니다. /MCLR 핀을 전압보호 리셋회로나 간단히 4.7K 를 5V 에 연결하여야 합니다. Pic 마이크로 MCU 는 칩 내부에 Power-On-Rset 회로를 가지고 있습니다. 따라서 보통 외부 풀업 저항으로 충분합니다. PIC 마이크로 MCU 가 전원인가시 동작하지 않는 경우에는 외부 회로가 필요합니다. 이에관한 자세한 정보는 Microchip 데이터 북을 참조합니다.

양질의 크리스탈과 캐패시터를 사용하여야 합니다. 캐패시터 값은 읽기가 어렵습니다. 캐패시터 값이 작거나 크면 발진하지 않습니다. 4MHz 크리스탈에서 세라믹 디스크 캐패시터 22pF(피코 패럿)이면 충분합니다.

파워 서플라이를 확인하여야 합니다. PIC 마이크로 MCU 는 매우 작은 전류를 소비합니다. 파워서플라이는 충분히 필터링 되어 있어야 합니다. 만약 PIC 마이크로 MCU 가 파워서플라이로부터 많은 전류를 공급하는 디바이스를 제어할때 PIC 마이크로 MCU 가 오동작하기에 충분한 글리치가 발생할수 있습니다.

2.5.1. PICmicro Specific Issues

PIC 마이크로 MCU 를 사용할 때 Microchip 데이터를 충분히 이해하는 것은 매우 중요합니다. 일부 디바이스는 핀 동작의 혼란이 예상되는 경우가 있습니다. The PIC16C62x 와 16F62x (the 16C620, 621, 622, 16F627 and 628)는 좋은 예입니다. 이들 PIC 마이크로 MCU 는 PORTA 또는 다른 포트에 아나로그 컴파레이터가 연결되어 있습니다. 칩이 스타트할 때 PORTA 는 아나로그 모드로 설정됩니다. 이것은 PORTA 의 핀 기능이 예상치 않은 동작을 하게 됩니다. 이 핀들을 디지털 모드로 되게 하려면 아래와 같은 명령을 프로그램의 시작위치에 추가 합니다:

```
CMCON = 7
```

아나로그 입력을 가진 PIC16C7xx, PIC16F87x 그리고 PIC12C67x 시리즈 PIC 마이크로 MCU 는 아나로그 모드로 동작하기 때문에 사용하려는 목적에 따라 디지털로 설정하여야 합니다:

```
ADCON1 = 7
```

PIC12F675 와 16F676 은 다른 레지스터에서 설정하여야 합니다.

```
ANSEL = 0
```

중요한 PORTA 의 특한 점은 핀 4 는 출력으로 사용할 때 일반적인 방법으로 사용할 수 없습니다. 다른 모든 핀은 바이폴러형 출력이지만 이 핀은 오픈 드레인 구조로 되어 있습니다. 0 을 출력하면 GND 로 끌어 내리는 기능만 있습니다. 1 을 출력하면 High 로 되는 것이 아니며 플로트 됩니다. 원하는 동작을 기대하려면 이 핀과 5V 사이에 풀업 저항을 추가 하여야 합니다. 저항 값은 1K 에서 4K 까지이며 이 핀과 연결된 회로에 따라 적절한 값을 선택합니다. 이 핀이 입력으로 사용할 때는 다른 핀과 같습니다.

일부 PIC 마이크로 MCU PIC16F627, 628, 873, 874, 876 그리고 877 은 저전

압 프로그래밍(low-voltage programming)을 지원합니다. 이 기능은 PORTB 핀의 하나 이상이 Low 로 다운되어 있지 않다면 이상한 동작을 합니다. 이것을 방지하는 방법은 PIC 마이크로 MCU 가 프로그램 될때 Low-Voltage-Programming 을 하지 않는것입니다.
PIC 마이크로 MCU 의 모든 핀은 파워 업시 입력상태가 됩니다. 이 핀을 출력상태로 하려면 사용하기 이전에 출력 상태로 지정하여야 합니다. 또는 PicBASIC Pro 커맨드를 사용합니다.
특정 PIC 마이크로 MCU 는 Microchip 의 데이터를 확인하도록 합니다.
PIC17Cxxx 디바이스는 PORTA 에 데이터 방향설정 레지스터(TRIS)가 없습니다. 그러므로 I2CREAD 와 I2CWRITE 와 같은 TRIS 와 관계있는 명령어는 PORTA 와 사용할 수 없습니다.

PIC12C5xx, 12CE5xx, 12C67x and 12CE67x 디바이스의 포트 핀은 GPIO 입니다. TRIS 레지스터의 이름은 TRISIO 입니다.

```
GPIO.0 = 1
TRISIO = %101010
```

PIC12C5xx 와 12CE5xx 디바이스에서 핀 GPIO.2 는 TRIS 레지스터 설정에 관계없이 입력으로 설정되어 있습니다. 이 핀을 표준 I/O 핀으로 사용하려면 프로그램 시작에 아래의 라인을 추가하여야 합니다.

```
OPTION_REG.5 = 0
```

위의 예에서 알수 있듯이 모든 PIC 마이크로 MCU 의 OPTION 레지스터 이름은 OPTION_REG 입니다.
어떤 PIC 마이크로 MCU 는 비 휘발 성 메모리이며 I2C 로 구동하는 내장 EEPROM 을 가지고 있습니다. 12CE67x 와 16CE62x 는 시리얼 EEPROM 을 내장하고 있으며 **READ** 나 **WRITE** 명령을 사용할수 없습니다. 대신 **I2CREAD** 와 **I2CWRITE** 를 사용합니다.
PIC12C671, 672, 12CE673 와 674 는 RC 오실레이터를 내장하고 있습니다. 이들 디바이스는 코드 영역의 마지막에 오실레이터 캘리브레이션 팩터를 포함하고 있습니다. 내장 오실레이터는 이 위치에서 데이터를 읽어 OSCCAL 레지스터로 옮겨서 정확하게 됩니다. 두개의 **DEFINE** 이 프로그램 시작에서 자동적으로 이 일을 실행하도록 만듭니다.

```
DEFINE OSCCAL_1K 1      ' Set OSCCAL for 1K device
DEFINE OSCCAL_2K 1      ' Set OSCCAL for 2K device
```

이 두 개의 DEFINE 중 하나를 PicBASIC 프로그램 초기에 설정하면 OSCCAL 이 설정됩니다.
만약 UV 소거형 PIC 마이크로 MCU 가 사용되었다면 이 값을 읽을수 없습니다. 이 두개의 DEFINE 이 지워졌다면 무한 프로그램 루프에 빠져있을 것입니다, 소거된 PIC 마이크로 MCU 는 프로그램 시작 위치에 아래의 라인을 추가 합니다.

```
OSCCAL = $a0 ' Set OSCCAL register to $a0
```

$a0 는 단순히 사용 예 입니다. 소거된 특별한 PIC 마이크로 MCU 에서 실제의 OSCCAL 값을 읽는 것이 필요합니다. PicBASIC Pro 는 가능하다면 PIC12C5xx 와 12CE51x 디바이스에서 OSCCAL 값을 자동으로 읽습니다. 이 두 디바이스에서 위의 DEFINE 을 사용할 필요는 없습니다.
다른 PIC 마이크로 MCU 의 OSCCAL 에 관한 정보는 Microchip 의 데이터를 확인합니다.

2.5.2. Code Crosses Page Boundary Messages

많은 PIC 마이크로 MCU 는 2K 나 8K 의 분리(segment)된 코드 페이지 메모리를 가지고 있습니다. 큰 파일이 컴파일, 어셈블될 때 첫 페이지 이상이 필요할 것입니다. 각 페이지가 사용될 때 어셈블러 PM 은 특정 영역에서 코드가 교차된 것을 알려줍니다. 이것은 정상적인 것이며 알람이 발생하지 않습니다. 단지 **BRANCH** 명령에 주의 하여야 합니다. 만약 **BRANCH** 가 다른 세그먼트의 라벨을 지시하면 정상적으로 동작하지 않습니다. 대신 어느 세그먼트의 라벨도 참조 가능한 **BRANCHL** 을 사용합니다.

2.5.3. Out of Memory Errors

대형 PicBASIC Pro 소스 프로그램을 PC 에서 컴파일 할때 메모리를 점유합니다. Out of Memory 에러가 발생하면 FILES 와 BUFFERS 를 수정합니다. PBPW.EXE 는 Windows 95, 98, NT 그리고 2000 에서 사용가능한 메모리를 이용하여 컴파일 합니다. 이들 32-비트 Windows 환경, MPLAB 또는 다른 Windows IDE 에서 DOS Shell 동작하는 것을 기억하여야 합니다. DOS 커맨드라인에서 Windows 버젼의 PBP 를 실행하는 것은 PBP 를 PBPW 로 대치하기만 하면 됩니다.

```
PBPW blink
```

2.6 코드 스타일

해독하기 쉬우며 관리하기 편한 코드 작성은 하나의 예술입니다. 몇가지 간단한 기교가 도움이 될 것입니다.

2.6.1. 주석문

많은 주석문을 사용합니다. 비록 지금은 작성한 코드의 기능이 완벽해 보일지라도 다른 사람이 보았을 때나 나중에 보더라도 무었을 하려 했는지 알수 있도록 한는것이 좋습니다. BASIC 소스에서 주석문이 많은 공간을 차지 하더라도 PIC 마이크로 MCU 에는 추가적인 공간을 점유하지 않으므로 자유롭게 사용할 수 있습니다. 주석문은 프로그램이 무었을 하는것인지 알수 있게 하는것이 유용합니다. 에를들어 "Set Pin0 to 1" 은 단순히 문법을 설명하므로 이것이 왜 필요한지는 알수 없습니다. "Turn on the Battery Low LED" 가 더 유용할 것입니다.

프로그램 시작 처음과 코드 섹션 앞에서 공간으로 비워두기 보다는 무슨 기능을 하는지 자세히 기술합니다. 프로그램이 시작하기 전에 무었을 하려고 하는지 누가 작성했는지 언제 작성했는지 등을 기록합니다. 리스트 개정 정보와 날짜를 기록합니다. 각 핀이 연결된 상황과 이 프로그램이 어떤 하드웨에 설계되고 동작하는지 기술합니다. 표준 크리스탈 주파수가 아니거나 특별한 컴파일러 옵션을 사용하였으면 주석문으로 표시합니다.

2.6.2. 핀과 변수이름

핀 또는 변수 이름을 Pin0 또는 B1 보다는 구체적인 것으로 정하도록 합니다. 자유로운 주석문과 함께 기능을 나타내는 핀이름과 변수 이름은 프로그램을 이해하기 쉽도록 합니다.

```
BattLED   var      PORTB.0      ' Low battery LED
level     var      byte         ' Variable will contain the battery level

  If level < 10 Then            ' If batt level is low
     High BattLED               ' Turn on the LED
  ENDif
```

2.6.3. 라벨

라벨은 "label1:" 또는 "here:" 보다도 구체적으로 경하도록 합니다. "loop:"과 같은 라벨도 구체적인 것입니다. 보통 라인이나 루틴을 점프하는 것은 무었인가 특별한 것입니다. 최소한 기능이 주석문과 함께 라벨로 표현하여 힌트를 줄수 있도록 합니다.

2.6.4. GOTO

너무 많은 GOTO 를 사용하지 않도록 합니다. GOTO 는 가능한 사용하지 않도록 하여야 합니다. 코드를 논리적으로 나누고 너무 많이 점프하지 않도록 합니다. GOSUB 가 도움이 될것입니다.

3. 컴맨드 라인 옵션

3.1 사용법

PicBasic Pro 컴파일러는 DOS 명령줄에서 다음과같은 명령 형식에 의하여 실행됩니다.

PBP Option Filename

옵션을 주지 않거나 여러개의 옵션을 지정된 파일과 같이 PBP 에 줄수 있습니다. 옵션은 (-) 또는 슬래시(/)기호로 시작합니다. 옵션 자체가 추가적인 정보를 원할 경우 연속하여 기술하면 됩니다. 여러개의 옵션이 기술될 경우 공백문자에 의하여 구분 됩니다. 다음은 여러개의 옵션이 사용되는 예입니다.

```
PBP -p16c71 -ampasm blink
```

BLINK.BAS 파일을 읽어서 어셈블러를 MPASM 으로 사용하고 타겟 프로세서가 PIC16C71 컴파일합니다. "-" 기호가 붙지 않은 첫번째 문자열이 파일이름으로 간주됩니다. 확장자가 없으면 .BAS 가 사용됩니다. PBP 는 컴파일 에러가 없으면 PM.EXE 에셈블러를 기본으로 실행합니다. PBP.EXE 와 PM.EXE 는 같은 디렉토리 내에 있어야 합니다. 컴파일러 에러가 발생하였거나 -s 옵션이 사용 되었다면 어셈블리가 실행되지 않습니다. PBP 가 아무런 옵션이나 파일이름이 주어져있지 않다면 간단한 도움말 메시지가 출력됩니다.

3.2 옵션

3.2.1 옵션 -a

PBP 는 어셈블리 언어로 PM 과 Microchip 사의 MPASM 을 선택하여 사용할 수 있습니다. MPASM 을 사용하려면 (Microchip 사로부터 구입)커멘드라인에서 -ampasm 을 지정 하여야 합니다.

PBP -ampasm filename

커멘드라인에서 아무런 어셈블러의 지정이 없으면 PM 이 사용됩니다. 더 자

세한것은 어셈블리 프로그램을 참고하시기 바랍니다.

3.2.2 옵션 -c

어셈블리언어 소스 파일 내부에 BASIC 소스파일을 주석문 형태로 삽입합니다. 이것은 디버깅 작업을 쉽게하여주며 PicBasic Pro 명령어에 대하여 어떤 어셈블리 언어가 만들어지는가를 배울 수 있습니다.

PBP -c filename

3.2.3 옵션 -h 또는 -?

PBP 가 간단한 도움말을 출력하도록 하여 줍니다. 옵션이나 파일 이름이 지정되어있지 않은 경우에도 같은 도움말이 디스프레이 됩니다.

3.2.4 옵션 -I

PicBasic Pro 소스파일에서 기술된 include 파일의 패스를 지정합니다.

3.2.5 옵션

PicBasic Pro 가 사용할 라이브러리를 선택합니다. 보통 이옵션은 configuration 파일에 마이크로 컨트롤러에 따른 라이브러리가 설정되어 있으므로 사용할 필요가 없습니다.

PBP -lpbpp2 filename

이예는 PicBasic Pro 가 PicStic2 라이브러리를 사용할것을 선택한 것입니다.

3.2.5 옵션 -o

어셈블러에게 옵션을 넘겨 줍니다. 즉 PBP 자신은 -o 의내용과 관계 없습니다. 컴파일후에 어셈블러 프로그램으로 옵션의 내용을 넘겨줍니다. 다음표는 PM 옵션의 일부입니다.

PM 옵션	설명
OD	리스팅, 심볼테이블, 맵파일의 생성
OL	리스팅만 출력

PBP -ol filename

이예는 컴파일이 성공적으로 완료된 후에 .LST 파일이 만들어 집니다. 한번에 더많은 옵션이 어셈블러로 전달될 수 있읍니다. 디스켓에 포함되어있는 어셈블러에 관한 정보를 확인하시기 바랍니다.

3.2.7 옵션 -p

PicBasic Pro 컴파일러는 특별히 다른 지정이 없으면 PIC16F84 를 기준으로 컴파일합니다. 다른 프로세서용 코드를 출력하기 원하면 -p 옵션을 사용하여야 합니다. 예를들어 16C74 가 타겟프로세서인 경우 커맨드 라인에서 다음과같이 입력합니다.

PBP -p16c74 filename

3.2.8 옵션 s

보통 PicBasic Pro 컴파일러는 정상적으로 컴파일이 완료된후 어셈블리 프로그램이 실행됩니다. -s 옵션을 사용하면 컴파일러는 .ASM 파일만을 생성하고 어셈블링을 시작하지 않습니다.

3.2.9 옵션 -v

-v 옵션은 컴파일이 실행되는동안 자세한 정보를 제공하도록 합니다.

4. PicBasic Pro 의 기초

4.1 선언자

선언자는 간단히 표현하면 이름을 뜻합니다. PBP 에서 선언자는 라벨이나 변수이름이 해당됩니다. 선언자는 문자, 숫자, 언더스코어(_)의 배열이 될 수 있습니다. 다만 선언자의 첫번째 문자가 숫자로 시작하면 안됩니다. 선언자는 문자의 대문자, 소문자 구별을 하지 않습니다. 그러므로 label 은 LABEL 과 같습니다. PicBasic Pro 에서 라벨의 길이는 제한이 없지만 처음 32 문자만 인식합니다.

4.2 라인 라벨

GOTo 문이나 GOSUB 문의 위치를 참조하기 위하여 사용합니다. PicBasic Pro 컴파일러는 라인라벨 형식을 사용합니다. 초기의 BASIC 은 모든 라인에 문번호를 사용하였지만 PicBasic Pro 컴파일러는 모든 라인에 번호를 필요로 하지 않습니다. 반면에 PBP 는 라인라벨로 모든 문장이 시작될수 있으며 콜론(:) 기호를 추가하면됩니다.

```
here: Serout 0, N2400, ["Hello, World!", 13, 10]
Goto here
```

4.3 변수

변수는 PicBasic Pro 프로그램에서 임시로 데이타를 저장하는 공간을 의미합니다. 변수는 Var 키워드에 의하여 선언됩니다. 변수는 비트, 바이트, 워드가 될수 있습니다. 모든 변수는 PBP 에의하여 마이크로컨트롤러의 RAM 위치를 지정합니다. 변수를 만들기위한 형식은 다음과 같습니다.

Label1 Var Size {.Modifiers}

Label1 은 키워드를 제외한 모든 선언자가 될수 있습니다. Size 는 BIT, BYTE 또는 WORD 입니다. Modifier 는 옵션이며 변수가 어떻게 사용되는지에 대한 추가적인 정보를 나타냅니다.

```
dog var byte
cat var bit
w0 var word
```

PicBasic Pro 는 미리 정하여진 사용자 변수는 없습니다. 편의상 BASIC Stamps 와 호환성을 유지하기 위하여 "bs1defs.bas"와 "bs2defs.bas"를 사용합니다. 이들중 하나를 PicBasic Pro 프로그램 처음에 포함하면 됩니다.

Include "bs1defs.bas"

또는

Include "bs2defs.bas"

여기에는 BASIC Stamps 에서 사용하는 Pin 에 대한 변수가 정의되어 있습니다. 그러나 이러한 내포(include)파일을 사용하는것보다 프로그래머 자신의 고유한 변수이름을 사용할 것을 권장합니다. 변수의 갯수는 PIC 마이크로컨트롤러의 종류와 사용되는 변수의 양에 따라 다릅니다. PBP 는 자신이 동작하기위하여 24 바이트정도의 최소한의 RAM 메모리를 점유합니다. 복잡한 수식을 처리할때 임시변수를 사용하기 위하여 추가적으로 RAM 메모리가 필요합니다.

4.4 변수의 재지정

변수를 다른 이름으로도 사용할때 var 를 사용합니다. 변수의 내부를 지정할때 유용하게 사용됩니다.

```
fido var dog ' fido 는 dog 의 다른 이름입니다.
b0 var w0.byte0 ' b0 는 w0 의 첫번째 바이트입니다.
b1 var w0.byte1 'b1 은 w0 의 두번째 바이트 입니다.
flea var dog.0    ' flea is bit0 of dog
```

첨부(Modifier)	설명
BIT0 or 0	바이트 또는 워드에서 비트 0 의 재정의
BIT1 or 1	바이트 또는 워드에서 비트 1 의 재정의
BIT2 or 0	바이트 또는 워드에서 비트 2 의 재정의
BIT3 or 0	바이트 또는 워드에서 비트 3 의 재정의
BIT4 or 0	바이트 또는 워드에서 비트 4 의 재정의
BIT5 or 0	바이트 또는 워드에서 비트 5 의 재정의
BIT6 or 0	바이트 또는 워드에서 비트 6 의 재정의
BIT7 or 0	바이트 또는 워드에서 비트 7 의 재정의
BIT8 or 0	워드에서 비트 8 의 재정의
BIT9 or 0	워드에서 비트 9 의 재정의
BIT10 or 0	워드에서 비트 10 의 재정의
BIT11 or 0	워드에서 비트 11 의 재정의
BIT12 or 0	워드에서 비트 12 의 재정의
BIT13 or 0	워드에서 비트 13 의 재정의
BIT14 or 0	워드에서 비트 14 의 재정의
BIT15 or 0	워드에서 비트 15 의 재정의
BYTE0 or LOWBYTE	워드에서 로우바이트의 재정의
BYTE1 or HIGHBYTE	워드에서 하이바이트의 재정의

4.5 배열

배열은 변수선언과 비슷한 방법으로 선언 합니다.

Label1 var Size(배열의 갯수)

Label1 은 키워드를 제외한 선언자입니다. Size 는 BIT, BYTE 또는 WORD 입니다. 배열의 갯수는 얼만큼 위치를 점유할 것인가를 나타냅니다.

```
sharks var byte[10]
fish var bit[8]
```

첫번째 배열의 데이타가 0 이됩니다. fish 배열에서 fish[0] 부터 fish[7] 까지 8 개의 구성원이 됩니다.

배열은 다음과 같은 제한이 있습니다.

크기	요소의 최대수
BIT	128
BYTE	64
WORD	32

4.6 상수

이름이 부가된 상수는 변수와 비슷한 방법으로 선언됩니다. 상수자체를 사용하는것보다 편리합니다. 만일 숫자의 변경이 필요하다면 소스프로그램 내에서만 가능합니다. 변수는 상수로써 저장될수 없습니다.

Label1 CON Constant expression
상수 사용예
mice con 3
traps con mice * 1000

4.7 심볼

심볼은 변수와 상수를 재정의할 수 있습니다. 심볼은 변수를 정의하는데에 사용할수 없습니다. 변수의 선언은 var 로 합니다.
SYMBOL lion = cat ' cat was previously created using VAR
SYMBOL mouse = 1 ' Same as mouse con 1

4.8 숫자 상수

PBP 는 10 진수, 바이너리, 헥사의 3 종류 숫자 상수 형식을 지원합니다. 바이너리 상수는 접두어 '%' 를 사용하며 헥사데시말은 접두어 '$'를 사용합니다. 십진수는 아무런 접두어도 필요치 않습니다.
100 ' Decimal Value
%100 ' Binary value for decimal 4
$100 ' Hexadecimal value for decimal 256
프로그램을 간편히 하기 위하여 한개의 문자가 ASCII 값에 해당하는 숫자로 변환될 수 있습니다. 문자 상수는 반드시 쌍따옴표를 사용하여야하며 반드시 한개의 ASCII 문자만 허용합니다.
"A" ' ASCII Value for decimal 65
"d" ' ASCII Value for decimal 100

4.9 스트링 상수

PBP 는 스트링 조작기능을 지원하지 않습니다.그러나 스트링은 약간의 명령어에 의하여 처리됩니다. 스트링 상수는 하나 이상의 문자로 이루어지며 앞뒤에 쌍따옴표 " 를 사용합니다.
"Hello" ' String (Short for "H", "e", "l", "l", "o")
스트링은 개별적인 문자값의 나열로 처리됩니다.

4.10 핀

PIC 의 입출력핀은 여러가지 방법으로 사용될 수 있습니다. 포트의 이름과 비트의 위치를 나타내는 숫자를 같이 사용하는 것이 좋은 방법입니다.
PORTB.1 = 1 ' Set PORTB, bit 1 to a 1
포트핀의 사용 용도를 쉽게 기억하기 위하여 var 에 의하여 이름을 지정하는 것이 가능합니다.
led var PORTA.0 ' Rename PORTA.0 as led
High Led ' Set led (PORT.0) high

4.11 주석문

PBP 는 문장의 처음에 REM 키워드로 시작하거나 싱글따옴표 (') 이후에 나타나는 모든 문자는 주석문으로 간주합니다. 이반적으로 다른 BASIC 과 달리 REM 은 REMark 의 약자가아닌 독립적인 키워드 입니다. 따라서 REM 자체는 안되지만 REM 으로 시작되는 변수를 만드는것이 가능하므로 주의하여야합니다. (REM 이 정확한 주석문의 시작을 나타내기 위하여 REM 이후에 반드시 공백문자 하나가 오도록 합니다.)

4.12 복합문

논리적으로 유사한 명령은 한줄에 기입하는것이 좀더 간결한 프로그램 방법이 됩니다. PBP 는 동일한 라인에서 여러개의 문장을 콜론(:)기호를 사용하여 구분 합니다.
W2 = W0
W0 = W1
W1 = W2
는 다음과 같습니다
W2 = W0 : W0 = W1 : W1 = W2

4.13 라인 확장문자

PBP 는 한줄의 최대 크기가 256 문자가 들어갈 수 있습니다. 이 보다도 큰 BASIC 문장이 필요한 경우 언더스코어(_) 를 사용하여 연장할 수 있습니다.
Branch B0 , [label0, label1, label2,_
label3, label4]

4.14 파일의 내포(INCLUDE)

다른 BASIC 소스 프로그램을 메인 프로그램에 포함할 수 있습니다. 표준화된 서브루틴, 특정 용도용 데이타 또는 별도로 보관하기 원하는 파일을 나누어서 관리하는 것이 가능합니다.
INCLUDE "modedefs.bas"

4.15 정의 (DEFINE)

클럭 오실레이터 주파수나 LCD 핀 정보와 같은 어떠한 요소는 PBP 에서 설정할 필요가 있습니다. 변경을 원하면 소스프로그램에서 변경하면 됩니다. DEFINE 은 미리 정의된 오실레이터값, DEBUG 핀, 바우드레이트, LCD 핀 위치를 변경할 필요가 있을때 사용합니다. DEFINE 에서 사용되는 문자는 반드시 대문자 이여야 합니다.

DEFINE	BUTTON_PAUSE	50	' Button debounce delay in ms
DEFINE	CHAR_PACING	1000	' Serout pacing in us

DEFINE	DEBUG_REG	_PORTL	' Debug pin port
DEFINE	DEBUG_BIT	0	' Debug pin bit
DEFINE	DEBUG_BAUD	2400	' Debug baud rate
DEFINE	DEBUG_MODE	1	' Debug mode: 0 = True, 1 = Inverted
DEFINE	DEBUG_PACING	1000	' Debug pacing in us
DEFINE	HSER_RCSTA	90h	' Set rcv reg
DEFINE	HSER_TXSTA	20h	' Set transmit reg
DEFINE	HSER_BAUD	2400	' Set baud rate
DEFINE	HSER_EVEN	1	' Use only if even parity desired
DEFINE	HSER_ODD	1	' Use only if odd parity desired
DEFINE	I2C_INTERNAL	1	' Use for internal EEPROM on 16CExxx and 12CExxx
DEFINE	I2C_SLOW	1	' Use for >8mHz OSC with standard speed devices
DEFINE	LCD_DREG	PORTB	' LCD data port
DEFINE	LCD_DBIT	0	' LCD data starting bit 0 or 4
DEFINE	LCD_RSREG	PORTB	' LCD register select
DEFINE	LCD_RSBIT	4	' LCD register select bit
DEFINE	LCD_EREG	PORTB	' LCD Enable port
DEFINE	LCD_BITS	5	' LCD enable bit
DEFINE	LCD_LINES	2	' number lines on LCD
DEFINE	OSC	4	' 3 4 8 10 12 16
DEFINE	OSCCAL_1K	1	' Set OSCCAL for PIC12C671
DEFINE	OSCCAL_2K	1	' Set OSCCAL for PIC12C672

4.16 수식 연산

PBP 는 모든 수식 연산의 계층 구조를 지원합니다. 이것은 연산자간의 우선순위가 있다는 뜻입니다. 곱하기와 나누기 연산은 가산연산과 감산연산 이전에 실행됩니다. 괄호를 사용하여 연산 순서를 정하는 것도 가능합니다.

A = (B + C) * (D - E)

모든 수식 연산은 16 비트 무부호 정수로 처리됩니다.

4.16.1 곱셈 (Multiplication)

PBP 는 16X16 의 곱셈을 지원합니다. * 연산자는 32 비트 결과의 하위 16 비트만 반환합니다. ** 는 상위 16 비트를 반환합니다. 이 두개의 연산자를 사용하여 곱셈의 32 비트 결과를 얻을 수 있습니다.

W1 = W0 * 1000 ' Multiply value in W0 by 1000 and place the result in W1

W2 = W0 ** 1000 ' Multiply W0 by 1000 and place the high order 16 bits (which may be 0) in W2

연산자 */ 는 32 비트 곱셈 결과에서 중간의 16 비트값을 반환합니다.

4.16.2 나눗셈 (Divide)

PBP 는 16 x 16 의 나눗셈을 지원합니다. 연산자 '/' 는 16 비트의 결과 값을 반환합니다. 연산자 '//'는 연산결과의 나머지(remainder)를 반환합니다.

W1 = W0 / 1000 ' Divede value in W0 by 1000 and place the result in W1

W2 = W0 // 1000 ' Divede value in W0 by 1000 and place the remainder in W2

4.16.3 쉬프트 (Shift)

연산자 '<<' 와 '>>' 는 1 부터 15 회까지 Shift Left 와 Shift Right 입니다. 새롭게 진입하는 수는 0 입니다.

B0 = B0 << 3 ' Shift B0 left 3 places (same as multiply by 8)

W1 = W0 >> 1 ' Shift W0 right 1 position and places result in W1 (same as divide by 2)

4.16.4 절대값 (ABS)

ABS 는 숫자의 절대값을 반환합니다. 바이트 변수의 값이 127 보다 큰경우 "256 - 변수값"을 반환합니다. 워드변수의 값이 32767 보다 큰경우 "65536 - 변수값"을 반환합니다.

B1 = ABS B0

4.16.5 COS

COS 는 8 비트의 cosine 값을 반환합니다. 반환값은 2 의 보수(즉 -127 부터 127)형태입니다. 4 분면 웨이브 테이블로부터 값을 가져옵니다. 0 부터 359 도 까지의 일반적인 체계와 달이 0 부터 255 까지의 바이너리 라디안 체계를 사용합니다.

```
B0 = COS B0
```

4.16.6 디코드 비트셋 코딩 숫자(DCD)

DCD 는 주어진 숫자에 해당되는 비트 위치를 셋트한 숫자 값을 반환합니다. 나머지 비트는 0 입니다.

```
B0 = DCD 2 ' Set B0 to %00000100
```

4.16.7 데시말 디지트값(DIG)

DIG 는 십진수의 지정된 디지트 값을 반환합니다.

```
B0 = 123 ' Set B0 to 123
B1 = B0 DIG 1 ' Sets B1 to 2 (digit 1 of 123)
```

4.16.8 최대치, 최소치 (MAX, MIN)

Max 와 Min 은 각각 두수중에서 최대치와 최소치를 구합니다. 일반적으로 변수에 저장되는 값의 크기를 제한 할때 사용합니다.

```
B1 = B0 MAX 100 ' Set B1 to the larger of B0 and 100 (B1 will be between 100 & 255)
B1 = B0 MIN 100 ' Set B1 to the smaller of B0 and 100 (B1 can't be bigger than 100)
```

4.16.9 상위비트위치값 반환

NCD 는 변수에서 상위비트(왼쪽)의 위치값을 반환합니다. 1 부터 16 까지 범위이며 변수값 중에 1 이 하나도 없을경우 0 이 반환 됩니다.

```
B0 = NCD %01001000 ' Sets B0 to 7
```

4.16.10 반전(REV)

REV 는 변수값의 하위비트를 반전합니다 비트 위치는 1 부터 16 까지 입니다.

```
B0 = %10101100 REV 4 ' Set B0 to %10100011
```

4.16.11 사인 (SIN)

SIN 은 8 비트의 sine 값을 반환 합니다. 반환값은 2 의보수(즉 -127 부터 127)형태입니다. 4 분면 웨이브 테이블로부터 값을 가져옵니다. 0 부터 359 도 까지의 일반적인 체계와 달이 0 부터 255 까지의 바이너리 라디안 체계를 사용합니다.

```
B0 = COS B0
```

4.16.12 평방근(SQR)

SQR 은 평방근을 구해냅니다. PicBasic Pro 는 정수에 대한 연산을 지원하므로 결과값은 8 비트 값입니다.

```
B0 = SQR W1 ' Sets B0 to sqare root of W1
```

4.16.13 논리연산자(Bitwise Operators)

Bitwise 연산자는 BOOL 연산 작용을 합니다. 변수의 비트중 일부분을 추출해 내거나 비트를 더하는 조작이 가능합니다.

```
B0 = B0 & %00000001 ' Isolate bit 0 of B0
B0 = B0 | %00000001 ' Set bit 0 of B0
B0 = B0 ^  %00000001 ' Reverse state of bit 0 of B0
```

4.17 비교연산자(Comparision Operator)

비교 연산자는 IF -- THEN 문에서 수식의 값을 서로 비교 검사할때 사용됩니다. 다음과 같은 연산자를 지원합니다.

비교연산자	설명
= or ==	Equal
<> or !=	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

4.18 논리 연산자(Logical Operators)

논리 연산자는 비트연산자와 다릅니다. 논리 연산자는 결과치가 항상 True / False 만 가집니다. 대부분 IF -- THEN 문과 한께 사용되며 다음과 같은 논리 연산자가 지원됩니다.

논리연산자	설명
AND or &&	Logical AND
Or or ||	Logical OR
XOR or ^ ^	Logical Exclusive OR
NOT AND	Logical NAND

NOT OR	Logical NOR
NOT XOR	Logical XOR

If (A==big) AND (B > mean) then run
논리연산의 정확한 순서를 지정하기위하여는 괄호를 사용합니다.

5. PicBasic Pro 의 실행문 레퍼런스

@	한개의 어셈블리 언어코드 삽입
ASM..ENDASM	어셈블리 언어 코드 섹션 삽입
BRANCH	계산된 GOTO (ON..GOTO 와 등가)
BRANCHL	페이지 영역을 넘는 BRANCH (long BRANCH)
BUTTON	지정된 핀에 대라여 디바운스와 자동 반복 입력
CALL	어셈블리 언어의 호출
CLEAR	모든 변수의 초기화 (변수를 0 으로 만든다)
COUNT	핀에 입력되는 펄수의 수를 카운트
DATA	EEPROM 에 저장되는 초기상수값 정의
DEBUG	지정된 핀과 바우드레이트에의한 비동기 직열 출력
DISABLE	ON INTERUPT 처리의 금지
DTMFOUT	지정된 핀에서 touch-tones 생성
EEPROM	내장 EEPROM 의 초기데이타 정의
ENABLE	ON INTERRUPT 처리의 가능상태 설정
END	프로그램 실행을 중지하고 저전력모드 상태로 들어간다.
FOR..NEXT	실행문의 반복실행
FREQOUT	핀에 대하여 2 중까지의 주파수 출력
GOSUB	지장된 라벨위치의 서브루틴 호출
GOTO	지정된 라벨로 실행제어 이동
HIGH	핀을 High 상태로 만든다.
HSERIN	하드웨어 비동기 시리얼 입력
HSEROUT	하드웨어 비동기 시리얼 출력
I2CREAD	I2C 소자로부터 데이터 입력
I2CWRITE	I2C 소자로 데이타 출력
IF..THEN..ELSE	조건부 실행문 제어
INPUT	핀으로부터 데이타 입력
{LET}	변수에 값을 지정
LCDOUT	LCD 에 문자 표시
LOOKDOWN	값을 찾기위한 상수테이블 검사
LOOKDOWN2	값을 찾기위한 상수/변수 테이블 검사
LOOKUP	테이블로부터 상수값의 가져오기
LOOKUP2	테이블로부터 상수/변수값 가져오기
LOW	핀을 Low 상태로 만든다.
NAP	짧은 주기시간동안 프로세서의 동작을 멈춘다
ON INTERRUPT	인터럽트 발생시 BASIC 서브루틴의 실행
OUTPUT	핀을 출력모드로 설정한다.
PAUSE	1mSec 단위의 지연시간 발생
PAUSEUS	1uSec 단위의 지연시간 발생
PEEK	레지스터의 데이타 읽기
POKE	레지스터로 값을 쓰기
POT	지정한 핀의 전위값 읽기

PULSIN	핀에대한 펄스폭측정
PULSEOUT	핀에 펄스를 출력한다.
PWM	핀에 대하여 펄스열을 출력한다.
RANDOM	의사 랜덤(난수)번호 발생
RCTIME	핀의 펄스폭측정
READ	내장 EEPROM 으로부터 데이타 읽기
RESUME	인터럽트 처리가 완료된후에 연속해서 실행
REVERSE	입력핀을 출력핀으로, 출력핀을 입력핀으로 전환
SERIN	비동기 시리얼 입력(BS1 스타일)
SERIN2	비동기 시리얼 입력(BS2 스타일)
SEROUT	비동기 시리얼 출력(BS1 스타일)
SEROUT2	비동기 시리얼 출력(BS2 스타일)
SHIFTIN	동기 시리얼 입력
SHIFTOUT	동기 시리얼 출력
SLEEP	정해진 시간동안 파워 다운모드
SOUND	핀에대하여 화이트노이즈또는 사운드 출력
STOP	프로그램 실행중지
SWAP	두개의 변수값 교환
TOGGLE	핀을 토글상태(현재 출력값 상태반전) 설정
WHILE..WEND	조건 검사값이 참인동안은 계속 반복 실행
WRITE	EEPROM 에 데이터 쓰기
XIN	X-10 입력
XOUT	X-10 출력

5.1. @

@ Statement

PBP 프로그램에서 한 줄의 어셈블리 명령어를 사용할때 @ 를 사용합니다. PicBasic Pro 명령문과 어셈블리 언어를 혼합하여 사용하는것을 가능하게 합니다.

```
i           var       byte
rollme      var       byte
        For i = 1 to 4
@       rfl       _rollme, F      ; Rotate byte left once
        Next i
```

단축키 @는 다른 화일로 되어있는 어셈블리 언어 루틴을 호출할 때에도 사용합니다.

```
@ include "fp.asm"
```

@는 어셈블리 명령어가 사용되기 전에 레지스터 페이지를 0 으로 리셋트합니다. 레지스터 페이지는 @ 명령어로 변경되지 않습니다.
자세한 것은 어셈블리 프로그램을 참고하십시오.

5.2 ADCIN

ADCIN Channel, Var

프로세서에 내장된 아나로그-디지털 컨버터 Channel 의 값을 읽어 Var 에 저장합니다. ADC 값을 직접 읽을 수 있으나 ADCIN 명령을 사용하는 것이 쉽습니다.
ADCIN 명령을 사용하기 이전에 원하는 핀을 입력으로 지정하기 위하여 TRIS 레지스터를 설정하여야 합니다. ADCON1 또는 ANSEL 또한 원하는 핀에 아나로그 입력으로 사용되도록 지정합니다. 어떤 경우에는 데이터 포맷과 클록 소스를 선택하기 위하여 사용합니다. 특정한 칩에대하여 레지스터를 설정하는것은 Microchips 사의 데이터 시트를 참고하시기 바랍니다. 디바이스에따라 8-, 10-, 또는 12- 비트 ADC 가 있습니다. ADCON1 의 상위비트는 결과값을 오른쪽으로 정렬할 것인지 왼쪽으로 정렬할 것인지를 지정합니다.

대부분의 경우 8-비트 결과는 왼쪽정렬(ADCON1.7 = 0)이며 10-과 12--비트 데이터는 오른쪽으로 정렬합니다.(ADCON1.7 = 1).

여러개의 DEFINES 가 사용될 수 있습니다. 디폴트 값은 아래와 같습니다.

```
DEFINE ADC_BITS 8     ' Set number of bits in result (8, 10, or 12)
DEFINE ADC_CLOCK    3 ' Set clock source (rc = 3)
DEFINE ADC_SAMPLUS 50 ' Set sampling time in microseconds
```

ADC_SAMPLUS 는 Channel 을 설정하고 아나로그-디지털 컨버젼을 시작하기까지의 시간입니다. 이것을 샘플링 타임이라고 합니다. 사용가능한 최소의 마이크로세컨드 수는 PAUSEUS 를 위한 최소시간으로 결정됩니다.

```
TRISA = 255    ' Set PORTA to all input
ADCON1 = 0    ' PORTA is analog
ADCIN 0, B0 ' Read channel 0 to B0
```

5.3 ASM..ENDASM

ASM
ENDASM

ASM 과 ENDASM 명령은 PBP 에게 이 두 명령문 사이에는 어셈블리 코드가 놓여진 것이며 BASIC 으로 인식하지 말 것을 알려주는 것입니다. 이 두 명령문을 이용하여 PicBASIC Pro 명령과 함께 어셈블리 명령어를 자유롭게 사용하는 것이 가능합니다.

어셈블리 프로그램 부분의 최대 문자 크기는 8 K 바이트 이내 이어야 합니다. 이것은 만들어지는 헥사 코드 바이트 수가 아니며 주석 문을 포함한 소스 코드의 크기를 의미합니다. 그러므로 8 K 바이트 이상의 어셈블리 소스코드 이상이 필요한 경우에는 ASM .. ENDASM 을 여러 구간으로 나누어서 사용하거나 다른 파일로 만들어 내포(include) 하는 방법을 사용합니다.

ASM 명령은 레지스터 페이지를 0 으로 리셋 합니다. 어셈블리 프로그램 상에서 레지스터 페이지를 변경하여 사용하였다면 ENDASM 문을 사용하기 이전에 반드시 0 으로 환원시켜야 합니다. 자세한 정보는 어셈블리 명령어 부분을 참고하시기 바랍니다.

```
ASM
      bsf     PORTA, 0        ; Set bit 0 on PORTA
      bcf     PORTB, 0        ; Clear bit 0 on PORTB
ENDASM
```

5.4. BRANCH

BRANCH Index, [Label{, Label...}]

BRANCH 명령은 변수 값에 따라서 다른 위치로 점프합니다. 이것은 다른 BASIC 언어에서 On..Goto 명령과 비슷합니다. Index 는 나열된 라벨중에서 하나를 선택합니다. index 값이 0 이면 첫번째 라벨 위치로 , 1 이면 두번째 라벨 위치로 점프합니다. 만약 index 값이 나열된 라벨의 개수와 같거나 크다면 점프는 발생하지 않으며 BRANCH 다음 문장이 실행됩니다. BRANCH 명령에서 최대로 사용 가능한 라벨 수는 255(PIC18Xxxx 에서 256) 입니다.

12 비트 코어와 14 비트 코어 그리고 PIC17Cxxx 디바이스에서 Label 은 BRANCH 명령과 동일한 코드페이지에 있어야 합니다. 만약 동일한 코드페이지가 확실하지 않다면 BRANCHL 명령을 사용합니다.

PIC18Xxxx 디바이스에서 Label 은 상대 점프방식을 사용하므로 BRANCH 명령으로부터 1K 범위 이내에 있어야 합니다. 만약 Label 이 이범위를 넘는경우 BRABCHL 을 사용하여야 합니다.

```
BRANCH      B4, [dog, cat, fish]
' Same as :
' If B4 = 0 Then dog (goto dog)
' If B4 = 1 Then car(goto cat)
' If B4 = 2 Then fish (goto fish)
```

5.5. BRANCHL

BRANCHL Index, [Label1{, Label...}]

BANCH(BREANCH LONG) 은 변수값에 따라 나열된 라벨위치로 점프한다는 기능은 동일합니다. BRANCH 명령과 가장 큰 차이점은 BRANCHL 명령은 다른 코드페이지로 점프가 가능하도록 실행코드가 만들어지므로 BRANCH 명령에서 만들어지는 헥사 코드와 비교하면 2 배 커집니다. 사용되는 PIC 마이크로 프로세서가 2 K 이하의 ROM 이거나 점프거리가 2K 바이트 이내이면 BRanch 명령을 사용합니다. 이것은 BRANCH 명령은 1 개의 코드 페이지인 2 K 바이트 이내에서 점프합니다. Index 는 나열된 라벨중에서 하나를 선택합니다. index 값이 0 이면 첫번째 라벨 위치로 , 1 이면 두번째 라벨 위치로 점프합니다. 만약 index 값이 나열된 라벨의 개수와 같거나 크다면 점프는 발생하지 않으며 BRANCH 다음 문장이 실행됩니다. BRANCH 명령에서 최대로 사용 가능한 라벨 수는 127(PIC18Xxxx 에서 256) 입니다.

```
BRANCH      B4, [dog, cat, fish]
' Same as :
' If B4 = 0 Then dog (goto dog)
' If B4 = 1 Then car(goto cat)
' If B4 = 2 Then fish (goto fish)
```

5.6. BUTTON

BUTTON Pin,Down,Delay,Rate,BVar,Action,Label

디바운스(노이즈, 채터링제거)와 자동 반복기능을 선택적으로 적용하여 Pin 의 데이터를 읽습니다. Pin 은 0 - 15 까지의 상수이거나 번호를 저장하고있는 변수, 또는 PORTA.0 과 같이 포트 핀을 직접 지정합니다.

Down 버튼이 눌려졌을 때의 논리상태(0 또는 1)입니다.

Delay 자동 반복기능이 시작되기 전에 사이클 횟수(0..255)입니다. 만약 0 로 지정하면 디바운스는 적용되지 않거나 자동반복기능이 적용됩니다. 만약 255 라면 디바운스가 적용되며 자동반복기능은 적용되지 않습니다.

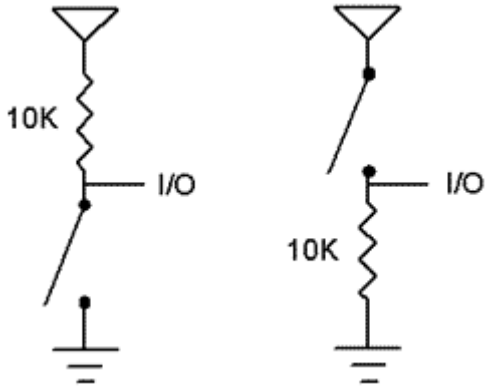
Rate 자동반복 레이트입니다. (0..255)

BVar 바이트 크기의 변수이며 내부적으로 딜레이/반복 카운트 용으로 사용됩니다. 이 변수는 사용되기전에 반드시 0 으로 초기화 되어야 하며 프로그램 다른 곳에서 사용할 수 없습니다.

Action 버튼이 작용할 때의 상태입니다. (0 은 눌러지지 않음 1 은 눌려짐)

Label 라벨이며 Action 이 참이 된 경우 입니다.

BUTTON 은 자동반복기능이 정상적으로 동작하기 위하여 필요합니다.




```
' GOTO notpressed if button not pressed on Pin2
BUTTON PORTB.2, 0, 100, 10, B2, 0, notpressed
```

BUTTON accomplishes debounce by delaying program execution for a period of milliseconds to wait for the contacts to settle down. The default debounce delay is 10ms. To change the debounce to another value, use **DEFINE**:

```
> Set button debounce delay to 50ms
DEFINE BUTTON_PAUSE 50
```

Be sure that BUTTON_PAUSE is all in upper case.

In general, it is easier to simply read the state of the pin in an **IF..THEN** than to use the **BUTTON** command as follows:

```
If PORTB.2 = 1 Then notpressed
```

Example program:

```
INCLUDE "modedefs.bas"      ' Include serial modes
SO      Con     0           ' Define serial output pin
Bpin    Con     2           ' Define Button input pin
B0      Var     Byte

        B0 = 0              ' Zero Button working buffer

loop: BUTTON Bpin,1,10,5,B0,0,notp    ' Check button
                                      (skip if not
                                      pressed)
        Serout SO,N2400,["Press",13,10] ' Indicate button
                                      pressed
notp: Serout SO,N2400,[#B0,13,10]     ' Show working variable

        Pause 100           ' Wait a little
        Goto loop           ' Do it forever
```

5.7. CALL

CALL Label

Label 이름의 어셈블리 언어로 작성된 서브루틴을 실행합니다.

GOSUB 문이 PICBasic Pro 의 서브루틴에서 사용됩니다. CALL 과 GOSUB 의 차이점은 CALL 명령은 Label 존재 유뮤를 어셈블리 전까지는 확인하지 않습니다. CALL 명령의 Label 은 어셈블리 언어 부분에서 있는 것이며

PicBasic Pro 에서는 참조할 수 없습니다.
더 자세한 CALL 명령은 어셈블리 프로그램 부분을 참고하시기 바랍니다.

CALL pass ' 어셈블리 서브루틴 _pass 를 호출하여 실행합니다.

5.8. CLEAR
모든 RAM 레지스터의 내용을 0 으로 만듭니다.
CLEAR 명령은 모든 뱅크의 레지스터의 내용을 0 으로 만듭니다. 이것은 내부적으로 사용되는 변수와 프로그램에서 선언된 모든 변수에 적용됩니다. BASIC Stamp 에서는 프로그램이 시작될 때 모든 변수를 클리어 하지만 Pic Basic Pro 에서는 자동적으로 클리어 되지 않습니다.일반적으로 변수는 CLEAR 명령에 의하여 크리어 되는 것보다 각자의 고유한 초기값으로 설정합니다.
CLEAR 는 12-비트 코어에서 Bank0 레지스터를 Zero 으로 하지 않습니다.

```
CLEAR       ' Clear all variavles to 0
```

5.9. CLEARWDT

CLEARWDT

워치독 타이머를 클리어 합니다.

워치독 타이머는 SLEEP 과 NAP 명령과 함께 PIC 프로세서가 정해진 시간 이후에 깨어날 수 있도록 합니다. 어셈블리명령(clrwdt)은 정상적인 상황에서 워치독 타이머를 타이밍 아웃으로 부터 보호합니다.
CLEARWDT 는 추가적으로 clrwdt 명령을 프로그램이 넣습니다.

```
      CLEARWDT        ' Clear Watchdog Timer
```

자동으로 삽입된 clrwdt 명령을 DEFINE 에 의하여 제거할수 있습니다. 대부분의 경우에 clrwdt 명령은 타이밍 루틴의 간섭을 주지 않기위하여 nop 명령으로 대치됩니다.

```
DEFINE NO_CLRWDT 1    ' Don't insert CLRWDTs
```

5.10. COUNT
COUNT Pin,Period,Var
선택한 핀에서 일정시간동안 입력되는 펄수를 카운트하여 변수에 저장합니다. Pin 은 자동적으로 입력상태로 됩니다. Pin 은 0-15 의 상수이거나 0-15 가 내장된 변수 또는 PORTA.0 와 같은 핀 이름입니다.
Periode 는 밀리초(1/1000) 단위입니다. DEFINEd OSC 에서 정해진 오실레이터 주파수에 의하여 정해집니다. COUNT 문은 Pin 이 low 에서 high 로 상태 변하는 것을 카운트하며 매우 빠른 반복주기로 검사합니다. 4MHz 오실레이터의 프로세서에서 20uS 주기이며 20MHz 오실레이터 프로세서에서 4uS 주기로 검사루프가 실행됩니다. 이것을 기초로 4MHz 프로세서는 25KHz , 20MHz 프로세서는 125KHz 의 카운트가 가능합니다.

```
' Count # of pulse on Pin1 in 100 miliseconds
COUNT PORTB.1, 100, W1

' Determine frequence on a pin
COUNT PORTA.2 , 1000, W1    ' Count for 1 second
Serout PORTB.0, N2400, [W1]
```

5.11. DATA
{Label} DATA { @ Location, } Constant {, Constant...}
프로세서가 프로그램될 때 내장된 비휘발성 EEPROM 에 상수 값을 저장합니다. 옵션 항목인 Location 이 생략되면 DATA 문의 첫번째 숫자는 0 번지부터 차례로 저장됩니다. 그 다음 상수는 뒤이어 저장됩니다. Location 값이 지정되면 Location 이 나타내는 어드레스부터 저장됩니다. DATA 문 앞에 label(콜론을 붙이지 않음)을 부착하면 나중에 프로그램에서 EEPROM 시작 어드레스를 지정할 수 있습니다. Constant 는 숫자 값 또는 스트링 값입니다. WORD 수정자가 사용되지 않으면 단지 숫자 값의 하위바이트 가 저장됩니다. DATA 는 PIC12F675, PIC16F84 , PIC16C84 그리고 PIC16F87x 시리즈 와 같이 EEPROM 이 내장된 경우에 사용 가능합니다. I2C 에 의하여 EEPROM 이 동작하는 12CE67X 와 16CE62X 에서는 DATA 문을 사용할 수 없습니다. EEPROM 은 비휘발성이므로 파워가 공급되지 않아도 저장된 데이터가 지워지지 않습니다.
마이크로 컨트롤러가 프로그램될 때 DATA 문에 지정된 데이터가 기록됩니다. 컨트롤러가 동작 중에는 WRITE 문에 의하여 EEPROM 에 데이터를 기록합니다.

```
' Store 10, 20 and 30 starting at location 5
DATA @5 , 10, 20, 30
' Assign a label to a word at the next location
dlabel DATA word $1234 '
' Skip 4 locations and store 10 0s
DATA (4), 0(0)
```

5.12. DEBUG

DEBUG Item{, Item...}

한 개 또는 여러 개의 항목을 미리 지정된 핀, 8 비트 no 패리티, 1 스톱비트의 비동기 모드 통신속도로 전송합니다. 지정된 핀은 자동적으로 출력모드로 됩니다. # 기호가 Item 앞에 붙으면 디지트의 ASCII 값을 시리얼로 전송합니다. DEBUG 는 SEROUT2 와 같은 데이터 수정자를 지원합니다(12 비트 코어는 지원하지 않음). 자세한것은 SEROUT2 를 참조하기 바랍니다.

수정자	동작
{I} {S} BIN {1..16}	Send binary digits
{I} {S} DEC {1..5}	Send decimal digit
{I} {S} HEX {1..4}	Send hexadecimal digits
REP c\n	Send character c repeated n times
STR ArrayVar{\n}	Send string of n characters

DEBUG 는 내장된 여러 비동기 시리얼 통신 함수의 하나입니다. 이 명령은 약간의 시리얼 통신 코드를 만듭니다. 이 명령은 디버그정보(변수, 프로그램 위치 표시)를 간단한 방법으로 외부로 전송합니다. 지정된 핀에 지정된 통신속도로 언제든지 시리얼 출력이 가능합니다.

시리얼 핀과 통신속도는 DEFINE 문에 의하여 지정합니다.

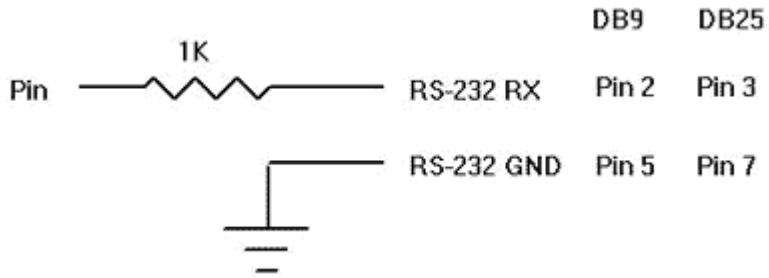
```
' Set Debug pin Port
DEFINE DEBUZG_REG PORTB

' Set DEBUG pin bit
DEFINE DEBUG_BIT 0

' Set Debug baud rate
DEFINE    DEBUG_BAUD 2400

' Set Debug mode: 0 = true, 1 = inverted
DEFINE DEBUG_MODE 1
```

DEBUG 는 오실레이터 클럭이 4 MHz 인 것으로 간주합니다. 다른 주파수를 사용하려면 DEFINE 으로 OSC 를 설정 하여야 합니다.

어떤 경우에 DEBUG 명령에 의한 시리얼 통신속도가 빠른 경우 전송되는 문자간에 시간 간격을 설정하는 것이 가능합니다. 문자간격시간은 DEFINE 으로 선언하며 1 부터 65536 마이크로 초 단위로 설정합니다. 예를 들어 전송되는 문자간의 간격을 1 밀리초로 하려면

DEFINE DEBUG_PACING 1000

RS-232 레벌 컨버터가 일반적으로 사용되지만 PICmicro 의 입출력 특성이 우수하므로 모든 경우에 반드시 레벨 컨버터를 필요로 하지 않습니다. Pic micro 와 RS-232 를 직접 연결하여 간단하게 사용하는 것이 가능합니다. TTL 반전기를 사용할 수 있습니다. RS-232 레벨 컨버터나 74LS04 와 같은 반전기를 사용한 경우 DEBUGIN_MODE = 1 로 하여야 합니다. 출력이 쇼트될 것을 생각하여 전류제한 저항을 부착합니다.




```
' Send the text "B0=" followed by the decimal value of B0 and a linefeed out serially
DEBUG "B0=", dec B0, 10
```

5.13. DEBUGIN

DEBUGIN { Timeout, Label, } [Item{, Item...}]

한 개 또는 여러 개의 항목을 미리 지정된 핀, 8 비트 no 패리티, 1 스톱비트의 비동기 모드 통신속도로 수신합니다. 지정된 핀은 자동적으로 입력모드로 됩니다. Timeout 과 Label 은 지정한 시간이내에 문자가 수신되지 않을 경우 점프하는 위치입니다. Timeout 의 시간 단위는 밀리 초 단위입니다. 지정한

Timeout 시간이내에 입력이 오지 않으면 프로그램은 DEBUGIN 커맨드를 빠져 나오며 Label 로 점프합니다.

DEBUGIN 은 SERIN2 와 동일한 데이터 수정자를 지원합니다. 자세한 것은 SERI2 을 참조하기 바랍니다.

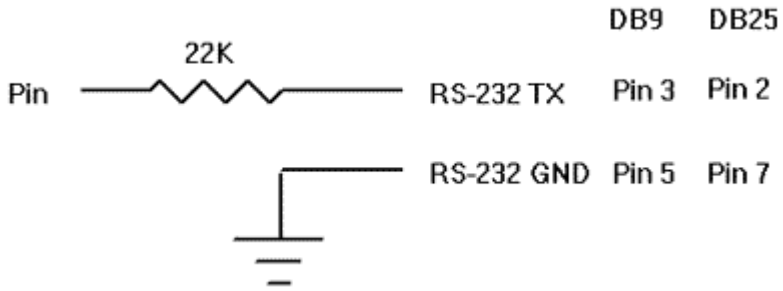
Modifier/수정자	Operation / 동작
BIN{1..16}	Receive binary digits
DEC{1..5}	Receive decimal digit
HEX{1..4}	Receive upper case hexadecimal digits
SKIP n	Skip n received characters
STR ArrayVar\n{\c}	Receive string of n characters optionally ended in character c
WAIT ( )	Wait for sequence of characters
WAITSTR ArrayVar{\n}	Wait for character string

DEBUGIN 은 내장된 여러 개의 비동기 통신 기능중의 하나입니다. 매우 작은 실행코드가 추가되며 외부로부터 전송되는 시리얼 데이터를 수신합니다. 지정된 핀을 어떤 임의의 시간에서도 정해진 통신속도로 데이터를 수신하는 용도로 사용할 수 있습니다.

시리얼 핀과 통신속도는 DEFINES: 으로 정의합니다.

```
' Set Debugin pin Port
DEFINE DEBUGIN_REG      PORTB
' Set Debugin Pin bit
DEFINE DEBUGIN_BIT      0
' Set Debugin mode: 0 = true, 1 = inverted
DEFINE DEBUGIN_MODE   1
```

만약 DEFINE 으로 초기 설정되지 않은 경우 DEFINEIN 명령은 DEBUG 에서 설정된 것을 사용합니다. DEBUGIN 에서 사용되는 통신속도는 DEBUG 와 항상 동일하게 사용되며 다르게 설정할 수 없습니다. DEBUGIN 은 4MHz 의 오실레이터 클럭을 기준으로 하여 통신속도가 정하여 집니다. 다른 오실레이터 주파수인경우 DEFINE 문에서 OSC 를 설정 하여야 합니다. 일반적으로 RS-232 레벨 변환 칩을 사용합니다만 RS-232 의 특성과 Picmicro 의 특성을 보면 대부분의 경우에 레벌변환기가 필요치 않습니다. 그리고 TTL 반전기를 사용할 수 있습니다. RS-232 레벨 컨버터나 74LS04 와 같은 반전기를 사용한 경우 DEBUGIN_MODE = 1 로 하여야 합니다. 반드시 전류제한 저항을 추가하여야합니다.




```
' Wait   until the character "A" is received serially and put next character into B0
DEBUGIN [wait ("A"), B0]
' Skip 2 chars and grap a 4 digit decimal number
DEBUGIN [skip 2, dec4 B0]
```

5.14. DISABLE

DISABLE

DISABLE 은 이 명령 이후에 디버거와 인터럽트 처리를 하지 않도록 합니다. 인터럽트가 발생하더라도 PICBasic Pro 프로그램의 BASIC 인터럽트 핸들러는 ENABLE 을 만날 때까지 동작하지 않습니다. DISABLE 과 ENABLE 명령은 컴파일러에게 지시하는 의사(pseudo) 명령이므로 실행코드를 만들지는 않습니다. 자세한 것은 ON DEBUG 명령과 ON INTERRUPT 명령을 참고하시기 바랍니다.

```
     DISABLE        ' Disable interrupts in handler
myint: led = 1      ' Turn on LED when interrupted
       Resume       ' Return to main program
       Enable       ' Enable interrupts after handler
```

5.15 DISABLE DEBUG

DISABLE DEBUG

이 명령 이후에는 디버그 작업을 중지합니다. ENABLE 명령이 나타나기 전까지는 디버그 모니터는 동작하지 않습니다.

DISABLE DEBG 와 ENABLE DEBUG 는 의사(pseudo) 명령이므로 실행코드를 만들지 않습니다. 더 자세한 것은 ON DEBUG 를 참조하십시오.

```
DISABLE DEBUG     ' Disable debug monitor calls
```

5.16. DISABLE INTERRUPT

DISABLE INTERRUPT

이 명령 이후에는 인터럽트 처리를 하지 않습니다. 인터럽트는 발생 하지만 Pic BASIC Pro 프로그램은 ENABLE 또는 ENABLE INTERRUT 를 만나기 전까지 인터럽트를 실행하지 않습니다. DISABLE INTERRUPT 과 ENABLE INTERRUPT 명령은 컴파일러에게 지시하는 의사(pseudo) 명령이므로 실행코드를 만들지는 않습니다. 자세한 것은 ON INTERRUPT 명령을 참고하시기 바랍니다.

```
        DISABLE INTERRUPT         ' Disable interrupts in handler
Myint: led = 1        ' Turn on LED when interrupted
        Resume        ' Return to main program
        Enable Interrupt      ' Enable interrupts after handler
```

5.17. DTMFOUT

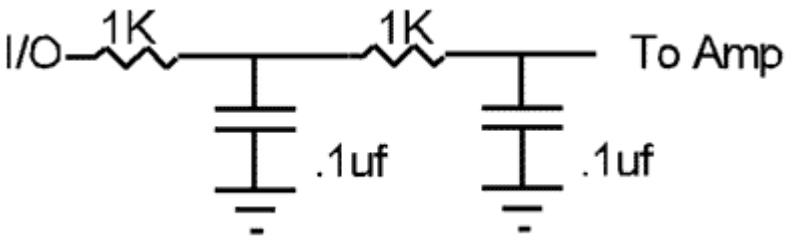
DTMFOUT Pin, {Onms, Offms, } [Tone{, Tone...}]

지정한 핀에 DTMF 음을 발생합니다. 핀은 자동으로 출력상태로 설정됩니다. Pin 은 0 - 15 의 상수 또는 이를 저장한 변수 또는 핀 이름 (예를 들어 PORTA.0)가 될 것입니다.

Onms 는 음이 방생되는 밀리세컨드 단위의 시간이며 Offms 는 밀리세컨드 단위의 음간의 무음 시간입니다. 이 것이 지정되지 않으면 Onms 는 200mS 그리고 Offms 는 50mS 로 기본설정됩니다.

Tones 는 0-15 의 번호입니다. 음 0-9 는 전화 키보드와 같습니다. 음 10 은 * 키이며 음 11 은 #키가 됩니다. 음 12-15 는 확장키 A-D 에 해당됩니다.

DTMF 는 FREQOUT 을 이용하여 두개의 음을 만듭니다.



FREQOUT 은 펄스 폭 변조 형태를 이용하여 음을 발생합니다. 핀은 구형파가 출력됩니다. 파형을 부드럽게 하기 위하여 필터가 필요합니다. DTMF 는 20MHz 나 40MHz 의 오실레이터에서 잘 동작합니다. 10MHz 와 4MHz 도 사용 가능하지만 필터회로를 만들기 힘들며 출력 진폭도 작아집니다. 다른 주파수는 DTMFOUT 을 사용할 수 없습니다. DTMF 명령은 12 비트 코어에서 스택과 RAM 의 제한으로 사용할 수 없습니다.

```
 ' Send DTMF tones for 212 on Pin1
DTMF PORTB.1, [2, 1, 2]
```

5.18. EEPROM

EEPROM {Location, } [Constant{, Constant ... }]

상수값을 내장된 EEPROM 에 저장합니다. Location 값을 생략하면 EEPROM 의 0 번지부터 연속적으로 저장합니다. Location 값을 지정하면 값이 저장될 시작위치를 지정하는 것입니다.

Constant 는 숫자 상수이거나 스트링 상수입니다. 단지 최하위 바이트만(Least Significant Byte)이 저장됩니다. 스트링은 ASCII 값의 연속으로 저장됩니다. 길이를 나타내는 정보나 스트링의 끝 위치를 알리는 정보가 포함되지 않습니다.

EEPROM 은 PIC16F675, PIC16F84, PIC16C84 와 같은 프로세서에 내장된 EEPROM 에 동작합니다. 프로세서에 내장된 EEPROM 이라 하더라도 I2C 방식을 사용하는 12CE67X 또는 16CE62X 는 사용할 수 없습니다. EEPROM 은 불휘발성 메모리 이므로 전원이 공급되지 않아도 데이터는 남아 있습니다. 데이터는 마이크로 프로세서가 프로그램될 때 EEPROM 에 기록됩니다. WRITE 명령은 프로그램 실행 시 EEPROM 에 데이터를 저장합니다. READ 명령은 저장된 DATA 값을 읽습니다.

```
        ' Store 10, 20 and 30 starting at location 4
        EEPROM 4, [10, 20, 30]
```

5.19. ENABLE

DISABLEs 명령에 의하여 금지된 디버그와 인터럽트를 이 명령 이후부터 동작 가능하게 합니다. DISABLE 과 ENABLE 명령은 컴파일러에게 지시하는 의사(pseudo) 명령이므로 실행코드를 만들지는 않습니다. 자세한 것은 ON DEBUG 와 ON INTERRUPT 명령을 참고하시기 바랍니다.

```
      Disable     ' Disable interrupts in handler
```

```
Myint: led = 1      ' Turn on LED when interrupted
       Resume       ' Return to main program
       ENABLE       ' Enable interrupts after handler
```

5.20. ENABLE DEBUG
ENABLE DEBUG

DISABELd 명령에 의하여 금지된 디버그 동작을 가능하게 합니다.
DISABLE DEBUG 와 ENABLE DEBUG 명령은 컴파일러에게 지시하는 의사(pseudo) 명령이므로 실행코드를 만들지는 않습니다. 자세한 것은 ON DEBUG 명령을 참고하시기 바랍니다.

5.21 ENABLE INTERRUPT
ENABLE INTERRUPT

DISABELd 명령에 의하여 금지된 디버그 동작을 가능하게 합니다.
DISABLE INTERRUPT 와 ENABLE INTERRPUT 명령은 컴파일러에게 지시하는 의사(pseudo) 명령이므로 실행코드를 만들지는 않습니다. 자세한 것은 ON INTERRUPT 명령을 참고하시기 바랍니다.

```
       Disable Interrupt     ' Disable interrupts in handler
Myint: led     = 1           ' Turn on LED when interrupted
       Resume                ' Return to main program
       ENABLE INTERRUPT      ' Enable interrupts after handler
```

5.22. END
END
프로그램의 실행을 멈추고 저전력 모드로 들어갑니다. 모든 I/O 핀은 현재의 상태를 유지합니다. END 명령은 sleep 인스트럭션을 계속해서 실행합니다.
END , STOP 또는 GOTO 는 모든 프로그램의 마지막에 와야 합니다.

```
END
```

5.23. ERASECODE
ERASECODE Block
PIC18Fxxx 와 같은 PIC 마이크로 MCU 는 WRITECODE 에 의하여 다시 쓰기 하기 전에 코드 영역을 지워야 합니다. 이런 소자는 블록을 한번에 지웁니다. 이레이즈 블록은 32Word(64 Bytes) 또는 소자에 따라 다른 크기 입니다. 이 크기는 일반적으로 라이트 블록 단위보다 큰 것입니다. 특정한 PIC 마이크로 MCU 데이터를 확인하여야 합니다. Block 을 지정하여 지우려는 첫번째 블록 위치를 지정할 수 있습니다. Block 은 바이트 단위의 어드레스 입니다. Block 이 코드 어드레스를 지정하지 않도록 주의하여야 합니다.
Flash 프로그램은 라이트는 ERASECODE 에 의하여 이레이즈 할 수 있도록 PIC 마이크로 MCU 디바이스 프로그램시 가능한 조건으로 Flash 프로그램을 라이팅 하여야 합니다.
이 명령을 사용할 때 사용할 수 없는 블록을 이레이즈 하도록 하여도 컴파일 에러가 발생하지 않습니다.

```
    ERASECODE     $100     ' Erase code block starting at location     $100
```

5.24. FOR.. NEXT

```
FOR Count = Start To End { STEP { - } Inc }
        { Body }
NEXT { Count }
```

FOR.. NEXT 루프는 변수로 지정하는 카운터값 만큼 여러개의 명령을 실행합니다.
1) Start 값은 인덱스 변수로 지정됩니다. Count 는 모든 변수 형식을 지원합니다.
2) Body 가 실행됩니다. Body 가 생략될수 있습니다. (아마도 딜레이 루프가 될 것입니다.)
3) Inc 값이 Count 에 더해(또는 Step 의 부호가 - 일때는 감산)집니다. 만약 STEP 을 생략하였다면 Count 는 하나씩 증가합니다.
4) Count 가 END 를 통과하지 못했거나 변수타입을 오버 하지 않았다면 Step 2 로 갑니다.

만약 루프 카운트 수가 255 를 넘는다면 워드형 변수를 사용하여야 합니다.

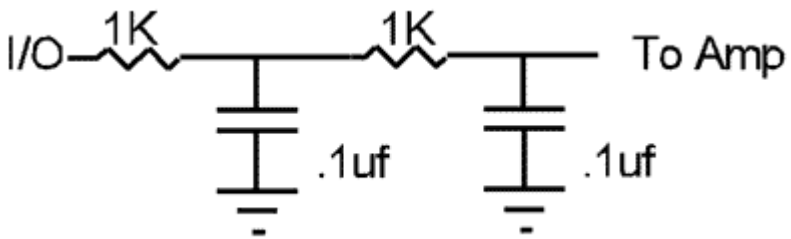
```
For i = 1 to 10       ' Count 1 to 10
Serout   0, N2400, [#i, " "]     ' Send each number to Pin0 serially
NEXT i                           ' Go back to and do next count
Serout   0,N2400,[10]     ' Send a linefeed
```

```
For i = B2 to 10    STEP -2        ' Count 20 to 10 by 2
Serout    0, N2400, [#B2, " "]     ' Send each number to Pin0 serially
NEXT B2                            ' Go back to and do next count
Serout    0,N2400,[10]     ' Send a linefeed
```

5.25. FREQOUT

FREQOUT Pin, Onms, Frequency1 {, Frequency2 }

지정한 Onms 밀리초 동안 지정한 핀에 주파수를 출력합니다. Pin 은 0-15 의 상수, 0-15 를 저장한 변수 또는 PORTA.0 과 같은 핀 이름이 될 것입니다.
하나 또는 2 개의 0 부터 32767 Hz 의 주파수가 동시에 출력됩니다. FREQOUT 은 펄스폭 변조로 음을 발생합니다. 신호를 사인파와 같이 부드럽게 하고 고조파를 제거하기 위하여 필터 회로가 필요합니다. FREQOUT 는 20 MHz 또는 40MHz 에서 잘동작합니다. 10MHz 와 4MHz 에서도 잘 동작하지만 필터회로를 만들기 어려우며 진폭이 작아집니다. 다른 오실레이터 주파수는 실제로 사용한 오실레이터와 20MHz 의 비율로 주파수가 출력됩니다. FREQ 는 RAM 과 Stack 의 제한 때문에 12 비트 코어에서 사용할 수 없습니다.




```
' Send 1KHz tone on Pin1 for 2 seconds
FREQOUT PORTB.1, 2000, 1000
' Send 350Hz / 440Hz (Dial Tone) for 2 seconds
FREQOUT PORTB.1, 2000, 350, 440
```

5.26. GOSUB

GOSUB Label1

Label 위치로 점프합니다. 이때 복귀할 어드레스를 스택에 저장합니다. GOTO 와 달리 RETURN 문을 만나면 실행을 멈추고 GOSUB 가 있었던 다음 라인을 실행합니다. 프로그램에서 GOSUB 는 사용 제한이 없습니다. 서브루틴은 내포할 수 있습니다. 서브루틴문 안에서 다른 서브루틴을 호출할 수 있습니다. 서브루틴은 4 개이하의 레벨로 제한됩니다. (17Cxxx 에서 12 레벨, 18Cxxx 에서 27 레벨)

```
GOSUB beep ' Execute subroutine named beep…
Beep: High 0              ' Turn on LED connected to Pin0
      Sound 1, [80, 10]       ' Beep speaker connected to Pin0
      Return                  ' Go back to main routine that called us
```

5.27. GOTO

GOTO Label

프로그램의 실행제어를 Label 로 옮깁니다.

```
    GOTO Send     ' Jump to statement labeled send
Send: Serout 0, N2400, ["Hi"]     ' Send "Hi" out Pin0 serially
```

5.28. HIGH

HIGH Pin

지정한 핀을 Higf 상태로 설정합니다. Pin 은 자동으로 출력상태로 됩니다. 핀은 0-15 의 상수 , 0-15 를 저장한 변수 또는 PORTA.0 과 같은 핀 번호가 됩니다.

```
    HIGH 0        ' Make Pin0 an output and set it high (~5 volts)
    HIGH PORTA.0    ' Make PORTA, Pin 0 an output and set it high ( ~5 volts)

Led         var          PORTB.0      ' Define LED Pin
          HIGH led                     ' Make LED Pin an output and set it high (~ 5 volts)
```

이전에 핀이 출력상태로 되어 있다면 빠르고(출력코드 수 관점에서) 간단한 방법으로 핀을 High 상태로 할 수 있습니다.

```
      PORTB.0 = 1      ' Set PORTB pin 0 high
```

5.29. HPWM

HPWM Channel, Dutycycle, Frequency

PWM 하드웨어를 내장한 PIC 마이크로 MCU 에서 변조된 펄스를 출력합니다. 이 것은 다른 명령어에 의하여 프로그램이 실행중이더라도 연속적으로 동작합니다.
Channel 은 PWM 하드웨어 채널을 지정합니다. 1,2 또는 3 개의 PWM 채널을 가진 프로세서가 있습니다. Microchip 의 데이터를 보면 특정칩의 고정된 하드웨어 핀을 알수 있습니다. 예를들어 PIC16F877 은 Channel 1 은 CPP1 이며 이것은 PORTC.2 입니다. Channel 2 는 CPP2 이며 PORTC.1 입니다.

18C452 는 HPWM 을 겸용하도록 되어 있습니다. 다음 예는 이핀을 사용하는 예입니다.

```
DEFINE    CCP1_REG    PORTC       ' Hpwm 1 pin port
DEFINE    CCP1_BIT    2           ' Hpwm 1 pin bit
DEFINE    CPP2_REG    PORTC       ' Hpwm 2 pin port
DEFINE    CCP2_BIT    1           ' Hpwm 2 pin bit
```

Dutycycle 은 신호의 on/off 를 지정합니다. 범위는 0 – 255 입니다. 0 은 off(모든 시간동안) 이며 255 는 on(모든 시간동안) 입니다.
127 은 50% 의 듀티비가 됩니다.
Frequency 는 PWM 신호의 원하는 주파수가 됩니다. 2 채널 디바이스에서 Frequency 는 양 채널 모두 같게 됩니다. 오실레이터 설정에서 모든 주파수가 가능하지 않습니다. 최고 주파수는 32768Hz 가 됩니다. 최저 주파수는 사용한 오실레이터에따리 아래 표와 같습니다.

OSC	14-bit code and 18Xxxx	17Cxxx
4MHz	245Hz	3907Hz
8MHz	489Hz	7813Hz
10MHz	611Hz	9766Hz
12MHz	733Hz	11719Hz
16MHz	977Hz	15625Hz
20MHz	1221Hz	19531Hz
24MHz	1465Hz	23438Hz
25MHz	1527Hz	24415Hz
33MHz	2015Hz	32227Hz
40MHz	2442Hz	na

다음의 DEFINE 문은 PIC17C7xx 디바이스체서 PWM 채널 2 와 PWM 채널 3 를 타이머 1 또는 2 로 지정하는 것입니다. DEFINE 이 정의되지 않았다면 timer1 이 사용됩니다.

```
DEFINE HPWM2_TIMER 1 ' Hpwm 2 timer select
DEFINE HPWM3_TIMER 1     ' Hpwm 3 timer select
```

HPWM 커맨드 이후에 CCP 컨트롤 레지스터는 PWM 모드가 됩니다. CCP 핀이 HPWM 커맨드 이후에 일반 I/O 로 사용되었다면 CCP 컨트롤 레지스터는 PWM 을 Off 로 설정하여야 됩니다.
특정 칩에 대하여는 Microchip 의 데이터를 확인하십시오.

```
  HPWM 1, 127, 1000     ' Send a 50% duty cycle PWM signal at 1 Khz
  HPWM 1, 64, 2000      ' Send a 25% duty cycle PWM signal    at 2 KHz
```

5.30. HSERIN

HSERIN {ParityLabel, } { Timeout, Label, } [Item{,…}]
한 개 또는 그 이상의 항목을 하드웨어 시리얼 통신을 지원하는 디바이스에서 수신합니다.
HSERIN 은 비동기 시리얼 통신의 여러 기능 중 하나입니다. 이 명령은 하드웨어 USART 를 가진 디바이스에서만 사용할 수 있습니다. 시리얼 입력 핀과 다른 파라메터는 디바이스 데이터 시트를 참고하여야 합니다. 시리얼 파라메터와 바우드 레이트는 DEFINE 을 사용하여 지정합니다.

```
' Set receive register to receiver enBLED
DEFINE HSER_RCSTA 90h
' Set transmit register to transmitter enabled
DEFINE HSER_TXSTA 20h
' Set baud rate
DEFINE HSER_BAUD 2400
' Set SPBRG directly (normally set by HSER_BAUD)
DEFINE HSER_SPBRG 25
```

HSER_RCSTA, HSER_TXSTA 그리고 HSER_SPBRG 는 단순히 PIC 마이크로 MCU 의 관계된 레지스터 RCSTA, TXSTA 그리고 SPBRG 입니다. HEX 값으로 프로그램이 시작하기 전에 한번 정의합니다. 자세한 것은 Microchip 데이터를 참고합니다.
TXSTA 레지스터 BRGH 비트(2 번 비트)는 바우드 레지스터 발생기의 고속 모드를 제어합니다. 특정 바우드레이트는 특정 오실레이터 속도를 필요로 하

며 이 비트는 정상 동작하기 위하여 필요합니다. 이렇게 하기 위하여 HSER_TXSTA 를 20h 대신에 24H 로 설정합니다. 하드웨어 시리얼 포트 바우드 레이트 테이블과 추가적인 정보는 Microchip 데이터를 참고합니다.
HSERIN 은 바우드레이트를 계산할 때 4MHz 를 기준으로 합니다. 다른 오실레이터 주파수에서도 바우드레이트가 유지되기 원하면 DEFINE 으로 새로운 오실레이터 값으로 OSC 를 설정합니다. 선택 사양인 Timeout 과 Label 은 정해진 시간동안 문자가 수신되지 않을 때 프로그램을 계속실행하기 위하여 포함됩니다. Timeout 은 1 밀리 초 단위로 지정합니다.
만약 Timeout 시간 이내에 아무런 문자가 수신되지 않으면 HSERIN 문을 빠져 나가며 Label 로 점프합니다. 시리얼 데이터 입력 포맷은 내정치 로 8 N 1 으로 데이터비트 8 비트, 패리티 없음, 1 스톱비트 입니다. DEFINES 문에 의하여 7E1 (7 Data, Even Parity, 1 Stop) 또는 7O1(7 Data, Odd Parity, 1 Stop)을 선택할 수 있습니다.

```
' Use only if even parity desired
DEFINE HSER_EVEN 1

' Use only if odd parity desired
DEFINE HSER_ODD 1

' Use 8 bits + parity
DEFINE HSER_BITS 9
```

옵션사항으로 ParityLabel 은 수신한 문자가 에러일 때 프로그램이 이 위치로 점프합니다. 이것은 단지 DEFINES 문에 의하여 Parity 가 사용상태로 선언된 이후에 사용 가능합니다.
하드웨어 시리얼 포트는 2 바이트의 입력 버퍼를 가지고 있습니다. 문자를 읽지 않으면 쉽게 오버 플로우가 됩니다. 이러한 상태가되면 USART 는 새로운 문자를 수신하지 못하며 초기화 하여야 합니다. 어버 플로우 에러는 RCSTA 레지스터에 CREN 비트를 토글하여 리셋 가능합니다. DEFINE 은 이 에러를 자동으로 클리어 할 수 있습니다. 그러나 에러가 발생한 것을 알 수 없으며 문자는 손실됩니다.

```
    DEFINE     HSER_CLRERROR
```

수작업으로 에러를 클리어 할 수 있습니다.
RCSTA.4 = 0
RCSTA.4 = 1
시리얼 수신은 하드웨어로 실행하므로 RS-232 드라이버를 생략하기 위하여 반전된 상태로 설정할 수 없습니다. 그러므로 HSERIN 문을 사용할 때 적절한 드라이버를 사용하여야 합니다.
2 개의 하드웨어 시리얼 포트가 있을 때 HSERIN 은 단지 첫번째로 사용됩니다. 2 번째 하드웨어 시리얼 포트는 HSERIN2 로 하여야 합니다. 또는 DEFINE 문에 의하여 HSERIN 이 첫번째 대신 두번 째 하드웨어 시리얼 포트로 동작할 것을 설정할 수 있습니다.
DEFINE HSER_PORT 2

HSERIN 은 SERIN2 와 동일한 데이터 수정명령을 사용합니다. 이와 관련된 정보는 SERIN2 를 참고합니다.

Modifier	Operation
BIN{1..16}	Receive binary digits
DEC{1..5}	Receive decimal digits
HEX{1..4}	Receive upper case hexadecimal digits
SKIP n	Skip n received characters
STR ArrayVar\n{\c}	Receive string on n characters optionally ended in character c
WAIT ( )	Wait for sequence of characters
WAITSTR Array{\n}	Wait for character string

HSERIN [B0, DEC W1]

5.31. HSERIN2

HSER2 {ParityLabel, } { Timeout, Label, } [Item{,…}]
하드웨어 시리얼 포트기 있는 프로세서의 2 번째 시리얼 포트로부터 하나 또는 그 이상의 데이터를 읽습니다.
HSERIN2 는 2 개의 UART 가 있는 PIC18F8720 의 ㄷ 번째 하드웨어 시리얼 포트를 사용한다는 것 이외에는 HSERIN 과 동일합니다.
이 명령은 2 개의 하드웨어 UART 가 있는 프로세서에서 동작합니다.
시리얼 파라메터와 통신속도는 DEFINES 를 이용하여 설정합니다.

```
' Set receive register to receiver enabled
DEFINE HSER2_RCSTA      90h
```

```
' Set transmit register to transmitter enabled
DEFINE HSER2_TXSTA        20h

' Set SPBRG (normally set by HSER2_BAUD)
DEFINE HSER2_SPBRG        25

' Use only if even parity desired
DEFINE HSER2_EVEN    1

' Use only if odd parity desired
DEFINE HSER2_ODD      1

' Use 8 bits + parity
DEFINE HSER2_BITS 9

' Automatically clear overflow errors
DEFINE HSER2_CLRERR          1

HSERIN2    [B0, DEC W1]
```

5.32. HSEROUT

HSEROUT [Item{, Item...}]

한 개 또는 그 이상의 항목을 하드웨어 시리얼 통신을 지원하는 디바이스에서 송신합니다.
HSEROUT 은 비동기 시리얼 통신의 여러 기능 중 하나입니다. 이 명령은 하드웨어 USART 를 가진 디바이스에서만 사용할 수 있습니다. 시리얼 출력 핀과 다른 파라메터는 디바이스 데이터 시트를 참고하여야 합니다. 시리얼 파라메터와 바우드 레이트는 DEFINE 을 사용하여 지정합니다.

```
' Set receive register to receiver enabled
DEFINE HSER_RCSTA 90h
' Set transmit register to transmitter enabled
DEFINE HSER_TXSTA 20h
' Set baud rate
DEFINE HSER_BAUD 2400
' Set SPBRG directly (normally set by HSER_BAUD)
DEFINE HSER_SPBRG 25
```

HSER_RCSTA, HSER_TXSTA 그리고 HSER_SPBRG 는 단순히 PIC 마이크로 MCU 의 관계된 레지스터 RCSTA, TXSTA 그리고 SPBRG 입니다. HEX 값으로 프로그램이 시작하기 전에 한번 정의합니다. 자세한 것은 Microchip 데이터를 참고합니다.
TXSTA 레지스터 BRGH 비트(2 번 비트)는 바우드 레지스터 발생기의 고속 모드를 제어합니다. 특정 바우드 레이트는 특정 오실레이터 속도를 필요로 하며 이 비트는 정상 동작하기 위하여 필요합니다. 이렇게 하기 위하여 HSER_TXSTA 를 20h 대신에 24H 로 설정합니다. 하드웨어 시리얼 포트 바우드 레이트 테이블과 추가적인 정보는 Microchip 데이터를 참고합니다.
HSERIN 은 바우드 레이트를 계산할 때 4MHz 를 기준으로 합니다. 다른 오실레이터 주파수에서도 바우드 레이트가 유지되기 원하면 DEFINE 으로 새로운 오실레이터 값으로 OSC 를 설정합니다.
. 시리얼 데이터 포맷은 내정 치로 8 N 1 으로 데이터비트 8 비트, 패리티 없음, 1 스톱비트 입니다. DEFINES 문에 의하여 7E1 (7 Data, Even Parity, 1 Stop) 또는 7O1(7 Data, Odd Parity, 1 Stop)을 선택할 수 있습니다.

```
' Use only if even parity desired
DEFINE HSER_EVEN 1

' Use only if odd parity desired
DEFINE HSER_ODD 1

' Use 8 bits + parity
DEFINE HSER_BITS 9
```

시리얼 수신은 하드웨어로 실행하므로 RS-232 드라이버를 생략하기 위하여 반전된 상태로 설정할 수 없습니다. 그러므로 HSERIN 문을 사용할 때 적절한 드라이버를 사용하여야 합니다.
2 개의 하드웨어 시리얼 포트가 있을 때 HSERIN 은 단지 첫번째로 사용됩니다. 2 번째 하드웨어 시리얼 포트는 HSERIN2 로 하여야 합니다. 또는 DEFINE 문에 의하여 HSERIN 이 첫번째 대신 두번 째 하드웨어 시리얼 포

트로 동작할 것을 설정할 수 있습니다.

```
DEFINE HSER_PORT 2
```

HSERIN 은 SERIN2 와 동일한 데이터 수정명령을 사용합니다. 이와 관련된 정보는 SERIN2 를 참고합니다.

Modifier	Operation
BIN{1..16}	Send binary digits
DEC{1..5}	Send decimal digits
HEX{1..4}	Send hexadecimal digits
REP c\n	Send character c repeated n times
STR ArrayVar\n{\c}	Send string of n characters

```
' Send the decimal value of B0 followed by a linefeed out the hardware USART
HSEROUT [DEC B0, 10]
```

5.33. HSEROUT2

HSEROUT2 [Item{, Item...}]

하드웨어 시리얼 포트가 있는 프로세서의 2 번째 시리얼 포트로부터 하나 또는 그 이상의 데이터를 송신합니다.

HSEROUT2 는 2 개의 UART 가 있는 PIC18F8720 의 ㄷ 번째 하드웨어 시리얼 포트를 사용한다는 것 이외에는 HSEROUT 과 동일합니다.

이 명령은 2 개의 하드웨어 UART 가 있는 프로세서에서 동작합니다.

시리얼 파라메터와 통신속도는 DEFINES 를 이용하여 설정합니다.

```
' Set receive register to receiver enabled
DEFINE      HSER2_RCSTA     90h

' Set transmit register to transmitter enabled
DEFINE      HSER2_TXSTA     20h

' Set baud rate
DEFINE      HSER2_BAUD 2400

' Set SPBRG2 (normally set by HSER2_BAUD)
DEFINE      HSER2_SPBRG     25

' Use only if even parity desired
DEFINE      HSER2_EVEN    1

' Use only if odd parity desired
DEFINE HSER2_ODD     1

' Use 8 bits + parity
DEFINE HSER2_BITS 9

' Send the decimal value of B0 followed by a linefeed out the hardware USART
HSERIN2    [B0, DEC W1]
```

5.34. I2CREAD

I2CREAD DataPin, ClockPin, Control, {Address, } [Var{, Var...}]{, Label}

제어 신호와 옵션 어드레스 바이트를 ClockPin 과 DataPin 으로 송신하고 수신된 데이터를 변수 Var 에 저장합니다. ClockPin 과 DataPin 은 0 - 15 의 상수 또는 0 - 15 를 저장하고 있는 변수 또는 PORTA.0 과 같은 핀 이름이 됩니다. I2CREAD 와 I2CWRITE 는 Microchip 사의 24C01B 와 이와 유사한 2-Wire I2C 인터페이스를 지원하는 EEPROM 으로 데이터를 Write 하거나 Read 합니다. 비 휘발성 메모리 이므로 파워가 Off 하여도 데이터는 남아있습니다.

이 명령은 I2C 마스터 모드로 동작하므로 I2C 인터페이스를 사용하는 온도센서 AD 컨버터와 함께 사용합니다.

12 비트 PIC 마이크로 MCU 에서는 I2C 클록 핀과 데이터핀을 DEFINE 을 이용하여 컴파일시 확정하여야 합니다.

```
    DEFINE     I2C_SCL     PORTA, 1       ' For 12-bit core only
    DEFINE     I2C_SDA    PORTA.0         ' For 12-bit core only
```

콘트롤 바이트의 상위 7 비트는 칩 셀렉트, 어드레스정보 등 디바이스에 따른 제어 코드이며 하위 1 비트는 Read / Write 방향을 표시합니다.

이 Control 바이트 포맷은 초기의 PicBASIC 컴파이러와 다릅니다. PBP I2C 동작과 함께 이 포맷을 이용하여야 합니다.

예를들어 24LC01B 를 사용할때 제어 코드는 %1010 이 됩니다. 칩 셀렉트를 사용하지 않으므로 %10100000 또는 $A0 가 됩니다. Control 바이트의 포맷은 아래 표와 같이 디바이스에따라 다릅니다.

Device	Capacity	Control	Address size

24LC01B	128 bytes	%1010xxx0	1 byte
24LC02B	256 bytes	%1010xxx0	1 byte
24LC04B	512 bytes	%1010xxb0	1 byte
24LC08B	1K bytes	%1010xbb0	1 byte
24LC16B	2K bytes	%1010bbb0	1 byte
24LC32B	4K bytes	%1010ddd0	2 bytes
24LC65	8K bytes	%1010ddd0	2 bytes

bbb = block select (high oeder address) bits
ddd = device select bits
xxx = don't care

Address 크기는 사용할 변수형(Byte 또는 Word) 에 따라 결정됩니다. Byte 사이즈 변수를 사용하면 8-비트 어드레스를 송신하게 되며 Word 사이즈 변수를 사용하게되면 16-비트 어드레스를 송신하게 됩니다. 통신하기에 적합한 크기의 어드레스를 결정하여야 합니다. 상수를 어드레스로 지정하면 안됩니다. 상수값의 크기에따라 Byte 또는 Word 로 변화하기 때문입니다.
Word 크기의 어드레스를 지정하면 High 바이트가 송신되고 뒤이어 Low 바이트가 송신됩니다. 이것은 Low Byte 부터 먼저 저장되는 변수의 저장순서와 다른것입니다.
수정자 STR 은 변수 이름 앞에 포함할 수 있습니다. 이 것은 Word 나 Byte 어레이 이어야 하며 Backslash(\) 와 어레이 원소 개수를 표시하여야 합니다.

```
A     var     byte[8]
      I2CREAD     PORTC.4, PORTC.3, $a0,0,[STR a\8]
```

Word 사이즈 어레이를 지정하였다면 2 바이트씩 처리하며 Low 바이트를 먼저 읽습니다.
This is the opposite of how simple words are read and is consistent with the way the compiler normally stores word-sized variables.
옵션인 Label 이 포함되면 I2C 디바이스로부터 ACK 신호가 오지 않았을 때 점프할 위입니다.
I2C 명령은 12CExxx 와 16Cexxx 디바이스와 같이 칩에 내장된 EEPROM 에도 사용할수 있습니다. 프로그램에서 아래와 같이 정의합니다.

```
    DEFINE    I2C_INTERNAL    1
```

12CE67x 디바이스에서 데이터 라인은 GPIO.6 이며 클록라인은 GPIO.7 입니다. 16CE62x 에서 데이터 라인은 EEINTF.1 이며 클록라인은 EEINTF.2 입니다.

자세한 것은 Microchip 데이터를 참고하기 바랍니다.
I2C 명령의 타이밍은 표준 속도가 100KHz 이며 8MHz 오실레이터에서 가능합니다.
고속 모드는 20MHz 를 사용하면 400KHz 가 됩니다.
8MHz 이상의 오실레이터를 사용할때 표준 클록동작속도를 원하면 DEFINE 문을 사용합니다.

```
    DEFINE    I2C_SLOW    1
```

메모리와 스택의 제한때문에 12 비트 코어 PIC 마이크로 MCU 는 속도설정을 할수 없습니다. 저속(100KHz) I2C 디바이스는 반드시 4MHz 오실레이터를 사용하여야 하며 4MHz 이상은 고속(400KHz) 디바이스를 사용하여야 합니다.
I2C 데이터 버스에서 클록라인을 Low 로하여 수신소자의 동작을 정지하려면 아래와 같이 DEFINE 문을 사용합니다.

```
DEFINE    I2C_HOLD
```


I2C 클록과 데이터 라인은 양방향 오픈 콜렉터 방법을 사용하므로 반드시 4.7K 로 그림과 같이 풀업하여야 합니다.
클록라인은 오픈 콜렉터 대신 바이폴라 라인으로 구성하려면 프로그램에 DEFINE 문을 추가 합니다.

```
        DEFINE    I2C_SCLOUT    1

addr      var      byte
```

```
cont      con      %10100000
          addr   = 17                    ' set address to 17
          ' Read data at address 17 into B2
          I2CREAD PORTA.0, PORTA.1, cont, addr,[B2]
```

I2CREAD 와 I2CWRITE 에 관련된 디바이스는 Microchip 사의 "Non-Volatile Memory Products Data Book"을 참고합니다.

5.35. I2CWRITE

I2CWRITE DataPin, ClockPin, Control, {Address, } [Var{, Var...}]{, Label}

I2CWRITE 는 제어 신호와 옵션 어드레스 바이트를 ClockPin 과 DataPin 으로 송신하고 뒤이어 변수 Var 에 저장된 데이터를 송신합니다. ClockPin 과 DataPin 은 0 - 15 의 상수 또는 0 - 15 를 저장하고 있는 변수 또는 PORTA.0 과 같은 핀 이름이 됩니다. I2CREAD 와 I2CWRITE 는 Microchip 사의 24C01B 와 이와 유사한 2-Wire I2C 인터페이스를 지원하는 EEPROM 으로 데이터를 Write 하거나 Read 합니다. 비 휘발성 메모리 이므로 파워가 Off 하여도 데이터는 남아있습니다. 이 명령은 I2C 마스터 모드로 동작하므로 I2C 인터페이스를 사용하는 온도센서 AD 컨버터와 함께 사용합니다.
12 비트 PIC 마이크로 MCU 에서는 I2C 클록 핀과 데이터핀을 DEFINE 을 이용하여 컴파일시 확정하여야 합니다.

```
    DEFINE     I2C_SCL     PORTA, 1       ' For 12-bit core only
    DEFINE     I2C_SDA    PORTA.0         ' For 12-bit core only
```

Address 크기는 사용할 변수형(Byte 또는 Word) 에 따라 결정됩니다. Byte 사이즈 변수를 사용하면 8-비트 어드레스를 송신하게 되며 Word 사이즈 변수를 사용하게되면 16-비트 어드레스를 송신하게 됩니다. 통신하기에 적합한 크기의 어드레스를 결정하여야 합니다. 상수를 어드레스로 지정하면 안됩니다. 상수값의 크기에따라 Byte 또는 Word 로 변화하기 때문입니다.
Word 크기의 어드레스를 지정하면 High 바이트가 송신되고 뒤이어 Low 바이트가 송신됩니다. 이것은 Low Byte 부터 먼저 저장되는 변수의 저장순서와 다른것입니다.

시리얼 EEPROM 에 라이팅 할때 10 mS 대기하는것이 필요합니다. 라이팅이 완료되기 이전에 I2CREAD 나 I2CWRITE 를 시도하면 무시됩니다.

하나의 I2CWRITE 문을 이용하여 여러개의 바이트를 라이트 할수 있습니다. 어떤 EEPROM 은 대기시간 필요없이 여러개의 바이트를 라이트할수 있습니다.

수정자 STR 은 변수 이름 앞에 포함할 수 있습니다. 이 것은 Word 나 Byte 어레이 이어야 하며 Backslash(\) 와 어레이 원소 개수를 표시하여야 합니다.

```
A     var       byte[8]
      I2CREAD     PORTC.4, PORTC.3, $a0,0,[STR a\8]
```

Word 사이즈 어레이를 지정하였다면 2 바이트씩 처리하며 Low 바이트를 먼저 읽습니다.
This is the opposite of how simple words are read and is consistent with the way the compiler normally stores word-sized variables.
옵션인 Label 이 포함되면 I2C 디바이스로부터 ACK 신호가 오지 않았을 때 점프할 위입니다.
I2C 명령은 12CExxx 와 16Cexxx 디바이스와 같이 칩에 내장된 EEPROM 에도 사용할수 있습니다. 프로그램에서 아래와 같이 정의합니다.

```
    DEFINE     I2C_INTERNAL     1
```

12CE67x 디바이스에서 데이터 라인은 GPIO.6 이며 클록라인은 GPIO.7 입니다. 16CE62x 에서 데이터 라인은 EEINTF.1 이며 클록라인은 EEINTF.2 입니다. 자세한 것은 Microchip 데이터를 참고하기 바랍니다.
I2C 명령의 타이밍은 표준 속도가 100KHz 이며 8MHz 오실레이터에서 가능합니다.
고속 모드는 20MHz 를 사용하면 400KHz 가 됩니다.
8MHz 이상의 오실레이터를 사용할때 표준 클록동작속도를 원하면 DEFINE 문을 사용합니다.

```
    DEFINE    I2C_SLOW     1
```

메모리와 스택의 제한때문에 12 비트 코어 PIC 마이크로 MCU 는 속도설정을 할수 없습니다. 저속(100KHz) I2C 디바이스는 반드시 4MHz 오실레이터를 사용하여야 하며 4MHz 이상은 고속(400KHz) 디바이스를 사용하여야 합니다.
I2C 데이터 버스에서 클록라인을 Low 로하여 수신소자의 동작을 정지하려면 아래와 같이 DEFINE 문을 사용합니다.

```
DEFINE     I2C_HOLD
```

I2C 클록과 데이터 라인은 양방향 오픈 콜렉터 방법을 사용하므로 반드시 4.7K 로 그림과 같이 풀업하여야 합니다.
클록라인은 오픈 콜렉터 대신 바이폴라 라인으로 구성하려면 프로그램에 DEFINE 문을 추가 합니다.

```
          DEFINE    I2C_SCLOUT     1

addr       var       byte
cont       con       %10100000
          addr    = 17                        ' set address to 17
          ' Read data at address 17 into B2
          I2CREAD PORTA.0, PORTA.1, cont, addr,[B2]
```

I2CREAD 와 I2CWRITE 에 관련된 디바이스는 Microchip 사의 "Non-Volatile Memory Products Data Book"을 참고합니다.

5.36. IF.. THEN

```
If Comp { AND/OR Comp … } THEN Label
If Comp { AND / OR Comp … } THEN Statement …
IF Comp { AND/OR Comp … } THEN
      Statement …
ELSE
      Statement …
ENDIF
```

하나 또는 여럿을 비교합니다. 각 Comp 항은 변수와 상수 또는 다른 변수 그리고 앞서 나열한 오퍼레이터를 포함합니다.
IF .. THEN 평가는 TRUE 와 FALSE 항을 비교합니다. 조건 검사 결과가 TRUE 일 때 THEN 이후를 실행합니다. FALSE 일 때 THEN 이후를 실행하지 않습니다. 비 교 결과 가 0 일 때 FALSE 가 되며 0 이 아닌 모든 수에 대하여 TRUE 가 됩니다. 모든 비교는 무부호이므로 PBP 는 무부호형만 지원합니다. IF .. THEN 은 0 보다 작은 수에 대하여 검사하기 위하여 사용할 수 없습니다.

괄호는 검사 순서를 지정하므로 중요합니다.

IF THEN 은 2 가지 방법으로 사용합니다.
하나는 IF THEN 에서 THEN 은 근본적으로 GOTO 와 같습니다. 조건이 참이면 프로그램의 THEN 이후의 Label 로 점프합니다. 조건이 거짓이면 IF.. THEN 아래를 실행합니다.

```
IF Pin0 = 0 Then pushed      ' If button connected to Pin0 is pushed (0), Jump to Label pushed
IF B0 >= 40 Then old          ' If the value in variable B0 is greater than or equal to 40, jump to old
IF PORTB.0 THEN itson      ' If PORTB, Pin 0 os high (1), jump to itson
IF (B0 = 10) And (B1 = 20) THEN loop
```

두번째 형식은 IF .. THEN 은 THEN 문 뒤의 그룹을 조건부로 실행합니다. 실행문은 THEN 이후에 놓여지거나 또는 옵션인 ELSE 뒤에 놓려집니다. 그리고 반드시 ENDIF 를 사용하여 구조를 완성합니다.

```
IF B0 <> 10 THEN B0 = B0 + 1; B1 = B1 – 1
IF B0 <> 10 THEN
   B0 = B0 + 1
   B1 = B1 – 1
ENDIF

IF B0 = 20 THEN
    LED = 1
ELSE
    LED = 0
ENDIF
```

5.73. INPUT

INPUT Pin
지정한 Pin 을 입력 상태로 합니다. Pin 은 0-15 의 상수이거나 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름이 될 것입니다.
다른 방법으로 핀을 간단하고 빠르게 입력으로 설정하는 것은

```
TRISB.0 = 1      ' Set PORTB, Pin 0 to an input
```

TRIS 레지스터를 한번에 설정하는 것으로 포트의 모든 핀을 입력으로 설정할 수 있습니다.
TRIS = %11111111 ' Set all of PORTB to inputs.

5.38. LCDIN
LCDIN { Address, } [Var{,Var… }]
LCD 에 어드레스를 지정하여 데이터를 읽어 Var 에 저장합니다.
LCD 는 문자메모리 RAM 을 가지고 있습니다. 대부분의 LCD 는 여분의 RAM 을 가지고 있습니다. 이 RAM 은 LCDOUT 명령에 의하여 데이터를 저장할 수 있습니다.
CG(문자발생) RAM 은 $40 부터 $7F 의 어드레스 범위입니다. 디스프레이 데이터 RAM 은 $80 부터 시작합니다. 특정 LCD 의 어드레스와 기능은 데이터시트를 참고합니다. LCD 의 Read/Write 핀을 PIC 마이크로 MCU 에 연결하여 리드(LCDIN)하거나 라이트 (WRITE)하도록 합니다. 2 개의 DEFINEs 핀 어드레스를 제어합니다.

```
DEFINE    LCD_RWREG    PORTE    ' LCD read/write pin port
DEFINE   LCD_RWBIT       2          ' LCD read/write pin bit
```

LCD 와 PIC 마이크로 MCU 를 연결하기 위한 정보는 LCDOUT 을 참고합니다.

```
LCDIN [B0]
```

5.39. LCDOUT
LCDOUT Item{, Item … }
액정디스프레이장치에 항목을 표시합니다. PBP 는 HITACHI 44780 컨트롤러를 지원합니다. 이 LCD 는 14 또는 16 핀으로 구성되어 있습니다.
항목 앞에 파운드기호(#) 가 있으면 각 디지트에 ASCII 표시를 LCD 로 보냅니다. LCDOUT SER2 에서 사용한 수정자를 사용할 수 있습니다. (12 비트 코어 는 제외)

Modifier	Operation
BIN{1..16}	Send binary digits
DEC{1..5}	Send decimal digits
HEX{1..4}	Send hexadecimal digits
REP c\n	Send character c repeated n times
STR ArrayVar\n{\c}	Send string of n characters

프로그램은 첫번째 커맨드를 LCD 에 보내기 전에 0.5 초 기다립니다. LCD 는 LCDOUT 을 이용하여 커맨드나 문자를 보내기 전에 초기화 됩니다. 동작 중 어떤 이유로 파워 다운 하였다가 다시 파워 온 되면 내부 플래그가 리셋 되며 LCDOUT 이 다음에 사용될 때 초기화 됩니다.
FLAGS = 0
커맨드는 LCD 에 $FE 에 이어서 보냅니다. 유용한 커맨드는 다음의 표와 같습니다.

Command	Operation
$FE, 1	Clear Display
$FE, 2	Return home
$FE, $0C	Cursor off
$FE, $0E	Underline cursor on
$FE, $0F	Blinking cursor on
$FE, $10	Move cursor left one position
$FE, $14	Move cursor right one position
$FE, $80	Move cursor to beginning of second of first line
$FE, $C0	Move cursor to beginning of second line
$FE, $94	Move cursor to beginning of third line
$FE, $D4	Move cursor to beginning of fourth line

여러줄로 구성된 LCD 에서 여러줄의 처음 위치로 커서가 오는 명령이 있습니다. 대부분의 LCD 는 표시되는 문자와 라인은 연속적이지 않습니다. 대부분의 16 x 2 디스프레이는 첫번째 라인의 어드레스는 $80 입니다. 그리고 두번째 라인의 어드레스는 $C0 입니다.
LCDOUT $FE, $80+4
는 첫번째 라인의 네번째 위치입니다.

```
LCDOUT $FE,1, "Hello"      ' Clear display and show "Hello"
LCDOUT $FE, $C0, "World"       ' Jump to second line and show "World"
LCDOUT B0, #B1            ' Display B0 and decimal ASCII value of B1
```

PIC 마이크로 MCU 와 LCD 를 연결할 때 4-비트 모드나 8 비트모드로 연결할 수 있습니다 만약 8-비트 모드를 사용한다면 8 개의 데이터 전부 하나의

포트에 연결합니다. 만약 4 비트 버스 모드가 된다면 상위 4 비트만 포트의 상위 4 비트 또는 하위 4 비트에 연결합니다, Enable 핀과 Register Select 핀은 다른 모든 핀에 연결할 수 있습니다. PBP 는 DEFINE 에 의하여 특정 핀을 지정하지 않으면 LCD 는 4 비트 버스모드, 이며 PIC 마이크로 MCU 의 PORTA.0-PORTA.3 에 연결합니다. Register Select 는 PORTA.4 에 ENABLE 은 PORTB.3 에 연결합니다. 또한 2 라인 LCD 인 것으로 초기화 합니다.
이러한 설정을 변경하려면 여러 개의 DEFINE 을 사용합니다. 모두 대문자를 사용하며 PicBASIC Pro 프로그램의 시작위치에 오도록 합니다.

```
' set LCD Data Port
DEFINE LCD_DREG PORTB
' Set starting Data bit ( 0 or 4) if 4-bit bus
DEFINE LCD_DBIT       4
' Set LCD Register Select port
DEFINE LCD_RSREG      PORTB
' Set LCD Enable Port
DEFINE LCD_EREG    PORTB
' Set LCD Enable bit
DEFINE LCD EBIT      0
' Set Number of lines on LCD
DEFINE LCD_LINES    2
' Set command delay time in us
DEFINE LCD_COMMADUS       2000
' Set data delay time in us
DEFINE LCD_DATAUS      50
```

위의 설정은 PBP 에게 2 라인 LCD 를 4 비트 모드로 연결하도록 한 것입니다. PORTB 에 4 비트를 연결하고 RS 를 PORTB.1 에 그리고 E 를 PORTB.0 에 연결합니다.
아래 그림은 내정값 상태로 LCD 와 Pic 마이크로 MCU 를 연결한 기본 도면 입니다.

5.40. {LET}
{LET} Var = Value
값을 Variable 에 지정합니다. Value 는 상수, 다른 변수 또는 연산 결과입니다. 연산자에 관한 자세한 것을 확인하시기 바랍니다. 키워드 LET 은 옵션입니다.

```
LET    B0    B * B2 + B3
B0 = sqr W1
```

5.41. LOOKDOWN

LOOKDOWN Search, [Constant{, Constant...}], Var

LOOKDOWN 문은 8 비트의 상수를 Search 대상 리스트에서 찾습니다. 만약 발견된다면 일치한 상수의 인덱스 값이 Var 에 저장됩니다. 만약 발견된 값이 첫번째라면 Var 에는 0 이 저장됩니다. 리스트의 두번 째와 일치한다면 1 이, 세번 째와 일치한다면 2 가 Var 에 저장되며, 만약 일치하는 것이 발견되지 않는다면 Var 값은 변화되지 않습니다.

상수리스트는 숫자와 문자가 혼합될 수 있습니다. 스트링에서 각 문자는 문자 ASCII 값의 분리된 상수로 취급됩니다. 변수 인덱스가 있는 배열변수는 배열이 상수인덱스를 허용하더라도 사용할 수 없습니다.

```
Serin 1,N2400,B0      ' Get hexadecimal character from Pin1 serially
LOOKDOWN B0, ["0123456789ABCDEF"], B1    ' Convert hexadecimal character in B0 to decimal value B1
Serout 0, N2400,[#B1]    ' Send decimal value to Pin0 serially
```

5.42. LOOKDOWN2

LOOKDOWN2 Search, {Test}[Value{, Value...}], Var

LOOKDOWN 문은 8 비트의 상수를 Search 대상 리스트에서 찾습니다. 만약 발견된다면 일치한 상수의 인덱스 값이 Var 에 저장됩니다. 만약 발견된 값이 첫번째라면 Var 에는 0 이 저장됩니다. 리스트의 두번 째와 일치한다면 1 이, 세번 째와 일치한다면 2 가 Var 에 저장되며, 만약 일치하는 것이 발견되지 않는다면 Var 값은 변화되지 않습니다.

옵션 파라메터 Test 는 동등조건(=) 이외에 다른 조건 검사를 할 수 있습니다. 예를 들어 ">"를 테스트 파라메터로 지정하여 Search 보다 큰 값을 탐색합니다. Test 가 주어지지 않으면 "=" 이 적용됩니다. Value 리스트는 8 비트, 16 비트 숫자, 스트링상수 그리고 변수가 혼합될수 있습니다. 스트링에서 각각의 문자는 문자의 ASCII 값과 동등한 분리된 상수로 취급됩니다. 수식은 Value 리스트에 사용될 수 없습니다.

배열 첨자가 있는 배열 변수는 LOOKDOWN2 에 사용할수 없습니다. 85 개(PIC18Xxxx 에서 256)까지의 리스트 Value 로 사용할 수 있습니다.

LOOKDOWN2 명령은 LOOKDOWN 명령에 비교하여 3 배 가량의 코드가 만들어집니다. 서치 리스트가 8 비트 상수이거나 스트링일때 LOOKDOWN 을 사용합니다.

```
LOOKDOWN2    W0, [512, W1, 1024], B0
LOOKDOWN2    W0, > [1000, 100, 10], B0
```

5.43. LOOKUP

LOOKUP   Index, [Constant{, Constant... }], Var

LOOKUP 문은 8 비트 상수 테이블에서 값을 가져올 때 사용합니다. Index 가 0 이면 Var 에는 첫번째 Constant 값이 저장됩니다. Index 값이 1 이면 두번째 상수 값을 가져옵니다. 만약 Index 값이 상수리스트보다 크다면 Var 값은 변경되지 않습니다.

상수 리스트는 숫자와 스트링 상수가 혼용될 수 있습니다. 스트링에서의 각 문자는 문자의 ASCII 값의 분리된 상수로 취급됩니다. 인덱스가 가능한 배열변수는 상수 인덱스가 가능하더라도 LOOKUP 사용을 할 수 없습니다.

```
    For     B0 = 0 To 5          ' Count from 0 to 5
        LOOKUP  B0,  ["Hello!"],  B1     ' Get character number B0 from string to Variable B1
        SEROUT 0, N2400, [B1]         ' Send character in B1 to Pin0 Serially
    Next B0                                ' Do next character
```

5.44. LOOKUP2

LOOKUP2 Index, [Value{, Value...}], Var

LOOKUP 문은 8 비트 Value 테이블에서 값을 가져올 때 사용합니다. Index 가 0 이면 Var 에는 첫번째 Value 값이 저장됩니다. Index 값이 1 이면 두번 째 Value 값을 가져옵니다. 만약 Index 값이 Value 리스트보다 크다면 Var 값은 변경되지 않습니다.

상수 리스트는 16 비트 숫자와 스트링 상수가 혼용될 수 있습니다. 수식표현은 Index 값을 사용할 수 있다 하더라도 Value 리스트에 사용할 수 없습니다. 스트링에서의 각 문자는 문자의 ASCII 값의 분리된 상수로 취급됩니다. 인덱스가 가능한 배열변수는 상수 인덱스가 가능하더라도 LOOKUP 사용을 할 수 없습니다. 85 개(PIC18Xxxx 에서 256 개)의 Value 가 리스트에 사용될 수

있습니다.
LOOKUP2 는 LOOKUP 에 비하여 3 배의 코드를 만듭니다. Value 리스트가 8 비트 상수로 구성된다면 LOOKUP 을 사용합니다.

LOOKUP2 B0, [256, 512, 1024], W1

5.45. LOW
LOW Pin
지정한 Pin 을 Low 상태로 설정합니다. Pin 은 자동으로 출력상태가 됩니다. Pin 은 0 - 15 의 상수 이거나 0-15 를 저장하고있는 변수 또는 PORTA.0 과 같은 핀 이름입니다.

```
LOW     0       ' Make Pin0 an output and set it low ( 0 volts)
LOW     PORTA.0     ' Make PORTA, pin 0 an output and set it low ( 0 volts)
Led   var   PORTB.0       ' Define LED Pin
LOW     led         ' Make LED pin an output and set it low ( 0 volts)
```

다른 방법으로 핀이 이미 출력상태로 되어 있다면 더 빠르고 쉬운 방법으로 Low 상태로 할수 있습니다.

```
PORTB.0 = 0     ' Set PORTB, Pin 0 low
```

5.46. NAP
NAP Period
짧은시간동안 마이크로컨트롤러를 로우파워모드 상태로 합니다. NAP 동안에는 전력소모가 줄어듭니다. NAP 은 하나의 워치독 주기동안 슬립상태러 가게 합니다. 워치독이 동작하지 않는다면 영원히 또는 인터럽트가 동작하거나 리셋이 걸릴때까지 슬립상태로 갑니다.
Period 는 12 비트, 14 비트 코어를 포함하여 워치독 타이머 프리스케일러를 설정하는데 사용합니다.
17Cxxx 와 18Xxxx 와 같은 16 비트 코드 디바이스는 포스트 프리스케일러를 사용하며 프로그램시 워치독 타이머 주기를 설정합니다. 16 비트 코어 디바이스 사용시 NAP 명령에서 Period 를 무시합니다.
12-와 14-비트 코어 디바이스에서 주기를 나열하였으며 R/C 로 동작하는 워치독 타이머에서 산출하였으므로 근사치입니다. 개별 칩마다 또한 온도에 따라 편차가 클수있습니다.
NAP 은 워치독 타이머를 사용하므로 오실레이터 주파수와 관계가 없습니다.

Period	Delay(Approx)
0	18 milliseconds
1	36 milliseconds
2	72 milliseconds
3	144 milliseconds
4	288 milliseconds
5	576 milliseconds
6	1.153 seconds
7	2.304 seconds

```
NAP   7     ' Low power pause for about 2.3 seconds
```

5.47. ON DEBUG
ON DEBUG GOTO Label
ON DEBUG 는 모든 PicBasic Pro 명령에서 디버그 모니터 루틴이 실행 가능하도록 허용합니다. 이 방법은 ON INTERRUPT GOTO 와 비슷합니다. ON DEBUG GOTO 를 만나면 프로그램에서 모든 PicBasic Pro 명령이전에 삽입한 지정한 디버그 라벨을 호출합니다. DISABLE DEBUG 는 호출명령 삽입을 금지합니다.
모니터 루틴은 각명령이 활성화되기 이전에 쓰여져 있을것입니다. 이 루틴은 LCD 로 데이터를 보내거나 시리얼 통신 프로그램이 될것입니다. 모든 프로그램 정보가 표시되거나 이방법으로 변경할수 있습니다.
WORD 사이즈 시스템 변수는 BANK0 에 놓여지게 됩니다.

5.48. On INTERRUPT

ON INTERRUPT GOTO Label
ON ONTRERRUPT 는 PicBasic Pro 서브루틴이 마이크로컨트롤러 인터럽트로 사용될수 있게 합니다. PicBasic Pro 컴파일러는 2 가지 방법으로 인터럽트를 사용할수 있습니다. 첫번째는 어셈블리 언어를 사용하는것입니다. 이 방법은 인터럽트를 빠르게 그리고 간단히 처리합니다. 그러나 반드시 어셈블리언어를 사용하여야 하며 BASIC 코드를 사용하지 못합니다. 두번째 방법은 PicBasic Pro 인터럽트 핸들러를 사용하는 것입니다. 이 방법은 다시 설명할 것입니다.
두번째 방법은 인터럽트 처리기를 PicBasic Pro 명령으로 작성합니다. 이것은

PicBasic Pro 서브루틴으로 보이지만 RESUME 로 끝납니다.
인터럽트가 발생하면 플래그가 설정되며 현재 실행중인 PicBasic Pro 명령이 완료된후 Label 위치의 인터럽트 핸들러로 점프합니다. 인터럽트 핸들러가 종료되면 RESUME 문은 인터럽트가 발생했던 원위치로 돌아갑니다.
DISABLE 과 ENABLE 은 인터럽트 가능성 없이 실행되도록 하기 위하여 PicBasic Pro 프로그램의 다른 섹션에 있을 수 있습니다. 또는 인터럽트 핸들러를 ON INTERRUPT 문 이전에 오게 합니다. 이렇게 하여 ON INTERRUPT 문을 만나기 이전에는 인터럽트 체크를 하지 않도록 합니다.
지연시간(Latency)은 실제의 인터럽트가 발생한 시간으로 부터 인터럽트 핸들러로 진입하기까지의 시간입니다. PicBasic Pro 문은 재사용형식(하나의 문이 실행될 때 다른 곳에서 현재 사용중인 문을 호출하여 사용하는 방법) 이 아니기 때문에 인터럽트 루틴에 진입하기 이전에 지연시간을 예상하는 것은 가능합니다. PBP 는 현재의 실행중인 문을 완료하지 않고는 인터럽트 핸들러로 들어갈 수 없습니다. 실행문이 인터럽트를 발생하기 전에 PAUSE 나 SERIN 을 실행 중이라면 지연시간이 발생합니다. 프로그램을 설계할때에 항상 이런일이 있을 수 있음을 고려하여야 합니다. 인터럽트가 좀더 빨리 실행되기를 원하면 어셈블리 언어 인터럽트 루틴을 사용하여야 합니다.
오버헤드(과도한 작업)도 고려하여야 합니다. ON INTERRUPT 는 모든 문장 앞에 인터럽트가 발생하였는지를 검사하는 코드가 부가됩니다. DISABLE 은 이 코드의 생성을 금지합니다. ENABLE 은 다시 코드를 생성합니다. 보통 추가적인 코드는 많지는 않습니다만 작은 마이크로 컨트롤러에서 큰 프로그램은 문제가 됩니다.
한 개 이상의 ON INTERRUPT 를 프로그램에서 사용할 수 있습니다.

```
ON INTERRUPT GOTO myint      ' Interrupt handler is myint
INTCON = %10010000           ' Enable RB0 interrupt

.................

DISABLE                      ' Disable interrupts in handler
Myint: led = 1               ' Turn on LED when interrupted
RESUME                       ' Return to main program
ENABLE                       ' Enable interrupts after handler
```

인터럽트를 중지하기 원하면 (또는 다시 필요할때 까지) 다시 ON INTERRUPT 를 사용합니다. 그러면 INTCON 이 $80 이 됩니다.

```
INTCON = $80
```

5.49. OUTPUT

OUTPUT Pin
지정한 핀을 출력상태로 합니다. 핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.

```
OUTPUT 0    ' Make Pin0 an output
OUTPUT PORTA.0        ' Make PORTA, Pin 0 an output
```

다른 방법으로 핀을 출력상태로 만드는 더빠르고 쉬운 방법이 있습니다.

```
TRIS.0 = 0      ' Set PORTB, pin 0 to an output
```

포트의 모든 핀을 출력으로 설정하려면 TRIS 를 한번 설정하면 됩니다.

```
TRIS = %00000000      ' Set all of PORTB to outputs
```

5.50. OWIN
OWIN Pin, Mode, [Item...]
옵션이며 One-Wire 디바이스로 리셋펄스를 송신한후 하나 또는 여러개의 비트 또는 데이터의 바이트를 읽습니다. 옵션으로 다른 리셋 펄스로 종료됩니다.
핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.
Mode 는 리셋 펄스를 보내고 오퍼에이션 이전/이후에 보내는 순서와 데이터가 비트 또는 바이트인지를 나타냅니다.
Mode 는 데이터 수신 전 후에 리셋을 보내거나 또는 그렇지 않음을 선택하며 비트 또는 바이트와 같은 항목의 크기를 지정합니다.

Mode bit Number	Effect
0	1 = send reset pulse before data
1	1 = send reset pulse after data
2	0 = byte-sized data, 1 = bit-sized data

Mode 의 사용예입니다: Mode %0000 (십진수 0) 은 리셋이 없으며 바이트 크기입니다. Mode %001(십진수 1)은 리셋이전에 데이터와 바이트크기 데이터입

니다. Mode%100(깁진수 4)는 리셋이 없으며 비트 크기입니다.
Item 은 하나 또는 그 이상의 변수 또는 커마에 의하여 분리되는 수정자입니다. 사용가능한 수정자는 STR 이며 바이트 어레이 변수에서 데이터를 읽는 것과 입력값의 수를 건너뛰는 SKIP 입니다.
SKIP 과 STR 수정자는 12 비트 코어는 RAM 과 STACK 의 제한으로 사용할 수 없습니다.
OWEN PORTC.0 %000, [STR temperature\2, SKIP 4, count_remain, count_per_c]
이 예는 리셋펄스를 방생하지않고 PORTC 의 핀 0 에 연결된 One-Wire 디바이스에서 바이트를 수신합니다. 2 개의 바이트를 수신하며 바이트 어레이 변수 temperature 저장합니다 4 개의 바이트를 건너뛰고 2 개의 바이트를 분리된 변수로 저장합니다.

5.51 OWOUT

PWOUT Pin, Mode,[Item...]
One-wire 디바이스에 선택적으로 리셋펄스를 보내거나 한개 또는 여러개의 비트 또는 바이트를 보냅니다. 선택적으로 리셋펄스를 보내 종료합니다.
핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.
Mode 는 데이터 전송 전 후에 리셋을 보내거나 또는 그렇지 않음을 선택하며 비트 또는 바이트와 같은 항목의 크기를 지정합니다.

Mode bit Number	Effect
0	1 = send reset pulse before data
1	1 = send reset pulse after data
2	0 = byte-sized data, 1 = bit-sized data

Mode 의 사용예입니다: Mode %0000 (십진수 0) 은 리셋이 없으며 바이트 크기입니다. Mode %001(십진수 1)은 리셋이전에 데이터와 바이트크기 데이터입니다. Mode%100(깁진수 4)는 리셋이 없으며 비트 크기입니다.
Item 은 하나 또는 그 이상의 상수, 변수 또는 커마에 의하여 분리되는 수정자입니다. 사용가능한 수정자는 바이트 어레이로 부터 읽은 데이터를 보내는 STR 과 데이터를 반복해서 보내는 REP 입니다.
REP 과 STR 수정자는 12 비트 코어에서 RAM 과 STACK 의 제한으로 사용할 수 없습니다.
OWOUT PORTC.0, %001, [$cc, $be]
이 예는 PORTC 의 핀 0 에 연결된 One-Wire 디바이스로 리셋펄스를 방생하고 뒤이어 $cc 와 $be 를 송신합니다.

5.52. PAUSE

PAUSE Period
Pause 는 Period 로 지정한 밀리초 동안 프로그램 실행을 정지합니다. Period 는 16 비트 입니다. 따라서 가능한 최대 지연시간은 65,535 밀리초(1 분을 약간 넘습니다)입니다. NAP 과 SLEEP 과 같은 다른 시간 지연 함수와 달리 PAUSE 는 저전력 소비 상태로 되지 않습니다. 그러나 좀더 정밀합니다. 시스템 클록과 같은 정밀도를 갖습니다.
PAUSE 는 4 MHz 를 기준으로 합니다. 4MHz 이외의 오실레이터 주파수를 사용한다면 PBP 에게 DDEFINE 을 사용하여 알려 주어야 합니다.
PAUSE 1000 ' Delay for 1 second

5.53. PAUSEUS

PAUSEUS Period
Pause 는 Period 로 지정한 마이크로초동안 프로그램의 실행을 정지합니다. Period 는 16 비트 이므로 최대 지연 가능한 시간은 65535 마이크로초입니다. NAP 과 SLEEP 과 같은 다른 시간 지연 함수와 달리 PAUSE 는 저전력 소비 상태로 되지 않습니다. 그러나 좀더 정밀합니다. 시스템 클록과 같은 정밀도를 갖습니다.
PAUSEUS 는 오실레이터 주파수에따라서 동작하기위한 최소의 사이클을 사용하므로 마이크로초의 최소 숫자보다 작은 딜레이를 가질수 없습니다. 더 작은 딜레이 시간을 원한다면 어셈블리로 프로그램 하여야 합니다.

OSC	Minimum delay
3(3.58)	20us
4	24us
8	12us
10	8us
12	7us
16	5us
20	3us
24	3us
25*	2us
32*	2us
33*	2us

40**	2us

* PIC17Cxxx and PIC18Xxxx only.
** PIC18Xxxx only
PAUSE 는 4 MHz 를 기준으로 합니다. 4MHz 이외의 오실레이터 주파수를 사용한다면 PBP 에게 DDEFINE 을 사용하여 알려 주어야 합니다.

```
PAUSEUS   1000      ' Delay for 1 millisecond
```

5.54. PEEK
PEEK Address, Var
마이크로 프로세서 내부의 지정한 Address 위치의 레지스터를 읽어 Var 에 저장합니다.
AD 컨버터와 많은 I/O 를 가진 특별한 PIC 마이크로 MCU 에서 사용합니다. Address 가 상수일때 레지스터 번호의 내용이 Var 에 저장됩니다. Address 가 PORTA 와 같은 특수기능 레지스터일때 이 레지스터의 내용이 Var 에 저장됩니다. Address 가 RAM 일때 RAM 위치의 값이 먼저 읽고 이 값에 의하여 지정된 레지스터의 내용이 Var 에 저장됩니다.
그러나 모든 PIC 마이크로 MCU 레지스터는 PEEK 와 POKE 이외의 방법으로 구동할수 있습니다. 모든 PIC 마이크로 MCU 레지스터는 PicBasic Pro 에서 8-비트 변수로 취급되며 다른 바이트 사이즈 변수의 사용이 가능합니다. 레지스터는 직접 읽거나 식에 사용이 가능합니다.

```
B0 = PORTA      ' Get current PORTA pin states to B0
```

5.55. PEEKCODE
PEEKCODE Address, Var
코드 영역에서 지정한 Address 의 값을 읽어 Var 에 저장합니다.
PEEKCODE 는 PIC 마이크로 MCU 의 코드 영역에 저장된 데이터를 읽습니다. 지정한 Address 와 함께 호출하며 리턴된 값을 Var 에 저장합니다. 지정한 위치는 retlw 와 데이터값을 포함하고 있어야 합니다. POKECODE 는 이 값을 디바이스 프로그램시에 저장할수 있습니다.

```
    POKECODE     $3ff, OSCAL       ' Get OSCAL value for PIC12C671/12CE673
    PEEKCODE     $7ff, OSCAL       ' Get OSCAL value for PIC12C672/12CE674
```

5.56. POKE
POKE Address, Value
지정한 어드레스로 마이크로 프로세서 레지스터에 값을 저장합니다.
AD 컨버터와 많은 I/O 를 가진 특별한 PIC 마이크로 MCU 에서 사용합니다. Address 가 상수일때 Value 는 이 레지스터 번호에 놓여지게 됩니다. Address 가 PORTA 와 같은 특수기능 레지스터의 이름이면 Value 는 이 레지스터에 라이트 됩니다. Address 가 RAM 위치이면 RAM 위치의 내용을 먼저 읽고 이 값이 지정하는 어드레스에 Var 을 라이트 합니다.
그러나 모든 PIC 마이크로 MCU 레지스터는 PEEK 와 POKE 이외의 방법으로 구동할 수 있습니다. 모든 PIC 마이크로 MCU 레지스터는 PicBasic Pro 에서 8-비트 변수로 취급되며 다른 바이트 사이즈 변수의 사용이 가능합니다. 레지스터는 직접 쓰거나 식에 사용이 가능합니다.

```
TRISA = 0       ' Set PORTA to all outputs
PORTA.0   = 1     ' Set PORTA bit 0 high
```

5.57. POKECODE
POKECODE {@Address,}Value{,Value...}
디바이스 프로그램시 Value 를 현재의 프로그램 어드레스 또는 선택적인 지정 어드레스에 저장합니다.
POKECODE 는 PIC 마이크로 MCU 의 코드 영역에서 테이블을 생성할 때 사용합니다. 데이터 W 와 함께 return 을 만듭니다.
옵션인 Address 를 지정하지 않으면 데이터 저장은 앞의 프로그램 명령 이후 즉시 라이트 됩니다. 프로그램 흐름에 방해가 되지 않기 위하여 POKECODE 는 프로그램에 마지막에 위치하는것이 좋습니다. END 또는 STOP 뒤에 오도록 합니다.

```
    POKECODE     10, 20, 30        ' Store 10, 20, and 30 in code space
Generate:
      Retlw     10
      Retlw     20
      Retlw     30

POKECODE @$7ff, $94          ' Set OSCAL value for PIC12C672/12CE674
Generate:
      Org       7ffh
      retw      94h
```

5.58. POT

POT Pin, Scale, Var

Pin 에 연결된 가변 저항 또는 다른 저항성분을 읽습니다.

핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.

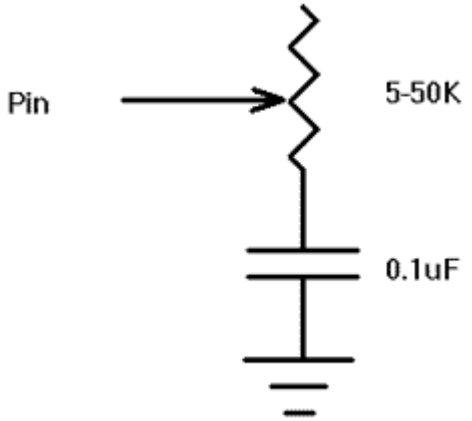
저항 값(보통 5K - 50K)은 레지스트를 통한 콘덴서의 방전시간으로 측정됩니다. Scale 은 RC 상수값을 교정합니다. 큰 RC 상수에서 Scale 은 작게(최소값은 1)하며 작은 RC 상수에서 Scale 은 크게 (최대값은 255)합니다. Scale 이 정확히 설정되면 Var 은 최소저항치에서 0 으로 근접하며 최대 저항치에서 255 로 근접합니다.

Scale 는 실험적으로 결정하여야 합니다.

최대 저항치에서 Scale 을 127 로 합니다. POT 명령을 실행하여 254 가 반환될때까지 Scale 을 교정합니다. 만약 255 가 반환되면 scale 을 작게 합니다. 반환된 값이 253 이거나 이보다 작으면 scale 을 증가합니다.

아래의 자동으로 한 예입니다. 반드시 최대 저항 치로 설정해 두어야 합니다.

```
B0      Var     Byte
Scale   Var     Byte

    For scale = 1 to 255
        POT 0, scale, B0
        IF (B0 > 253) Then calibrated
    Next scale
    Serout  2,0,["Increase R or C.", 10, 13]
    STOP

Calibrated:
   Serout 2,0,["Scale = ",#scale, 10, 13]
```



5.59. PULSIN

PULSIN Pin, State, Var

Pin 에 입력된 펄스의 폭을 측정합니다. State 가 0 이면 Low 펄스를 측정합니다. State 가 1 이면 펄스 High 를 측정합니다. 펄스 에지가 발생하지 않거나 펄스 폭이 측정 가능 범위보다 클 때 0 이 반환됩니다. 8 비트 Var 을 사용하면 16 비트 측정값의 LSB 만 반환합니다.

Pin 은 자동으로 입력으로 설정됩니다. 핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.

PULSEIN 의 분해능은 오실레이터 주파수에 따릅니다. 4MHz 오실레이터를 사용할때 10uS 단위의 결과가 됩니다. 20MHz 오실레이터를 사용하면 2 uS 의 분해능이 됩니다. PULSEIN 은 DEFINE 과 관계가 없습니다. 분해능은 언제나 실제의 오실레이터 주파수에 따릅니다. PULSEIN 은 65535 를 카운트할 때까지 입력펄스를 기다립니다. 대기 시간을 줄이고 싶을때 DEFINE 을 사용합니다.

```
DEFINE  PULSIN_MAX    1000
```

위의 DEFINE 은 RCTIME 에도 같이 적용됩니다.

```
' Measure high pulse on Pin4 stored in W3
PULSIN PORtB.4, W3
```

5.60. PULSOUT

PULSOUT Pin, Period

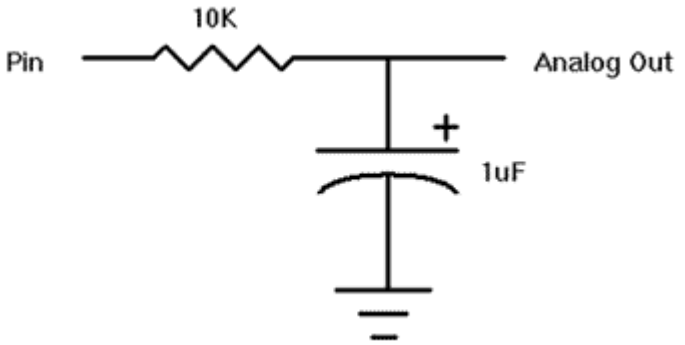
지정한 핀에 펄스를 발생합니다. 펄스는 핀을 2 번 토글링 합니다. 따라서 핀의 처음 상태가 펄스의 극성을 결정합니다. Pin 은 자동으로 출력상태로 됩니다. 핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.
PULSEIN 의 분해능은 오실레이터 주파수에 따릅니다. 4MHz 오실레이터를 사용할때 10uS 단위의 결과가 됩니다. 20MHz 오실레이터를 사용하면 2 uS 의 분해능이 됩니다. PULSEIN 은 DEFINE 과 관계가 없습니다.

```
' Send a pulse 1mSec long (at 4MHz) to Pin5
PULSOUT    PORTB.5, 100
```

5.61. PWM

PWM Pin, Duty,Cycle

지정한 핀에 펄스 폭 변조된 펄스열을 출력합니다. PWM 의 각 사이클은 256 스텝입니다. 가 PWM 사이클의 Dudy 는 0 (0%) 에서 255(100%)입니다. PWM 사이클은 Cycle 시간동안 반복됩니다. 핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.
PWM 의 Cycle Time 은 오실레이터 주파수에 의하여 결정됩니다. 4MHz 오실레이터를 사용하면 각 Cycle 시간은 5 mS 가 됩니다. 20MHz 를 사용하면 각 Cycle 시간은 1mS 가 됩니다. OSC 를 DEFINE 으로 설정하는것은 관계가 없습니다. Cycle 시간은 언제나 실제 사용된 오실레이터 주파수에 따릅니다.
연속적인 PWM 신호 출력을 원한다면 PWM 대신 HPWM 을 사용하여 하드웨어로 하여야 합니다. 유용하게 사용하려면 필터를 추가합니다. RC 회로를 이용하여 DA 컨버트로 사용할수 있습니다.




```
    PWM     PORTB.7, 127, 100     ; Send a 50% duty cycle PWM signal out Pin7 for
100 cycles
```

5.62. RANDOM

RANDOM Var

변수 Var 에 의사 랜덤수를 저장합니다. Var 은 16 비트 변수입니다. 어레이 변수는 상수인덱스를 사용할수 있으나 RANDOM 문에서 사용할 수 없습니다.

```
RANDOM W4     ' Randomize value in W4
```

5.63. RCTIME

RCTIME Pin, State, Var

RCTIME 은 특정 상태의 핀에 주어진 시간을 측정합니다. 기본적으로 PULSIN 의 절반입니다. Pin 은 자동으로 입력으로 설정됩니다. 핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.
RCTIME 은 가변저항을 읽는데 사용합니다. 저항은 콘덴서에 방전되는 시간을 측정합니다.
PULSEIN 의 분해능은 오실레이터 주파수에 따릅니다. 4MHz 오실레이터를 사용할때 10uS 단위의 결과가 됩니다. 20MHz 오실레이터를 사용하면 2 uS 의 분해능이 됩니다. PULSEIN 은 DEFINE 과 관계가 없습니다. 분해능은 언제나 실제의 오실레이터 주파수에 따릅니다. PULSEIN 은 65535 를 카운트할 때까지 입력펄스를 기다립니다. 대기 시간을 줄이고 싶을때 DEFINE 을 사용합니다.

```
DEFINE    PULSIN_MAX     1000
```

위의 DEFINE 은 PULSIN 에도 같이 적용됩니다.

```
Low PORTB.3             ' Discharge cap to start
Pause   10              ' Discharge for 10ms
RCTIME PORTB.3, 0, W0       ' Read potentiometer on Pin3
```

5.64. READ

READ Address, Var

프로세서에 내장된 EEPROM 에 어드레스를 지정하여 바이트를 읽어 Var 에 저장합니다. 이 명령은 PIC12F67x, 16F84, 16C84 그리고 16F87x 시리즈와 같이 EEPROM 을 내장한 PIC 마이크로 MCU 에서 사용할 수 있습니다.

READ 는 I2C 방식의 EEPROM 을 가진 PIC12CE67x 와 PIC16CE62x 에서 사용할 수 없습니다. 대신 I2CREAD 명령을 사용하여야 합니다.

2 바이트로 구성된 Word 형을 읽으려면 분리하여 읽어야 합니다.

```
W     Var     Word
      READ    0,w.BYTE0
      READ    1,w.BYTE1
```

5.65. READCODE

READCODE Address, Var

Address 로 지정한 위치의 Code 를 Var 에 저장합니다.

일부 PIC16Fxxx 그리고 PIC18Xxxx 디바이스는 실행시에 코드를 읽는것을 허용합니다. 추가적인 데이터 저장이나 프로그램 코드의 존재 유무를 검사할 수 있습니다.

PIC18Xxxx 디바이스에서 0 ~ 65536 사이의 어드레스 범위의 코드 BYTE 또는 WORD 크기의 데이터를 읽을 수 있습니다.

프로그램 어드레스를 결정하기 위하여 리스팅 파일을 검사할수 있습니다.

```
      READCODE   $100, w        ' Put the code word at location $100 into w
```

5.66. REPEAT..UNTIL

```
REPEAT
        Statement...
UNTIL Condition
```

UNTIL 에서 검사한 조건이 참이면 Statement 를 반복실행합니다. Condition 은 모든 비교 식입니다.

```
   I = 0
   REPEAT
      PORTB.0[i] = 0
      I = i + 1
   UNTIL   I   > 7
```

5.67. RESUME

RESUME {Label1}

인터럽트 처리후에 호출 위치로 돌아가도록 합니다. 옵션인 Label 은 인터럽트가 발생한 위치 대신에 지정한 Label 위치로 가도록 합니다. 이 경우 스택의 다른 복귀 어드레스는 더이상 구동할수 없습니다.

자세한것은 INTERRUPT 를 참고합니다.

```
Clockint: seconds = seconds + 1       ' Count time
          RESUME          ' Return to program after interrupt

Error:      High errorled       ' Turn on error LED
          RESUME   restart       ' Resume somewhere else
```

5.68. RETURN

REZTURN

서브루틴에서 복귀합니다. RETURN 은 서브루틴을 호출한 GOSUB 다음 문장으로 실행제어를 옮깁니다.

```
        Gosub     sub1          ' Go to subroutine labeled sub1

                ...
sub1: Serout     0,N2400,["Lunch"]        ' Send "Lunch" out Pin0 serially
        RETURN ' Return to main program after Gosub
```

5.69. REVERSE

REVERSE Pin

Pin 이 입력이라면 출력상태로 전환합니다. 핀이 출력상태면 입력상태로 전환합니다.
 핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.

```
OUT PUT     4          ' Make Pin4 an output
REVERSE     4          ' Change Pin4 to an input
```

5.70. SELECT CASE

```
SELECT CASE Var
    CASE Expr1 {, Expr ... }
        Statement
    CASE Expr2 {, Expr ...}
       Statement
    { CASE ELSE
        Statement... }
END SELECT
```

CASE 문은 다중의 IF.. THEN 을 사용하는 것보다 편리합니다. 이 실행문은 변수의 다른값을 비교하거나 값의 범위 그리고 값을 기반으로 한 동작 결정에 사용합니다. 모든 비교에 있 Var 는 SELECT CASE 문에서 지정한 모든 비교 조건에 사용합니다. IS 는 동등 조건 이외에 모두 적용됩니다. CASE 문에서 검사조건이 맞는 것이 없으면 옵션인 CASE ELSE 문을 실행합니다. END 문은 SELECT CASE 문을 닫습니다.

```
SELECT   CASE    x
       CASE    1
            Y = 10
      CASE      2, 3
   Y = 20
   CASE      IS   > 5
      Y = 100
   CASE ELSE
       Y = 0
END SELECT
```

5.71. SERIN

SERIN Pin , Mode, {Timeout, Label, }{[Qual...], } { Item... }
표준 비동기 통신 포맷 8 데이터버스, 패리티 없음, 1 스톱미트로 하나또는 그이상의 항목을 수신합니다. Pin 은 자동적으로 수신 상태가 됩니다. . 핀 번호는 0-15 의 상수, 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.
Mode 이름 (즉 T2400) 은 MODEFS.BAS 에서 정의되어 있습니다. 이를 이용하려면 다음라인을 넣습니다.

```
Include "modefs.bas
```

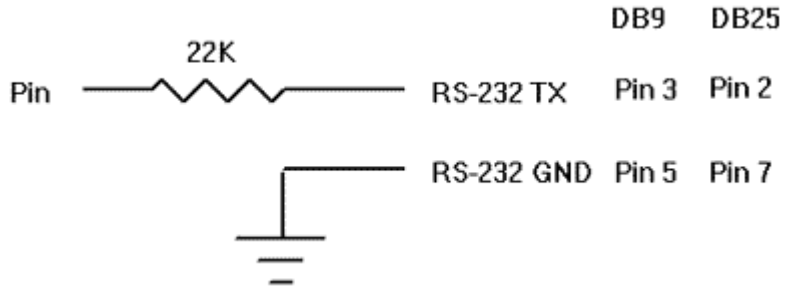
을 PICBASIC Pro 프로그램에 처음에 넣습니다. BS1DEFS.BAS 와 BS2DEFS.BAS 에는 이미 MODEFS.BAS 가 포함되어 있습니다. 이 파일을 사용하고 있으면 다시 포함시키지 않도록 합니다. Mode 번호는 이 파일을 포함하지 않아도 사용할 수 있습니다.

Mode	Mode No.	Baud Rate	State
T2400	0	2400	TRUE
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	Inverted
N1200	5	1200	
N9600	6	9600	
N300	7	300	

옵션인 Timeout 과 Label 은 일정시간동안 문자가 수신되지 않을때 프로그램을 계속 실행하도록 합니다. Timeout 은 1 밀리초 단위 입니다. 시리얼 입력핀이 1 밀리초동안 아이들 상태로 놓여진다면 프로그램은 SERIN 에서 빠져나오며 Label 로 점프합니다. 한개 또는 여러개의 브라켓에 쌓여진 검사기 이후헤 데이터 리스트가 놓여집니다. 검사과정에서 수신된 바이트와 일치하지 않으면 검사기는 반복 실행됩니다. (i.e. 다음에 수신된 바이트는 검사기 리스트에서 첫번째 항목과 비교 됩니다.) 검사항은 상수, 변수 또는 스트링 상수가 됩니다. 스트링의 각문자는 개별적인 검사항으로 취급됩니다.

검사항이 만족되면 SERIN 은 각 항목을 관련된 변수에 저장합니다. 변수가

하나만 설정되었다면 수신된 ASCII 문자 값은 변수에 저장됩니다. 변수앞에 # 기호가 붙어있으면 SERIN 은 십진수 값을 ASCII 로 변환하여 변수에 저장합니다. 숫자가 수신되기 전까지는 모든 문자가 무시됩니다. 숫자가 아닌 문자가 수신되면 종료되며 문자는 무시됩니다. SERIN 은 4MHz 오실레이터를 비트 타이밍 발생으로 사용합니다. 원하는 다른 바우드 레이트는 다른 오실레이터 주파수가 필요할 때 DEFINE 을 사용합니다.
일반적으로 저렴한 싱글 칩 RS-232 레벨 변환기를 사용합니다. PIC 마이크로 MCU 의 우수한 특성으로 대부분의 경우 레벨 컨버터가 필요치 않습니다. 반전 입력을 사용할 때 전류를 제한하기 위한 저항을 사용합니다.



SERIN 1, N2400, ["A"], B0 ' Wait until the character "A" is received serially on Pin1 and put next character into B0

5.72. SERIN2

SERIN2 DataPin{\FlowPin},Mode,{ParityLabel,}{Timeout,Label,}[Item…]
표준 비동기 시리얼 통신 규정으로 한 개 또는 여러 개의 항목을 수신합니다. SERIN2 는 BS2 의 Serin 커맨드와 비슷합니다. DataPin 은 자동으로 입력 상태가 됩니다. 옵션인 FlowPin 은 자동으로 출력상태가 됩니다. DataPin 과 FlowPin 은 0 – 15 의 상수이거나 0 – 15 를 저장한 변수 또는 PORTA.0 과 같은 포트이름입니다.
옵션 플로우 컨트롤핀 FlowPin 은 수신시 오버런을 방지하기 위하여 사용할 수 있습니다. 사용된다면 FlowPin 은 자동적으로 각 문자의 통신을 허용할 때 활성화 됩니다. 활성화 상태는 Mode 에서 지정한 극성에 따릅니다.
Mode 는 바우드레이트와 동작 파라메터를 지정하는데 사용합니다. 하위 13 비트는 바우드레이트를 설정합니다. Bit13 은 패리티 유/무를 선택합니다. Bit 14 는 인버트/노멀 레벨 상태를 선택합니다. Bit15 는 사용하지 않습니다.
바우드 레이트 비트는 microsecond-20 으로 비트 시간을 지정합니다. 바우드 레이트가 주어지면 아래식으로 설정값을 산출합니다.
(1000000/baud)-20
몇 가지 표준 바우드 레이트를 아래 표에 나열합니다.

Baud Rate	Bits 0 – 12
300	3313
600	1646
1200	813
2400	396
4800	188
9600*	84
19200*	32

* 4MHz 보다 빠른 오실레이터가 필요합니다.
Bit13 은 패리티비트 유(bit 13 = 1) 또는 무(bit13 = 0)를 선택합니다. 일반적으로 시리얼 통신은 8N1(8 데이터 비트, 패리티 없음, 1 스톱 비트)을 사용합니다. Even Parity 대신에 Odd Parity 를 사용하려면 다음과 같이 프로그램에서 DEFINE 을 사용합니다.
DEFINE SER2_ODD 1
Bit 14 는 데이터와 플로우 컨틀로핀의 레벨을 선택합니다. Bit = 0 이면 데이터는 RS-232 드라이버를 사용합니다. Bit14 = 1 이면 반전된 데이터를 수신하며 RS-232 드라이버를 사용하지 않습니다.
Mode 의 사용 예 입니다:
Mode = 84 (9600 baud, no parity, true), Mode = 16780 (2400 baud, no parity, inverted), Mode = 27889(300 baud, even parity, inverted) 추가적인 Mode 의 예는 Appendix A 에 있습니다.
ParityLabel 이 포함되면 문자 수신시 parity 에라가 발생할 때 Label 로 점프합니다. 이것은 Parity 를 사용할 때(bit 13 = 1)사용 합니다.
옵션인 Timeout 과 Label 은 문자가 일정시간동안 수신되지 않을 때 프로그램이 계속 실행되기 위하여 사용합니다. Timeout 은 1 millisecond 단위 입니다. Timeout 시간동안 시리얼 입력핀에 데이터가 수신되지 않으면 SERIN2 커맨드는 종료되며 Label 로 점프합니다.
8 비트 이외(7 비트, 패리티 포함)의 데이터를 사용하려면 DEFINE 을 사용합니다. SER2_BITS 데이터 비트는 4 비트부터 8 비트(DEFINE 지정하지 않은 디폴트 값) 범위 입니다. 패리티를 사용하게 되면 1 개 비트가 추가 됩니다. SER2_BITS 를 9 로 설정하면 8 개의 데이터 비트와 9 번째 패리티 비트가 됩니다.

패리티가 금지(디폴트)된 상태에서:

```
DEFINE SER2_BITS 4     ' Set Serin2 and Serout2 data bits to 4
DEFINE SER2_BITS 5     ' Set Serin2 and Serout2 data bits to 5
DEFINE SER2_BITS 6     ' Set Serin2 and Serout2 data bits to 6
DEFINE SER2_BITS 7     ' Set Serin2 and Serout2 data bits to 7
DEFINE SER2_BITS 8     ' Set Serin2 and Serout2 data bits to 8 (디폴트)
```

패리티 사용상태에서:

```
DEFINE SER2_BITS 5     ' Set Serin2 and Serout2 data bits to 4
DEFINE SER2_BITS 6     ' Set Serin2 and Serout2 data bits to 5
DEFINE SER2_BITS 7     ' Set Serin2 and Serout2 data bits to 6
DEFINE SER2_BITS 8     ' Set Serin2 and Serout2 data bits to 7 (디폴트)
DEFINE SER2_BITS 9     ' Set Serin2 and Serout2 data bits to 8
```

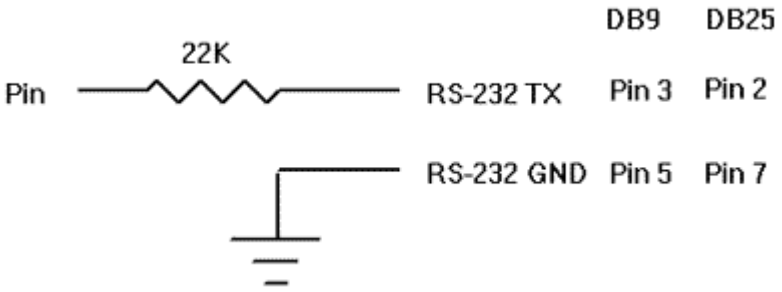
SERIN2 는 다양한 데이터 수정자를 혼용하여 사용할수 있습니다.

Modifier	Operation
Bin{1..16}	바이너리 디지트를 수신합니다.
DEC{1..5}	십진수를 수신합니다.
HEX{1..4}	대문자 HEX 값을 수신합니다.
SKIP n	N 개 수신 문자를 건너뜁니다.
STR ArrayVar\n{\c}	N 문자의 스트링을 수신하며 종료문자 c 를 지정할 수 있습니다.
WAIT( )	문자 입력을 기다립니다.
WAITSTR ArrayVar{\n}	스트링 입력을 기다립니다.

1) 변수이름 앞에 BIN 을 지정하면 ASCII 문자는 바이너리 값입니다. 예를 들어 BIN B0 는 "1000" 의 문자를 수신했을 때 B0 는 8 이 저장됩니다.
2) 변수이름 앞에 DEC 를 지정하면 수신한 ASCII 문자는 십진수 입니다. 예를 들어 DEC B0 는 "123"을 수신했을 때 B0 는 123 이 저장됩니다.
3) 변수이름 앞에 HEX 를 지정하면 수신한 ASCII 문자는 핵사데시멀(16 진수) 입니다. 예를 들어 HEX B0 는 "FE"을 수신했을 때 B0 는 254 이 저장됩니다.
4) SKIP 은 입력 스트림(데이터 열)에서 지정한 n 개의 문자를 건너뜁니다. 예를 들어 SKIP 4 는 4 개 문자를 건너 뜁니다.
5) STR 뒤이은 어레이 변수이름, 문자 수 그리고 옵션인 종료문자는 스트링을 수신합니다. 스트링 길이는 카운트 숫자 N 또는 종료문자 C 를 만날 때 결정됩니다.
6)수신한 데이터 항목은 WAIT 에서 괄호 사이에 지정한 한 개 또는 그 이상의 검사문자를 포함할 수 있습니다. SERIN2 는 데이털르 수신하기 전에 정해진 순서로 검사 문자를 수신하여야 합니다. 검사 문자 순서에서 다음 문자가 일치 하지 않으면 문자 검사는 처음부터 다시 시작합니다. 검사문자는 상수, 변수 또는 스트링이 될수 있습니다. 스트링의 각문자는 개별적인 검사 문자로 사용됩니다.
7) WAITSTR 은 SERIN2 에서 문자 스트링을 기다립니다.

WAIT 또는 WAITSTR 검사항이 만족되면 , SERIN2 는 각 항목의 데이터를 변수에 저장하기 시작합니다. 변수가 1 개라면 수쉰된 ASCII 문자가 변수에 저장됩니다. 변수앞에 BIN, DEC 또는 HEX 가 있으면 SERIN2 는 바이너리, 십진수, 또는 헥사데시말로 변환하여 저장합니다. 숫자가 오기전에 문자가 온다면 모두 무시됩니다. 숫자가아민 문자가 올때 입력은 종료처리됩니다. BIN, DEC 그리고 HEX 는 숫자가 따를것입니다. 일반적으로 이들 수정자는 입력으로 많은 디지트를 수신합니다. 그러나 숫자가 수정자를 따라온다면 SERIN2 는 디지트 수를 수신하며 필요하면 여분의 디지트를 건너뜁니다.
SERIN2 는 비트 타이밍을 발생할때 4MHz 오실레이터를 기준으로 합니다. 다른 바우드 레이트는 적합한 다른 오실레이터 주파수를 사용하여야 합니다. DEFINE 을 사용하여 오실레이터 주파수를 설정할 수 있습니다. 9600BPS 보다 큰 통신속도를 원할때 4MHz 보다 빠른 오실레이터 주파수를 사용하여야 합니다.
단일 칩 RS-232 레벨 컨버터를 사용합니다만 PIC 마이트로 MCU 의 우수한 I/O 특성 덕분에 RS-232 를 구현할수 있습니다. 대부분의 응용에서 레벨 컨버터를 필요로하지 않습니다. 반전된 TTL (Mode bit 14 = 1) 을 사용할 수 있습니다. 전류제한 저항을 사용할것을 권장합니다.

SERIN2 는 RAM 과 STACK 의 제한으로 12 비트 프로세서에서 사용할 수 없습니다.




```
' Wait until the character "A" is received serially on Pin1 and put next character into B0
SERIN2    1,16780,[WAIT("A"),B0
' Skip 2 chars and grap a 4 digit decimal number
SERIN2 PORTA.1, 84,[SKIP 2, DEC4 B0]
SERIN2 PORTA.1\PORTA.0, 84,100,tlabel,[WAIT ("x",b0), STR ar]
```

5.73. SEROUT

SEROUT Pin, Mode,[Item{,Item...}]

표준 비동기 시리얼 통신 규정으로 한 개 또는 여러 개의 항목을 송신합니다. SEROUT 은는 BS2 의 Serout 커맨드와 비슷합니다. DataPin 은 자동으로 출력상태가 됩니다. DataPin 과 FlowPin 은 0 – 15 의 상수이거나 0 – 15 를 저장한 변수 또는 PORTA.0 과 같은 포트이름입니다.

Mode 이름(예: T2400)은 MODEFS.BAS 에 정의되어 있습니다. 이를 사용하려면 아래와 같이 Pic Basic Pro 프로그램 처음에 추가합니다.

```
Include "modefs.bas"
```

BS1DEFS.BAS 와 BS2DEFS.BAS 에는 이미 MODEF.BAS 를 포함하고 있으므로 다시 지정하면 안됩니다. Mode 번호는 이 파일을 포함하지 않아도 사용할 수 있습니다.

Mode	Mode NO.	Baud Rate	State
T2400	0	2400	Driven True
T1200	1	1200	
T9600	2	9600	
T300	3	300	
N2400	4	2400	Driven Inverted
N1200	5	1200	
N9600	6	9600	
N300	7	300	
OT2400	8	2400	Open True*
OT1200	9	1200	
OT9600	10	9600	
OT300	11	300	
ON2400	12	2400	Open Inverted*
ON1200	13	1200	
ON9600	14	9600	
ON300	15	300	

* Open 모드는 12 비트 PIC 마이크로 MCU 에서 사용할 수 없습니다.

SEROUT 은 3 개의 다른 데이터 타입을 지원하며 자유롭게 혼용하여 사용할 수 있습니다.

1) 스트링 상수는 문자의 집합으로 출력이 됩니다.

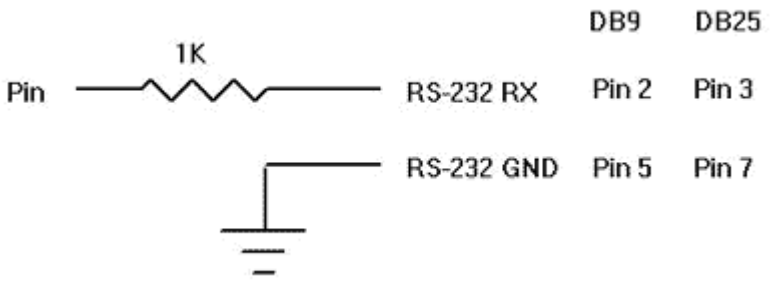
2) 숫자 값은 (변수 또는 상수) 해당 ASCII 코드로 출력합니다. 13 은 캐리지 리턴(CR)이며 10 은 라인피드(LF)입니다.

3) 숫자값 앞에 파운드기호(#)가 있으면 십진값의 ASCII 코드를 송출합니다. 예를들어 Wo - 123 일때 #W0(또는 #123)은 "1", "2", "3"을 송출합니다.

SEROUT 은 4MHz 의 오실레이터 주파수를 기준으로 비트 타이밍을 발생합니다. 다른 바우드 레이트에서 적절한 오실레이터 주파수를 선택하여야 합니다. 새로운 오실레이터값을 설정하려면 DEFINE 을 사용합니다. 어떤경우엔 SEROUT 명령에 의하여 출력되는 속도가 수순 속도보다 빠른 경우가 있습니다. (바우드레이트는 같은나 단위시간당 문자수) 전송되는 문자 사이에 추가적인 시간을 설정할수 있습니다. DEFINE 을 사용하여 문자간의 간격을 설정할수 있습니다. DEFINE 문으로 설정할수 있는 문자 간격 시간은 1 에서 65536 uS 입니다.

DEFINE CHAR_PACING 1000

단일 칩 RS-232 레벨 컨버터를 사용합니다만 PIC 마이크로 MCU 의 우수한 I/O 특성 덕분에 RS-232 를 구현할 수 있습니다. 대부분의 응용에서 레벨 컨버터를 필요로 하지 않습니다. 반전된 TTL (Mode bit 14 = 1) 을 사용할 수 있습니다. 전류제한 저항을 사용할 것을 권장합니다.



SEROUT 0,N2400,[#B0,10] ' Send the ASCII value of B0 followed by a linefeed out Pin0 serially

5.74. SEROUT2

SEROUT2 DataPin{\FlowPin},Mode,{Pace,}{Timeout,Label,}[Item…]
표준 비동기 시리얼 통신 규정으로 한 개 또는 여러 개의 항목을 송신합니다. SEROUT2 는 BS2 의 Serout 커맨드와 비슷합니다. DataPin 은 자동으로 출력상태가 됩니다. 옵션인 FlowPin 은 자동으로 입력상태가 됩니다. DataPin 과 FlowPin 은 0 – 15 의 상수이거나 0 – 15 를 저장한 변수 또는 PORTA.0 과 같은 포트이름입니다.
옵션 플로우 컨트롤핀 FlowPin 은 수신 쪽에서 오버런을 방지하기 위하여 사용할 수 있습니다. 사용된다면 FlowPin 은 자동적으로 각 문자의 통신을 허용할 때 활성화 됩니다. 활성화 상태는 Mode 에서 지정한 극성에 따릅니다.
옵션인 Timeout 과 Label 은 FlowPin 이 일정시간동안 활성화 되지 않을 때 프로그램이 계속 실행되기 위하여 사용합니다. Timeout 은 1 millisecond 단위입니다. FlowPin Timeout 시간동안 비 활성화 상태이면 SEROUT2 커맨드는 종료되며 Label 로 점프합니다.

어떤경우에 SEROUT2 명령의 통신속도는 수신하기에 너무 빠를 수 있습니다. 플로우 컨트롤 하기 위하여 여분의 핀을 사용하는것은 바람직하지 않습니다. 문자간의 전송 간격시간을 1 부터 65536 millisecond 설정할 수 있습니다.

Mode 는 바우드 레이트와 동작 파라메터를 지정하는데 사용합니다. 하위 13 비트는 바우드 레이트를 설정합니다. Bit13 은 패리티 유/무를 선택합니다. Bit 14 는 인버트/노멀 레벨 상태를 선택합니다. Bit15 는 사용하지 않습니다.
바우드 레이트 비트는 microsecond-20 으로 비트 시간을 지정합니다. 바우드 레이트가 주어지면 아래식으로 설정값을 산출합니다.
(1000000/baud)-20
몇 가지 표준 바우드 레이트를 아래 표에 나열합니다.

Baud Rate	Bits 0 – 12
300	3313
600	1646
1200	813
2400	396
4800	188
9600*	84
19200*	32

* 4MHz 보다 빠른 오실레이터가 필요합니다.
Bit13 은 패리티비트 유(bit 13 = 1) 또는 무(bit13 = 0)를 선택합니다. 일반적으로 시리얼 통신은 8N1(8 데이터 비트, 패리티 없음, 1 스톱 비트)을 사용합니다. Parity 를 선택하면 데이터는 7E1(7 데이터, Even 패리티, 1 Stop 비트)로 송출합니다.
Even Parity 대신에 Odd Parity 를 사용하려면 다음과 같이 프로그램에서 DEFINE 을 사용합니다.

```
DEFINE  SER2_ODD   1
```

Bit 14 는 데이터와 플로우 컨틀롤 핀의 레벨을 선택합니다. Bit = 0 이면 데이터는 RS-232 드라이버를 사용합니다. Bit14 = 1 이면 반전된 데이터를 수신하며 RS-232 드라이버를 사용하지 않습니다.
Mode 의 사용 예 입니다:
Mode = 84 (9600 baud, no parity, true), Mode = 16780 (2400 baud, no parity, inverted), Mode = 60657(300 baud, even parity, inverted, open) 추가적인 Mode 의 예는 Appendix A 에 있습니다.

8 비트 이외(7 비트, 패리티 포함)의 데이터를 사용하려면 DEFINE 을 사용합니다. SER2_BITS 데이터 비트는 4 비트부터 8 비트(DEFINE 지정하지 않은 디폴트 값) 범위 입니다. 패리티를 사용하게 되면 1 개 비트가 추가 됩니다. SER2_BITS 를 9 로 설정하면 8 개의 데이터 비트와 9 번째 패리티 비트가 됩니다.

패리티가 금지(디폴트)된 상태에서:

```
DEFINE SER2_BITS 4     ' Set Serin2 and Serout2 data bits to 4
DEFINE SER2_BITS 5     ' Set Serin2 and Serout2 data bits to 5
DEFINE SER2_BITS 6     ' Set Serin2 and Serout2 data bits to 6
DEFINE SER2_BITS 7     ' Set Serin2 and Serout2 data bits to 7
DEFINE SER2_BITS 8     ' Set Serin2 and Serout2 data bits to 8 (디폴트)
```

패리티 사용상태에서:

```
DEFINE SER2_BITS 5     ' Set Serin2 and Serout2 data bits to 4
DEFINE SER2_BITS 6     ' Set Serin2 and Serout2 data bits to 5
DEFINE SER2_BITS 7     ' Set Serin2 and Serout2 data bits to 6
DEFINE SER2_BITS 8     ' Set Serin2 and Serout2 data bits to 7 (디폴트)
DEFINE SER2_BITS 9     ' Set Serin2 and Serout2 data bits to 8
```

SERIN2 는 다양한 데이터 수정자를 혼용하여 사용할 수 있습니다.

Modifier	Operation
Bin{1..16}	바이너리 디지트를 송신합니다.
DEC{1..5}	십진수를 송신합니다.
HEX{1..4}	대문자 HEX 값을 송신합니다.
REP c\n	문자 C 를 N 회 반복하여 송신합니다..
STR ArrayVar{\n}	스트링 입력을 기다립니다.

1) 스트링 상수를 출력하면 문자열이 출력됩니다.
2) 숫자값(상수 또는 변수)은 해당 ASCII 문자로 출력됩니다. 캐리지 리턴(CR)13 과 라인피트(LF) 10 이 뒤이어 출력됩니다.
3) 숫자값에 앞서 BIN 을 지정하면 바이너리의 해당 ASCII 코드가 출력됩니다. 예를들어 B0 = 8 일때 BIN B0(또는 BIN 8)은 "1000"을 송신합니다.
4) 숫자 값에 앞서 DEC 를 지정하면 해당 ASCII 코드가 출력됩니다. 예를들어 B0 = 123 일때 DEC B0(또는 DEC123)는 "123"을 출력합니다.
5) 숫자값에 앞서 HEX 를 지정하면 해당 ASCII 코드가 출력됩니다. 예를들어 B0 = 254 일때 HEX B0 (또는 HEX 254)는 "FE"를 송신합니다.
6) REP 에 뒤이어 문자와 카운트 숫자를 지정하면 문자가 반복 출력됩니다. 예를들어 REP "0"\4 는 "0000"을 송신합니다.
7) STR 에 이어 바이트 어레이 변수와 옵션 인 카운트 숫자가 오면 스트링을 송신합니다. 스트링 종료는 카운트 숫자로 지정하거나 스트링의 0 을 만나면 종료됩니다.

BIN, DEC 그리고 HEX 는 앞에 놓여지거나 또는 여러 옵션 파라메터 뒤에 올수 있습니다.
I 를 앞서 사용하면 출력은 "%", "#", 또는 "$" 가 바이너리, 십진수, 헥사데시멀에 앞서 출력됩니다.
S 를 앞에 놓으면 최상위 비트가 세트되어 있을때 출력은 "-" 이 앞에 음수인것을 표시하게 됩니다. PBP 의 모든 수식 연산은 무부호 처리되는것을 명심하여야 합니다. 그러나 무 부호 연산의 결과가 부호연산이 되게 할수 있습니다. 예를들어 B0 = 9 - 10 일때 DEC B0 는 "255"가 될것입니다. SDEC B0 는 최상위 비트를 보내므로 "-1"을 송신합니다. 약간의 기교에 의하여 무부호 숫자를 부호숫자로 보내는것입니다.
BIN, DEC 그리고 HEX 는 숫자에 따라올 수 있습니다. 일반적으로 이들 수정자는 정확하게 표시되며 많은 디지트가 필요합니다. 그러나 숫자가 수정자 뒤에 있으면 SEROUT2 는 필요한경우 앞에 0 을 송신하며 디지트 숫자를 송신합니다. 상위 순서 디지트를 교정하여야 합니다. 예를들어 BIN6 8 은 "001000" 그리고 BIN2 는 "00"을 송신합니다.
모든 수정자는 한번 조합될수 있습니다. (예를들어 ISDEC4 B0)

SEROUT2 는 비트 타이밍을 발생할때 4MHz 오실레이터를 기준으로 합니다. 다른 바우드 레이트는 적합한 다른 오실레이터 주파수를 사용하여야 합니다. DEFINE 을 사용하여 오실레이터 주파수를 설정할 수 있습니다. 9600BPS 보다 큰 통신속도를 원할때 4MHz 보다 빠른 오실레이터 주파수를 사용하여야 합니다.
단일 칩 RS-232 레벨 컨버터를 사용합니다만 PIC 마이트로 MCU 의 우수한 I/O 특성 덕분에 RS-232 를 구현할수 있습니다. 대부분의 응용에서 레벨 컨버터를 필요로하지 않습니다. 반전된 TTL (Mode bit 14 = 1) 을 사용할 수 있습니다. 전류제한 저항을 사용할 것을 권장합니다.

SERIN2 는 RAM 과 STACK 의 제한으로 12 비트 프로세서에서 사용할 수 없습니다.

```
' Send the ASCII value of B0 followed by a linefeed out    Pin0 serially at 2400 baud
SEROUT2 0, 16780,[DEC B0, 10]
'Send "B0 =" followed by the binary value of B0 out PORTA pin1 serially at 9600 baud
SEROUT2 PORTA.1, 84,["B0=", IHEX4 B0]
```

5.75. SHIFTIN

SHIFTIN DataPin, ClockPin, Mode,[Var{\Bits}...]

ClockPin 에 클록을 발생하여 동기식으로 DataPin 에 시리얼 쉬프트 데이터를 입력하여 Var 에 저장합니다. ClockPin 과 DataPin 은 0 – 15 의 상수이거나 0 – 15 를 저장한 변수 또는 PORTA.0 과 같은 포트 이름입니다. \Bits 는 옵션이며 시프트 하는 비트수를 지정합니다. 지정하지 않으면 변수 형에 관계없이 8 비트가 쉬프트 되는것으로 설정됩니다. Bits 쉬프트는 사용한 Mode 가 LSB 이든 MSB 이든 관계없이 낮은 순서의 비트부터 쉬프트 합니다. Mode 이름(즉 MSBPRE)는 MODEFS.BAS 에 정의되어 있습니다. 이것을 사용하려면

Include "modefs.bas"를 Pic Basic Pro 프로그램의 시작 부분에 추가합니다. 만약 이 파일이 이미 내포되어 있다면 다시 선언하면 안됩니다. Mode 번호는 이 파일을 내포하지 않아도 사용할수 있습니다. 일부 Modes 는 이름을 가지고 있지 않습니다.

Mode 0-3 은 아이들 상태에서 클록이 Low 입니다. 데이터 비트 구간 내에서 High 로 변한뒤 Low 로 돌아옵니다. Mode 4-7 은 아이들 상태에서 클록이 High 입니다. 데이터 비트 구간 내에서 Low 로 변한 뒤에 High 로 돌아옵니다.

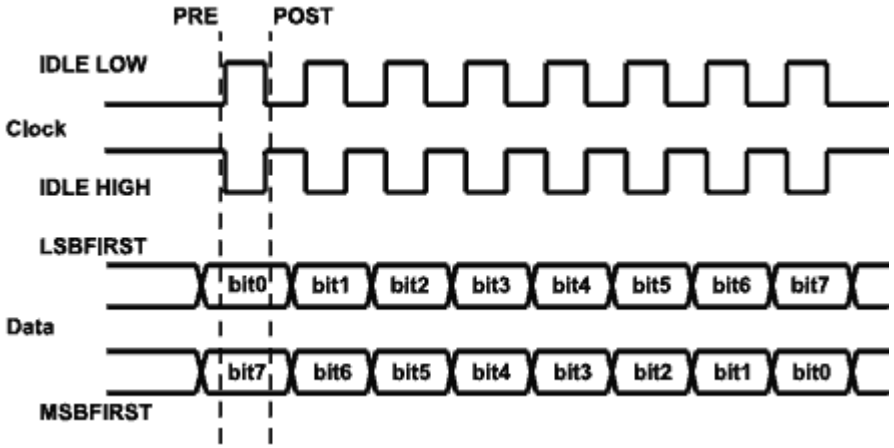
Mode	Mode No	Operation
MSBPRE	0	최상위 비트를 첫번째로 쉬프트 합니다. 클록을 보내기 전에 데이터를 데이터를 읽습니다. 아이들 상태에서 클록은 Low 입니다.
LSBPRE	1	최하위 비트를 첫번째로 쉬프트 합니다. 클록을 보내기 전에 데이터를 읽습니다. 아이들 상태에서 클록은 Low 입니다.
MSBPOST	2	최상위 비트를 첫번째로 쉬프트 합니다. 클록을 보낸 후 데이터를 데이터를 읽습니다. 아이들 상태에서 클록은 Low 입니다.
LSBPOST	3	최하위 비트를 첫번째로 쉬프트 합니다. 클록을 보낸 후 데이터를 읽습니다. 아이들 상태에서 클록은 Low 입니다.
	4	최상위 비트를 첫번째로 쉬프트 합니다. 클록을 보내기 전에 데이터를 데이터를 읽습니다. 아이들 상태에서 클록은 High 입니다.
	5	최하위 비트를 첫번째로 쉬프트 합니다. 클록을 보내기 전에 데이터를 읽습니다. 아이들 상태에서 클록은 High 입니다.
	6	최상위 비트를 첫번째로 쉬프트 합니다. 클록을 보낸 후 데이터를 데이터를 읽습니다. 아이들 상태에서 클록은 High 입니다.
	7	최하위 비트를 첫번째로 쉬프트 합니다. 클록을 보낸 후 데이터를 읽습니다. 아이들 상태에서 클록은 High 입니다.

쉬프트 클록은 약 50KHz 근처의 주파수이며 오실레이터 주파수에 따라 다릅니다. 액티브 상태는 최소한 2 uS 를 유지합니다. DEFINE 문을 사용하여 액티브 클록 상태를 65535(65.535 mS)까지 추가하여 낮은 클록 속도로 할 수 있습니다. 최소 추가 딜레이는 PAUSEUS 타이밍에 의하여 정의됩니다. 이 DEFINE 은 12-비트 PIC 마이크로 MCU 에서 사용할 수 없습니다.

예를들어 100uS 를 추가하여 클록 속도를 낮게 하려면

DEFINE SHIFT_PAUSEUS 100

아래의 그림은 여러 모드에 대하여 클록과 데이터의 관계를 보여줍니다.



SHIFTIN 0,1,MSBPRE, [B0, B1\4]

5.75. SHIFTOUT

SHIFTOUT DataPin, ClockPin, Mode,[Var{\Bits}...]
ClockPin 은 ClockPin 과 DataPin 에 Var 에 저장된 데이터를 동기식으로 쉬프트 아웃합니다. ClockPin 과 DataPin 은 0 – 15 의 상수이거나 0 – 15 를 저장한 변수 또는 PORTA.0 과 같은 포트 이름입니다. \Bits 는 옵션이며 시프트 하는 비트수를 지정합니다. 지정하지 않으면 변수 형에 관계없이 8 비트가 쉬프트 되는 것으로 설정됩니다. Bits 쉬프트는 사용한 Mode 가 LSB 이든 MSB 이든 관계없이 낮은 순서의 비트부터 쉬프트 합니다. 한 개의 변수에 대하여 16 개 Bits 까지 쉬프트 아웃 할 수 있습니다. 16 비트 이상이 필요하면 변수를 여러 개 사용하거나 대괄호[ ] 안에 상수를 포함합니다.
Mode 이름(즉 LSBFIRST)는 MODEFS.BAS 에 정의되어 있습니다. 이것을 사용하려면

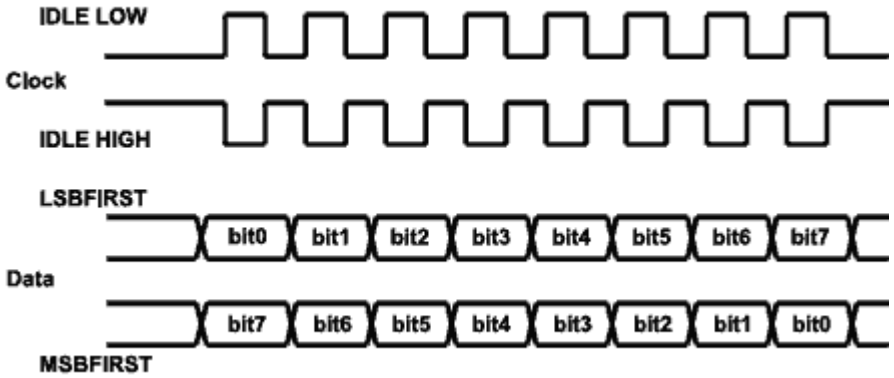
Include "modefs.bas"를 Pic Basic Pro 프로그램의 시작 부분에 추가합니다. 만약 이 파일이 이미 내포되어 있다면 다시 선언하면 안됩니다. Mode 번호는 이 파일을 내포하지 않아도 사용할 수 있습니다. 일부 Modes 는 이름을 가지고 있지 않습니다.
Mode 0-1 은 아이들 상태에서 클록이 Low 입니다. 데이터 비트 구간 내에서 High 로 변한 뒤에 Low 로 돌아옵니다. Mode 4-5 은 아이들 상태에서 클록이 High 입니다. 데이터 비트 구간 내에서 Low 로 변한 뒤에 High 로 돌아옵니다.

Mode	Mode No	Operation
LSBFIRST	0	최하위 비트부터 쉬프트 아웃 합니다. 클록의 아이들 상태는 Low 입니다.
MSBFIRST	1	최상위 비트부터 쉬프트 아웃 합니다. 클록의 아이들 상태는 Low 입니다.
	4	최하위 비트부터 쉬프트 아웃 합니다. 클록의 아이들 상태는 High 입니다.
	5	최상위 비트부터 쉬프트 아웃 합니다. 클록의 아이들 상태는 High 입니다.

쉬프트 클록은 약 50KHz 근처의 주파수이며 오실레이터 주파수에 따라 다릅니다. 액티브 상태는 최소한 2 uS 를 유지합니다. DEFINE 문을 사용하여 액티브 클록 상태를 65535(65.535 mS)까지 추가하여 낮은 클록 속도로 할 수 있습니다. 최소 추가 딜레이는 PAUSEUS 타이밍에 의하여 정의됩니다. 이 DEFINE 은 12-비트 PIC 마이크로 MCU 에서 사용할 수 없습니다.
예를들어 100uS 를 추가하여 클록 속도를 낮게 하려면

DEFINE SHIFT_PAUSEUS 100

아래의 그림은 여러 모드에 대하여 클록과 데이터의 관계를 보여줍니다.




```
SHIFTOUT 0,1,MSBFIRST,[B0,B1]
SHIFTOUT PORTA.1,PORTA.2,1,[wordvar\4]
SHIFTOUT PORTC.1,PORTB.1,4,[$1234\16, $56]
```

5.74. SLEEP

SLEEP Period
지정한 Period 시간동안 마이크로 컨트롤러가 저전력 동작상태가 되게 합니다. Periods 는 16 비트이며 딜레이는 65536 초 (18 시간이상)가능합니다.
SLEEP 은 Watchdog 타이머를 사용하므로 오실레이터 주파수에 관계없습니다. 보통 2.3 초이며 디바이스의 사양과 온도에 따라 다릅니다. 이러한 편차는 BASIC Stamp 와 다릅니다. PIC 마이크로 MCU 가 레셋될 때 내부레지스터가 초기화 되므로 변경되게 됩니다. 이 값은 프로그램의 예상하는것보다 많이 다릅니다. 교정하지 않은상태에서 SLEEP 커맨드가 동작하면 이 논의는 다른 것이 됩니다.

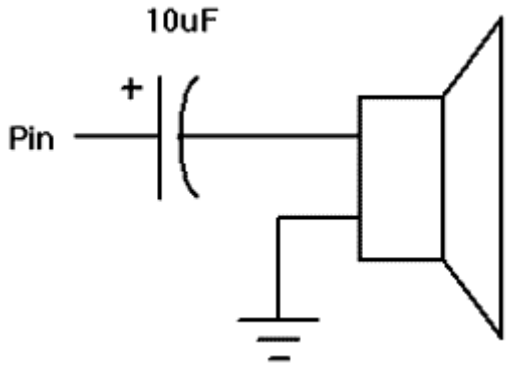
```
SLEEP   60    '   Sleep for about 1 minute
```

5.75. SOUND

SOUND *Pin*,[*Note*,*Duration*{,*Note*,*Duration*...}]

음과 함께 화이트 노이즈를 발생합니다. Pin 은 자동적으로 출력이 됩니다. Pin 은 상수이거나, 0-15 를 포함하는 변수이거나 PORTA.0 과 같은 포트 이름입니다.

Note0 은 아무소리나지 않습니다. Note 1-127 은 음을 발생합니다. Note 128-255 는 백색잡음입니다. 음과 백색 노이즈는 오름 차순입니다. (즉 1 과 128 이 낮은 주파수이며 127 과 255 가 높은 주파수 입니다.) Note 1 은 약 78.74 Hz 이며 Note 127 은 10,000hz 입니다.
Duration 은 0-255 이며 얼마나 긴 시간 Note 를 출력할 것인지를 결정하며 12 mS 단위로 증가합니다. Note 와 Duration 이 상수가 되지는 않습니다.
SOUND 출력은 TTL 레벨 구형파입니다. PIC 마이크로 MCU 의 우수한 I/O 특성덕분에 캐패시터를 통하여 스피커에 직접 연결하는 것이 가능합니다. 콘덴서 용량값은 관심있는 주파수와 스피커 임피던스에 따라 결정합니다. 피에조 스피커는 직접 연결할수 있습니다.




```
SOUND PORTB.7,[100,10,50,10] ' Send 2 sounds consecutively
to Pin7
```

5.75. STOP
STOP
프로그램 실행을 정지하며 무한 루프 상태로 진입합니다. 이 명령은 마이크로 프로세서를 저전력모드로 하지는 않습니다. 마이크로 컨트롤러는 영원히 동작하고 있습니다.

```
STOP ' Stop program dead in its tracks
```

5.77. SWAP
SWAP *Variable*,*Variable*
2 개의 변수값을 서로 교환합니다. 일반적으로 2 개의 변수값을 교환하는 것은 번거로운 일입니다. SWAP 은 임시 변수가 필요없이 하나의 명령으로 실행됩니다. 비트, 바이트 , 워드 변수를 사용할수 있습니다. 인덱스 변수가 있는 어레이 변수는 상수 인덱스가 허용되더라도 SAWP 을 사용할수 없습니다.

```
temp = B0 ' Old way
B0 = B1
B1 = temp
SWAP B0,B1 ' New way
```

5.78. TOGGLE
TOGGLE *Pin*
지정한 Pin 의 상태를 반전합니다. Pin 은 상수, 0-15, 또는 변수 또는 PORTA.0 과 같은 포트이름 입니다.

```
Low 0 ' Start Pin0 as low
TOGGLE 0 ' Change state of Pin0 to high
```

5.79. USBIN
USBIN *Endpoint*,*Buffer*,*Countvar*,*Label*
Endpoint 로부터 USB 데이터를 가져와 Buffer 에 저장합니다. Buffer 는 데이터를 저장할 적절한 길이의 바이트 어레이 입니다. Countervar 은 Buffer 에 전달하는 바이트 숫자를 저장하고 있습니다. Label 은 아무런 데이터가 없을 때 점프할 위치 입니다.
이 명령은 프로세서에 USB 포트가 내장된 PIC16C745 와 16C765 에서 사용할 수 있습니다.
USB 서브디렉토리에는 수정된 USB 라이브러리가 예제프로그램과 같이 있습

니다. USB 프로그램은 여러가지 추가적인 파일이 필요하며 어떤 파일은 사용자의 특정 용도를 위하여 수정이 필요합니다. USB 서브 디렉토리에 USB 커맨드에 자세한 문서가 있습니다.
USB 통신은 동기통신(SHIFTIN 그리고 SHIFTOUT)과 비동기 통신(SERIN, SEROUT)에 비교하여 복잡합니다. 많은 USB 사전 지식이 필요합니다. Microchip 의 웹사이트에서 USB 정보를 스터디 합니다. 또한 Jan Axelson 저서인 USB Complete"가 유용합니다.

```
USBIN 1, buffer, cnt, idleloop
```

5.80. USBINIT

USBINIT

USB 통신을 사용할 때 USBINT 는 프로그램의 처음 명령으로 사용합니다. PIC 마이크로 MCU 의 USB 를 초기화 합니다. 그리고 USB 버스가 초기화되고 사용가능한 상태를 기다립니다.

이 명령은 프로세서에 USB 포트가 내장된 PIC16C745 와 16C765 에서 사용할 수 있습니다.
USB 서브디렉토리에는 수정된 USB 라이브러리가 예제프로그램과 같이 있습니다. USB 프로그램은 여러가지 추가적인 파일이 필요하며 어떤 파일은 사용자의 특정 용도를 위하여 수정이 필요합니다. USB 서브 디렉토리에 USB 커맨드에 자세한 문서가 있습니다.
USB 통신은 동기통신(SHIFTIN 그리고 SHIFTOUT)과 비동기 통신(SERIN, SEROUT)에 비교하여 복잡합니다. 많은 USB 사전 지식이 필요합니다. Microchip 의 웹사이트에서 USB 정보를 스터디 합니다. 또한 Jan Axelson 저서인 USB Complete"가 유용합니다.

```
USBINIT
```

5.81. USBOUT

USBOUT *Endpoint,Buffer,Count,Label*

배열 변수 Buffer 에서 Count 에 저장된 바이트수의 바이트를 읽어 USB Endpoint 로 보냅니다. USB 버퍼가 여유 공간이 없어 전송이 되지 않으면 프로그램은 Label 로 점프합니다.

이 명령은 프로세서에 USB 포트가 내장된 PIC16C745 와 16C765 에서 사용할 수 있습니다.
USB 서브디렉토리에는 수정된 USB 라이브러리가 예제프로그램과 같이 있습니다. USB 프로그램은 여러가지 추가적인 파일이 필요하며 어떤 파일은 사용자의 특정 용도를 위하여 수정이 필요합니다. USB 서브 디렉토리에 USB 커맨드에 자세한 문서가 있습니다.
USB 통신은 동기통신(SHIFTIN 그리고 SHIFTOUT)과 비동기 통신(SERIN, SEROUT)에 비교하여 복잡합니다. 많은 USB 사전 지식이 필요합니다. Microchip 의 웹사이트에서 USB 정보를 스터디 합니다. 또한 Jan Axelson 저서인 USB Complete"가 유용합니다.

```
USBOUT 1, buffer,4,outloop
```

5.82. WHILE..WEND

WHILE *Condition*
Statement...
WEND

검사조건 Condition 이 참일 때 Statements 를 반복 실행합니다. Condition 이 더 이상 참이아닐 때 WEND 이하를 실행합니다. Condition 은 모든 비교수식입니다.

```
i = 1
WHILE i <= 10
Serout 0,N2400,[”No:”,#i,13,10]
i = i + 1
WEND
```

5.83. WRITE

WRITE *Address,Value*

내장된 EEPROM 에 지정한 어드레스로 Value 를 라이트 합니다. 이 명령은

PIC 마이크로 MCU 에 EEPROM 을 내장한 PIC16F84, 16C84 그리고 16F87x 시리즈에서 사용할 수 있습니다. WRITE 는 동작중에 EEPROM 에 값을 라이트 할 수 있습니다. 디바이스 프로그램시에 EEPROM 에 데이터를 라이트 하려면 DATA 문이나 EEPROM 문을 사용합니다.
WRITE 문은 자체적인 라이트시간을 가지고 있으며 PIC 마이크로 MCU 에서 10mS 가 소요됩니다.
프로그램에서 인터럽트를 사용하고 있다면 WRITE 명령을 실행하기 이전에 동작금지 하여야 합니다. WRITE 명령 실행 후 그리고 다시 인터럽트 가능상태로 설정 합니다. WRITE 명령 실행중에 인터럽트가 발생하면 WRITE 실행은 실패합니다.
WRITE 는 PIC12CE67x 와 16CE62x 디바이스와 같이 I2C 시리얼 인터페이스 EEPROM 은 동작하지 않습니다. 대신에 I2CWRITE 명령을 사용합니다.

```
        WRITE 5,B0 ' Send value in B0 to EEPROM
location 5
```

Word 변수를 라이트 하려면 2 개 바이트를 분리하여 라이트 합니다.

```
w   Var Word
    WRITE 0,w.BYTE0
    WRITE 1,w.BYTE1
```

5.84. WRITECODE

WRITECODE *Address,Value*

코드 영역 Address 위치에 Value 를 라이트 합니다.
일부 PIC16Fxxx 와 PIC18Fxxx 디바이스는 동작중에 프로그램 영역 코드가 라이트 되는것을 허용합니다. 자신이 코드를 수정하여 라이트한느것은 위험한 기술입니다. 64-256 바이트 이상의 비 휘발성 데이터가 가능하게 됩니다. 사용자는 액티브 프로그램 메모리에 오버 라이트하지 않도록 매우 주의하여야 합니다.
리스팅 파일은 프로그램의 어드레스를 확인해 볼수 있습니다.
PIC16Fxxx 는 14 비트 사이즈의 데이터를 코드 영역에 라이트 할수 있습니다. PIC18Fxxx 는 16 비트 사이즈의 Byte 또는 Word 데이터를 코드 영역 Address 0 ~ 65535 에 라이트 할 수 있습니다.
블록 억세스 디바이스인 PIC16F877 과 18F452 는 모든 블록을 한번에 라이트 하여야 합니다. 라이트 블록 사이즈는 PIC 마이크로 MCU 에따라 다릅니다. 개별적 디바이스의 블록사이즈 정보는 Microchips 사의 데이터 쉬트를 참고 합니다.
PIC18Fxxx 시리즈와 같이 WRITECODE 에 의하여 라이트하기 전에 코드 영역을 소거해야 되는 디바이스가 잇습니다. 자세한 정보는 ERASECODE 명령을 참고하시기 바랍니다.

프로그램에서 인터럽트를 사용하고 있다면 WRITECODE 명령을 실행하기 이전에 동작금지 하여야 합니다. WRITECODE 명령 실행 후 그리고 다시 인터럽트 가능상태로 설정 합니다. WRITECODE 명령 실행 중에 인터럽트가 발생하면 WRITECODE 실행은 실패합니다.
WRITECODE 를 사용하려면 PIC 마이크로 MCU 프로그램시 환경설정에서 가능상태로 하여야 합니다.

```
     WRITECODE $100,w ' Send value in W to code space
location $100
```

5.85. XIN

XIN *DataPin,ZeroPin,{Timeout,Label,}*[*Var{,...}*]

X-10 데이터를 수신하여 House Code 와 Key Code 를 Var 에 저장합니다.
XIN 은 X-10 디바이스로부터 정보를 읽습니다. X-10 모듈은 여러 상표로 제조됩니다. 마이크로 컨트롤러와 AC 파워 라인을 연결하여야 합니다. TW-523 은 양방향 X-10 통신을 위하여 X-IN 이 필요합니다. 이 디바이스는 파워라인 인터페이스와 AC 라인과 마이크로 프로세서를 아이솔레이션 합니다. X-10 포맷은 특허이며 이 인터페이스는 라이센스 비용을 지불하여야 합니다.
Data Pin 은 자동적으로 입력상태가 되며 X-10 인터페이스로부터 데이터를 수신합니다. ZeroPin 은 자동적으로 입력상태가 되며 X-10 인터페이스의 제로 크로스 타이밍을 수신합니다. 양쪽핀 모두 5Volts 에 4.7K 저항으로 풀업 하여야 합니다. DataPin 과 ZeroPin 은 0 – 15 의 상수이거나 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.
옵션인 Timeout 과 label 은 일정시간 X-10 데이터가 수신되지 않을 때 프로그램을 계속 진행하기 위하여 사용합니다. Timeout 은 AC 파워라인의 반주기 (약 8.33 mS)로 지정되어 있습니다.
XIN 은 ZeroPin 에서 수신한 AC 파워라인의 각각의 제로 크로싱에서 데이터

를 처리합니다. 만약 ZeroPin 에서 아무런 변화가 없다면 XIN 은 무한히 대기할 것입니다.
Var 가 Word 사이즈 일 때 각각의 House Code 는 Word 의 상위바이트에 저장됩니다. Var 가 Byte 이라면 단지 Key Code 만 저장됩니다.
House Code 는 0 과 15 사이의 숫자이며 X-10 모듈의 A 에서 P 로 설정되는 House Code 에 대응합니다.
Key Code 는 특정 X-10 모듈의 번호이거나 모듈에 의하여 정해지는 기능번호입니다. 일반적인 상황에서 첫번째 커맨드는 X-10 모듈을 지정하는 번호가 송신됩니다. 뒤이어 원하는 기능 번호가 송신됩니다. 일부 기능은 모든 모듈에서 한번 동작하며 따라서 모듈 번호가 필요치 않습니다. 나중의 나오는 예제를 보면 분명히 알게 될 것입니다. Key Code 번호는 모듈번호 1-16 에 대응합니다.
이들 Key Code 번호는 X-10 모듈에서 주고 받는 실제의 번호와 다릅니다. 이 차이는 BS2 에서 Key Code 와 일치합니다. Stamp 변환을 금지하려면 아래와 같이 DEFINE 을 사용합니다.

```
DEFINE    XINXLAT_OFF      1
```

XIN 은 RAM 과 STACK 의 제한으로 12 비트 PIC 마이크로 MCU 에는 사용할 수 없습니다.
XOUT 에서 결선 방법과 함께 기능이 나열됩니다.

```
housekey Var Word
' Get X-10 data
loop: XIN PORTA.2,PORTA.0,[housekey]
' Display X-10 data on LCD
Lcdout $fe,1,"House=",#housekey.byte1,
"Key=",#housekey.byte0
Goto loop ' Do it forever
' Check for X-10 data, go to nodata if none
XIN PORTA.2,PORTA.0,1,nodata,[housekey]
```

5.86. XOUT

XOUT *DataPin,ZeroPin,*[*HouseCode**KeyCode*{*Repeat*}{,...}]

HouseCode 를 보내고 KeyCode 를 보냅니다. 그리고 X-10 포맷의 Repeat 번호를 보냅니다. 선택사양인 Repeat 를생략하면 2 번으로 간주합니다. Repeat 는 일반적으로 Bright 와 Dim 커맨드로 예약됩니다.

XOUT 은 X-10 모듈로 제어정보를 보낼 때 사용합니다. X-10 모듈은 여러 상표로 제조됩니다. 마이크로 컨트롤러와 AC 파워 라인을 연결하여야 합니다. PL-513 은 단 방향이며 TW-523 은 양방향 X-10 통신을 위하여 TW-523 이 필요합니다. 이 디바이스는 파워라인 인터페이스와 AC 라인과 마이크로 프로세서를 아이솔레이션 합니다. X-10 포맷은 특허이며 이 인터페이스는 라이센스 비용을 지불하여야 합니다.

Data Pin 은 자동적으로 X-10 인터페이스로 데이터를보내기 위하여 출력상태가 됩니다. ZeroPin 은 자동적으로 입력상태가 되며 X-10 인터페이스의 제로 크로스 타이밍을 수신합니다. 양쪽 핀 모두 5Volts 에 4.7K 저항으로 풀업 하여야 합니다. DataPin 과 ZeroPin 은 0 – 15 의 상수이거나 0-15 를 저장하는 변수 또는 PORTA.0 과 같은 핀 이름입니다.
.
XOUT 은 ZeroPin 에서 수신한 AC 파워라인의 각각의 제로 크로싱에서 데이터를 처리합니다. 만약 ZeroPin 에서 아무런 변화가 없다면 XIN 은 무한히 대기할 것입니다.

House Code 는 0 과 15 사이의 숫자이며 X-10 모듈의 A 에서 P 로 설정되는 House Code 에 대응합니다.
Key Code 는 특정 X-10 모듈의 번호이거나 모듈에 의하여 정해지는 기능번호입니다. 일반적인 상황에서 첫번째 커맨드는 X-10 모듈을 지정하는 번호가 송신됩니다. 뒤이어 원하는 기능 번호가 송신됩니다. 일부 기능은 모든 모듈에서 한번 동작하며 따라서 모듈 번호가 필요치 않습니다. 나중의 나오는 예제를 보면 분명히 알게 될 것입니다. Key Code 번호 0-15 는 모듈번호 1-16 에 대응합니다.
KeyCode(기능)이름은 MODEFS.BAS 파일에서 정의됩니다. 이를 사용하려면 아래 라인을 프로그램의 맨 처음에 추가 합니다.

```
Include "modedefs.bas"
```

XOUT 에서 결선 방법과 함께 기능이 나열됩니다.
BS1DEFS.BAS 와 BS2DEFS.BAS 는 이미 MODEFS.BAS 에 포함되어 있습니다. 이 파일이 포함되어 있다면 다시 포함시키면 않됩니다. KeyCode 번호는

이 파일을 포함하지 않고 사용할수 있습니다.

KeyCode	KeyCode No.	Operation
uniton	%10010	Turn module on
Unitoff	%11010	Turn module off
Unitsoff	%11100	Turn all modules off
Lighton	%10100	Turn all light modules on
lightsoff	%1000	Turn all light modules off
Bright	%10110	Brighten light module
dim	%1110	Dim light module

이들 Key Code 번호는 X-10 모듈에서 주고 받는 실제의 번호와 다릅니다. 이 차이는 BS2 에서 Key Code 와 일치합니다. Stamp 변환을 금지하려면 아래와 같이 DEFINE 을 사용합니다.

```
DEFINE     XOUTXLAT_OFF      1
```

XOUT 은 RAM 과 STACK 의 제한으로 12 비트 PIC 마이크로 MCU 에는 사용할 수 없습니다.
X-10 인터페이스는 4 개의 연결선이 필요합니다. X-10 인터페이스(제로크로스와 수신데이터)는 오픈 컬렉터이며 4.7K 저항을 5V 에 풀업하여야 합니다. 각 인터페이스의 결선표는 아래와 같습니다.

PL-513 결선

Wire No.	Wire Color	Connection
1	Black	Zero crossing output
2	Red	Zero crossing common
3	Green	X-10 transmit common
4	Yellow	X-10 transmit input

TW-523 결선

Wire No.	Wire Color	Connection
1	Black	Zero crossing output
2	Red	Common
3	Green	X-10 receice output
4	Yellow	X-10 transmit input

```
house Var Byte
unit Var Byte
Include "modedefs.bas"
house = 0 ' Set house to 0 (A)
unit = 8 ' Set unit to 8 (9)
' Turn on unit 8 in house 0
XOUT PORTA.1,PORTA.0,[house\unit,house\unitOn]
' Turn off all the lights in house 0
XOUT PORTA.1,PORTA.0,[house\lightsOff]
' Blink light 0 on and off every 10 seconds
XOUT PORTA.1,PORTA.0,[house\0]
loop: XOUT PORTA.1,PORTA.0,[house\unitOn]
Pause 10000 ' Wait 10 seconds
XOUT PORTA.1,PORTA.0,[house\unitOff]
Pause 10000 ' Wait 10 seconds
Goto loop
```

6. Structure of a Compiled Program

PBP 는 사용하기 쉽도록 설계되었습니다.
프로그램이 컴파일 되면 PBP 의 최소의 지식으로 동작합니다. 어떤 사람은 제품 생산 시 PBP 의 내부 동작을 이해하여야 확신할 수 있다고 합니다. 이 섹센에서 파일의 사용법과 PBP 가 만들어내는 출력 그리고 무었을 할 수 있는지에 관한 아이디어를 줄 것입니다.

6.1. Target Specific Headers

프로그램 컴파일 시 3 개의 타겟(PIC 마이크로 MCU) 지정 헤더 파일을 사용합니다. 하나는 PBP 에서 사용합니다 다른 2 파일은 어셈블러 사용 시 포함합니다.
마이크로 컨트롤러의 이름에 뒤이은 확장자가 *.BAS 인 파일은 PBP 가 칩의 특수정보를 알기 위하여 필요한 파일입니다. 이 내포파일은 칩의 메모리 정보, 라이브러리, 변수 정의등 을 가지고 있습니다. PIC16F84 가 기본 프로세서 입니다. 파일이름은 16F84.BAS 입니다.
마이크로 컨트롤러의 이름에 뒤이은 확장자가 *.INC 인 파일은 만들어진 어셈블리 파일을 만들 정보를 가지고 있습니다.
마지막으로 어셈블리 자신의 내포 파일을 가지고 있습니다. 이 파일은

M16F8x.INC 이며 INC 서브 디렉토리에 있습니다.

6.2. The Library Files

PBP 는 모든 코드와 마이크로 컨트롤러의 그룹별 정의파일인 라이브러리 셋을 가지고 있습니다. 14-비트 코어 PIC 마이크로 MCU 에서 이 파일 이름은 PBPPIC14 입니다.

PBPPIB14.LIB 는 컴파일러에서 사용되는 모든 어셈블리 언어 서브루틴을 포함하고 있습니다. PBPPIC14.MAC 는 이 서브루틴이 호출하는 모든 매크로를 저장하고 있습니다. 대부분의 PicBasic Pro 명령어는 매크로로 구성되며 라이브러리 파일과 관계되어 있습니다.

PBPPIC14.RAM 은 VAR 문장과 관련하여 라이브러리에서 필요한 메모리를 허용합니다.

PIC14EXT.BAS 는 모든 14-비트 PIC 마이크로 MCU 레지스터 이름을 PBP 에게 알려주기 위한 외부 정의를 포함하고 있습니다.

6.3. PBP Generated Code

PICBasic Pro 컴파일러는 여러단계를 거칩니다. 먼저 PBP는 .ASM 파일을 만듭니다. 이 파일에는 마크로만 포함되어 있습니다. 만냑 현단계에서 에러가 없으면 어셈블러를 기동합니다. 어셈블러는 관련된 파일을 만듭니다. 여기에는 .HEX 파일이 포함되어 있습니다. 리스팅 파일과 디버그 파일 출력도 가능합니다.

6.4. .ASM File Structure

.ASM 파일은 매우 특수구조를 가집니다.
파일은 어셈블러에서 사용하는 Equate 정보가 있습니다. 그다음 타겟으로 사용하는 프로세서에 관한 환경 데이터와 같은 기본적인 정보를 INCLUDE 가 따라옵니다.다음으로 모든 변수의 위치 그리고 재지정 정보등이 나열됩니다. 다음으로 EEPROM 이 초기화 됩니다. 마크로 파일을 포함하는 INCLUDE 가 놓여지며 뒤이어 라이브러리 서브루틴 INCLUDE 가 놓여집니다. 마지막 으로 실제의 코드가 배치됩니다.
이 프로그램 코드는 PicBasic Pro 에서 만들어진 마크로 리스트가 됩니다.

7. Other PicBasic Pro Considerations

7.1. How Fast is Fast Enough?

기본적으로 PicBasicPro 컴파일러는 PIC마이크로MCU가 4Mhz 의 크리스탈이나 레조네이터로 동작하는것으로 실행 코드를 만듭니다. 시간과 밀접한 관련이 있는 명령은 딜레이 명령으로 1 uS 인스트럭션을 사용합니다. 이것은 PAUSE 1000 에서 정확한 1 초 그리고 SERIN 과 SEROUT 명령에서 정확한 바우드레이트를 만들어 냅니다. PIC마이크로MCU가 4MHz에 동작하는것이 좋으나 이 외의 주파수로 동작할때 도 있습니다. 또는 시리얼 입력과 출력에서 19200BPS 이상의 속도가 필요할 수 있습니다. PicBasic Pro 프로그램이 4MHz 이외의 주파수로 동작하는것은 두가지 다른 방법이 있습니다. 첫번째로 PBP에 알리지 않고 단순히 4MHz 이외의 오실레이터 주파수를 사용하는것입니다. 이 방법은 시간과 밀접한 명령에 주의를 한다면 매우 유용한 방법입니다 19200BPS의 통신속도를 원하면 마이크로 프로세서의 클록을 4MHz 대신에 8MHz 를 사용하는것입니다. 이렇게 하면 SERIN 과 SEROUT커맨드를 포함하여 모든것이 2 배로 빠르게 동작합니다. SERIN 과 SEROUT에서 9600으로 지정하였다면 크리스탈 스피드가 2 배로 되므로 실제의 바우드 레이트도 19200 바우드 가 됩니다. 그러나 PAUSE 와 SOUND 도 두배가 되는것을 명심하여야 합니다.
PAUSE 1000 명령은 8MHz 크리스탈을 사용할때 0.5 초만 기다리라는 것이 될것입니다. 바람직한 방법은 사용할 크리스탈 주파수를 DEFINE문을 사용하여 PBP 에게 알려주는 것입니다. DEFINE 문은 LCDOUT에서 설명하였습니다.

DEFINE OSC 8

위와 같이 PicBasicPro 프로그램의 시작부에서 포함하면 PBP에게 8MHz 오실레이터로 동작하는것을 알리는것입니다. 사용가능한 오실레이터 주파수 설정과 : OSC 최소딜레이입니다.
3 (3.58) 20us
4 24us
8 12us
10 8us

12 7us
16 5us
20 3us
24 3us
25* 2us
32* 2us
33* 2us
40** 2us
* PIC17Cxxx and PIC18Xxxx only.
** PIC18Xxxx only.
PBP에게 오실레이터 주파수를 알려 보정 하도록 하며 다음 명령이 정확한 타이밍으로 동작하도록 합니다.
COUNT, **DEBUG**, **DEBUGIN**, **DTMFOUT**, **FREQOUT**, **HPWM**, **HSERIN**, **HSEROUT**, **I2CREAD**, **I2CWRITE**, **LCDOUT**, **OWIN**, **OWOUT**, **PAUSE**, **PAUSEUS**, **SERIN**, **SERIN2**, **SEROUT**, **SEROUT2**, **SHIFTIN**, **SHIFTOUT**, **SOUND**, **XIN**, **XOUT**.
오실레이터 주파수를 변경하므로 **PULSEIN, PULSEOUT** 그리고 **RCTIME** 명령의 분해능을 향상할수 있습니다. 4MHz 주파수에서 10uS 의 분해능으로 동작하지만 20MHz 크리스탈을 사용하면 분해능은 5 배 증가하여 2 uS 가 됩니다. 그러나 적절한지 검토하여야 합니다. 펄스폭은 16비트 카운터로 측정하는것이므로 2 uS 의 분해능으로 동작할때 측정가능한 최대 시간은 131,070 uS 가 됩니다.
낮은 크리스탈 주파수인 32.768KHz 를 사용할수도 있습니다. 파워를 줄이는 효과가 잇습니다. 그러마 SERIN과 SEROUT 명령은 사용할수 없을것이며 워치독 타이머는 주기적으로 프로그램을 리스타트 하게 할것입니다. 특정한 용도에 대하여 이 크리스탈 주파수가 사용한지는 실험을 통하여 확인하여야 합니다.

7.2. Configuration Settings

앞서 설명한대로 특정 디바이스에 대한 디폴트 환경 설정은 확장자가 .INC 인 파일이며 파일이름은 16F84.INC 과 같이 디바이스 이름과 같습니다. 이 설정은 디바이스가 프로그램 될때 변경될 수도 있습니다. 대부분의 디바이스에서 오실레이터 주파수는 XT 입니다. 이것은 4MHz 로 디폴트 설정값입니다. 빠른 주파수를 사용한다면 이 설정은 HS 로 변경됩니다.
워치독 타이머 는 PBP에 의하여 동작할수 있습니다. 사용한다면 TMR0 프리스케일러는 NAP과 SLEEP명령에 의하여 사용됩니다.

7.3. RAM Usage

일반적으로 PBP가 마이크로 컨트로러에서 RAM 이 어떻게 배치되는가를 알필요는 없습니다. PBP는 프로그래머가 관여하지 않아도 좋도록 되어 있습니다.
그러나 알고 있으면 유용할대가 있습니다. 변수는 PIC마이크로MCU의 RAM레지스터에 저장됩니다. 첫번째로 가능한 RAM 어드레스는 PIC16F84와 다른 작은 규모의 PIC마이크로MCU에서 $0C입니다. PIC16C74와 다른 대형 PIC마이크로MCU에서 $20입니다. 사용하려는 마이크로 프로세서의 RAM 레지스터 시작번지는 Microchip 의 데이터를 참고합니다. 변수는 RAM 에서 연속적으로 정해진 순서에 의하여 배열됩니다. 순서는 Word 배열형이 먼저이며 그다음 Byte 배열형 그리고 Bit 배열형입니다. 그다음 Word변수, Byte변수 그리고 Bit변수가 배치됩니다. Bit변수는 가능한 Byte단위로 패킹됩니다. 이 순서는 RAM의 최적 상태가 됩니다. (PIC18Xxxx디바이스와 어레이는 나중에 배치됩니다.)
어레이는 대부분의 PIC마이크로MCU 에서 한개의 뱅크에 저장됩니다. 뱅크를 교차하지 배치할 수 없습니다. 이것은 실제적인 어레이 길이의 제한이 됩니다. PBP에게 어레이의 저장 뱅크를 지정할 수 있습니다.
penny VAR WORD **BANK0**
nickel VAR BYTE **BANK1**
뱅크를 지정하면 먼저 처리됩니다. 요청한 뱅크의 여유공간이 없을 때 사용가능한 공간을 Warnning 메시지와 함께 알려줍니다. 변수 선언시 특정 어드레스를 지정할수 있습니다. 대부분의 경우 PBP에게 맞기는것이 좋습니다. 그러나 인터럽트 핸들러에서 W 레지스터에 저장할때 고정된 어드레스를 사용합니다. 뱅크를 선택할와 유사한 방법으로 합니다.
w_store VAR BYTE $20
여러 시스템 변수는 컴파일러의 라이브러리 서브루틴이 사용하기 위하여 자동적으로 24 바이트를 배치합니다. 이 변수는 PBPPIC14.RAM 에 배정되어 있으며 반드시 Bank0(또는 PIC18Xxxx 디바이스에서 Bank A) 입니다.
사용자 변수는 변수앞에 언더스코어(_)가 붙여집니다. 반면에 시스템 사용변수는 언더스코어가 붙지 않습니다. 그러므로 서로 혼동되지 않습니다.
R0 VAR WORD **SYSTEM**
BASCIC Stamp 에서 사용하는 변수 B0 - B25와 W0 - W12는 자동으로

배정되지 않습니다. VAR 를 사용하여 변수 선언을 하여야 합니다. 그러나 이 변수가 만들어져 있기를 원한다면 PicBasic Pro 프로그램 초기에 BS1DEFS.BAS or BS2DEFS.BAS를 내포하도록 하면 됩니다. 이들 변수는 영역이 분리되며 나중에 상요자가 선언하는 다른 변수들과 분리됩니다. 추가적인 임시 변수가 컴파일러에 의하여 자동으로 만들어지며 이 변수의 목록은 전체 메모리 맵과 함께 만들어진 .ASM 또는 .LST 파일에서 볼수 있습니다. 변수를 만들 충분한 RAM 메모리가 부족하다면 변수 배치할수 없음을 알리는 Error 메시지가 출력됩니다.

7.4. Reserved Words

예약어란 간단하게 말해서 컴파일러에서 사용하는 단어이므로 사용자가 변수이름, 라벨로 정의할수 없는 단어 입니다. 이들 예약어는 명령어(names of commands), 의사명령어(pseudo-ops), 변수형(variable types) 또는 PICmicro MCU의 레지스터(registers) 입니다. 의사명령어(pseudo-ops), 변수형(variable types) 그리고 명령어(commands) 키워드는 Appendix C.를 참고합니다. PICmicro MCU 레지스터는 PIC14EXT.BAS 에 정의되어 있습니다. 만약 BS1DEFS.BAS, BS2DEFS.BAS 또는 MODEDEFS.BAS 파일이 포함된다면 내부에서 정의된 정의는 예약어가 되므로 다시 정의할수 없습니다.

7.5. Life After 2K

PicBasic Pro 컴파일러는 2K 이상이 가능합니다. PIC마이크로MCU는 세그멘트 코드 스페이스 구조로 되어 있습니다. PIC마이크로MCU는 14비트 명령 코어에서 Call 과 Goto 명령은 2K 프로그램 영역을 지정하기에 충분한 비트를 갖습니다. 2K 영역을 넘도록 하려면 Call이나 Goto를 사용하기 이전에 PCLATCH레지스터를 사용하여야 합니다. PBP는 자동으로 PCLATCH비트를 설정합니다. 그러나 약간의 제한은 있습니다. PicBasic Pro 라이브러리는 코드 스페이스의 0 페이지에 모두 들어가야 합니다. (12비트 코어에서 페이지 0를 2 개로 나눈것중 첫번째) 일반적으로 PicBasic Pro 프로그램과 라이브러리는 2K 보다 작으므로 논란의 대상이 되지 않습니다. 그러나 추가적인 라이브러리가 사용된다면 주의하여야 합니다. 어셈블리로 작성한 인터럽트 핸들러는 코드 스페이스의 0 페이지 이내에 들어가야 합니다. PicBasic Pro 프로그램을 시작하기 전에 이러한것을 생각하여야 하며 더 자세한 것은 다음 어셈블리 언어 설명에서 참고합니다. PCLATCH비트를 설정하는것은 코드가 추가되는 것입니다. PBP는 어떤 Call 이나 Goto 이전에 12-비트 코어PIC마이크로MCU에서 512 워드, 14-비트 코어에서 2K 이상의 코드 스페이스 그리고 PIC17Cxxx디바이스에서 8K 이상의 코드 스페이스를 넘어가면 PCLATCH를 세트합니다.
PicBasic Pro 명령에는 2K 문제를 해결하는 BRANCHL 이 있습니다. BRANCHL은 PIC18Xxxx디바이스에서 1K 이상 그리고 다른 모든 디바이스에서 페이지 경계를 넘어가는 라벨 위치로 브랜치하도록 합니다. PIC마이크로MCU가 하니의 프로그램 스페이스를 가지고 있다면 BRANCHL 명령에 비하여 작은 코드를 만드는 BRANCH명령을 사용합니다. 마이크로 컨트롤러가 하나의 코드 페이지 이상을 가지고 있으면 BRANCH가 동일한 페이지에서 놓여진다고 확신할수 없을때 BRANCHL을 사용합니다. 페이지 영역이 교차될때 어셈블러는 경고 메시지를 출력합니다.이것은 일반적인 것이며 BRANCH 명령이 페이지를 교차하는가를 확인합니다.

7.6. 12-Bit Core Considerations

일반적으로 12-비트 PIC마이크로MCU의 구조때문에 다른 PIC마이크로MCU 패밀리에 비교하여 많은 컴파일시간이 소요됩니다. 대부분의 경우 다른 디바이스 패밀리를 선택하는것이 바람직 합니다. 그러나 많은 유용한 프로그램이 12-비트 코어 디바이스를 사용할수 있습니다. 변수용 RAM 메모리의 제한과 라이브러리 루틴이 첫번째 256 워드이어랴 하는 2 가지 중요한 제한 사항이 있습니다. 이 러한 제한이 일부 명령어를 사용할수 없으며 다른 동작으로 수정하여야 합니다. 12-비트 PIC마이크로MCU는 RAM의 20 그리고 22 사이의 바이트를 내부 변수로 사용하며 여분의 RAM은 필요한 임시 변수 영역으로 사용합니다. 이 RAM 배치는 4 레벨 소프트웨어 스택을 포함하므로 BASIC프로그램은 4 레벨 깊이의 GOSUB 를 사용할수 있습니다. 일부 PIC마이크로MCU는 24 또는 25바이트의 RAM 을 가지고 있으므로 이들 디바이스는 매우 작은 사용자 변수 영역을 가집니다. 컴파일 과정에서 변수를 배정할수 없다는 에러 메시지가 출력되면 더 많은 RAM을 가진 다른 PIC마이크로MCU를 선택하여야 합니다. 12-비트 PIC마이크로MCU는 첫번째 코드페이지의 처음 절반 영역을 호출할수 있습니다. 그러므로 컴파일러의 라이브러리 루틴은 모두 호출 가능하려면 PIC마이크로MCU 코드영역의 처음 256 영역에 있어야 합니다. I2CREAD 와 같은 많은 라이브러리 루틴은 크기가 큽니다. 약간의 루틴만 처음 256 워드에 들어갈 것입니다. 더 많은

라이브러리를 사용하기 원한다면 12-비트 코어 대신 14-비트 또는 16-비트 코어 PIC마이크로MCU를 사용하여야 합니다.

8. Assembly Language Programming

어셈블리 언어 루틴은 PicBasic Pro 컴파일러 프로그램과 함께 유용하게 사용할수 있습니다. 일반적인 모든 작업은 BASIC에 의하여 완전하게 실행할수 있지만 특별히 빠른 속도가 필요하거나 작은 코드 스페이스가 필요하거나 컴파일러에게 맞기기에 적합치 않은 특별한 작업에이 있을수 있습니다. 이럴때 인-라인 어셈블러의 능력을 이용합니다. 대부분의 프로그램은 PicBasic Pro 프로그램으로 작성하고 약간의 어셈블리 언어를 추가하여 기능성을 증가하는 것이 유용합니다. 이 추가적인 코드는 PBP 프로그램에 직접 삽입하거나 다른파일을 만들어 내포합니다.

8.1. Two Assemblers - No Waiting

PBP 가 컴파일을 시작하면 처음에는 어셈블리언어로 변환합니다. 그리고 자동으로 어셈블러를 기동합니다. 이 변환은 마이크로 프로세서에 프로그래밍 할수 있는 .HEX파일을 만듭니다. 두개의 다른 어셈블러가 PBP에서 사용할 수 있습니다. Pic Macro Assembler인 PM 과 Microchip사의 MPASM 입니다. PM은 컴파일러에 포함되어 있습니다. MPASM은 Microchip사의 웹사이트에서 직접 입수하여야 합니다. 두 어셈블러는 모두 장단점이 있습니다. PM은 PBP에 포함되므로 사용하기 편리합니다. MPASM에 비하여 실행속도가 빠릅니다. DOS모드에서 큰 프로그램의 어셈블이 가능합니다. PM은 8051스타일의 명령어 셋과 Microchip사의 기호보다 편리한것이 있습니다. PICmicro Macro Assembler의 완전한 정보는 PM.TXT를 참고합니다. 반면에 MPASM 은 .COD 파일을 출력합니다. 이 파일은 시뮬레이터나 에뮬레이터에게 매우 유용한 정보를 제공합니다. MPASM은 web과 Microchip사 데이터북에 있는 광범위한 예제를 사용할 수 있습니다. PBP는 디폴트로 PBP를 사용합니다. PBP와 함께 MPASM을 사용하기 원하면 MPASM의 모든 파일을 정해진 서브 디렉토리에 설치합니다. 이 서브 디렉토리는 DOS의 PATH 처리 하여야 합니다. MPASM은 PBP와 함께 두 가지 방법으로 사용할수 있습니다. 명령어 라인 옵션에 "-ampasm" 을 지정하면 컴파일이 끝나면 MPASM을 기동합니다. MPASM은 자신의 윈도우 스크린을 표시합니다.
PBP -ampasm filename

다른 방법으로 명령어 라인 옵션에 "-amp" 는 MPASM 을 quit 모드로 기동하며 에러사항만 표시합니다. However, the launcher consumes additional memory that is therefore not available to MPASM.
PBP -amp filename

최대 메모리 효율을 위하여 Window 버젼 MPASM에서 "-ampasmwin"을 지정합니다. 아무튼 MPASM 은 PBP에 포함되지 않으며 Microchip사로부터 입수하여야 합니다.

8.2. Programming in Assembly Language

PBP 프로그램은 "at" 기호(@)를 어셈블리 라인 앞에 놓아 한 라인의 어셈블리 명령을 사용할 수 있습니다. 또는 여러 라인의 어셈블리 코드를 사용하려면 ASM 키워드 이후에 어셈블리 라인을 놓고 ENDASM 키워드로 종료하면 됩니다. 두 키워드 모두 한개 라인을 점유합니다.
@ bsf PORTA,0
Asm
bsf STATUS,RP0
bcf TRISA,0
bcf STATUS,RP0
Endasm
어셈블리 라인은 어셈블리 출력 파일에 포함됩니다. 이 것은 PM과 PICmicro Macro Assembler 모두 적용됩니다. 그러나 프로그래머는 PBP 라이브러리와 친숙하여야 합니다. PBP의 표기법은 다른 상용화된 컴파일러와 유사하며 특이한 것은 없으므로 인-라인 어셈블리를 충분히 사용할수 있습니다. PBP프로그램에서 정의한 선언자 이름은 언더스코어(_)가 앞에 표시된것 이외에는어셈블리에서 선언된 것과 비슷합니다. 이 것은 어셈블리 프로그램에서 사용자 변수, 상수 그리고 라벨 위치를 구동할 수 있도록 합니다. 그러므로 어셈블리 프로그램에서 언더스코어를 포함하면 PBP에서 만든 기호와 혼동될수 있습니다. 어셈블리 언어에서 사용한 언더스코어 때문에 PBP에서 구동하지는 않습니다. PBP에서 만든 언더스코어 붙은 이름은 컴파일러에서 정의된 실제 정보의 그림자 입니다. 그러므로 인-라인 어셈블러가 출력 파일에 직접 복사되어도 컴파일러에 의하여 구동되지 않습니다. 컴파일러는 어셈블리 프로그램의 어던 유형이나 값 정보도 가지 않을뿐 아니라 존재 한느것 자체를 관여하지 않습니다. 만약 변수를 PBP와 어셈블리 프로그램이 공유하여야 한다면

PBP 에서 반드시 변수 선언 하여야 합니다. 이대 언더 스코어 기호는 혼란될 우려가 있으므로 사용하여선 안됩니다. 문제는 내부 라이브러리인데 다행이 대부분의 라이브러리 식별자는 '?' 또는 하나의 작업 레지스터(R0 와 같은)를 참조하도록 되어 있습니다. 문제를 일으키지 않기 위하여 이러한 이름은 사용하지 않습니다. 만약 이름이 중복되어 충돌하면 어셈블러는 중복 정의 에러를 표시할것입니다. 어셈블리 언어에서 주석문 시작 기호는 PicBasic Pro 의 싱글 따옴표(')에서 세미콜론(;)으로 변경됩니다.

‘ PicBasic Pro comment
; Assembly language comment

8.3. Placement of In-line Assembly

PBP 실행문은 소스 코드에서 나타난 순서대로 실행됩니다. 코드의 구성은 다음과 같습니다. : 0 에서 시작하며 리셋 벡터 입니다. PBP는 약간의 스타트 업 코드를 삽입하고 뒤이어 INIT 로 점프 하는 코드가 놓여집니다. 다음으로 호출한 라이브러리가 배치됩니다. 라이브러리의 마지막에 INIT 가 잇으며 추가적인 초기화 코드가 배치됩니다. 마지막으로 라벨 MAIN 이 잇으며 컴파일된 PicBasic Pro 명령이 추가 됩니다. PicBasic Pro 소스 프로그램의 첫번째 실행 라인이 시작라인이 됩니다.

8.4. Another Assembly Issue

PIC마이크로MCU 레지스터는 뱅크로 구성됩니다. PBP는 이 레지스터 뱅크를 관리합니다. TRIS 레지스터를 지정하면 알수 있습니다. 예를들어 PORT를 구동하기전에 뱅크셀렉트가 필요합니다. Call이나 Goto 이전에 벵크 셀렉트 비트를 0 으로 하여야합니다.

PICmicro MCU registers are banked. PBP keeps track of which register bank it is pointing to at all times. It knows if it is pointing to a TRIS register, for example, it needs to change the bank select bits before it can access a PORT.

It also knows to reset the bank select bits to 0 before making a Call or a Goto. It does this because it can’t know the state of the bank select bits at the new location. So anytime there is a change of locale or a label that can be called or jumped to, the bank select bits are zeroed.

It also resets the bank select bits before each **ASM** and the **@** assembler shortcut. Once again, the assembler routine won’t know the current state of the bits so they are set to a known state. The assembler code must be sure to reset the bank select bits before it exits, if it has altered them.

9. Interrupts

인터럽트는 디버깅이 어렵기는 하지만 프로그램을 유용하게 만듭니다. 인터럽트는 하드웨어 이벤트, 또는 I/O 핀 변화, 타이머의 타임 아웃등 입니다. 인터럽트 가능 상태라면 인터럽트는 무었을 실행하고 있었든지간에 프로세서를 정지하고 인터럽트 서비스 루틴으로 점프합니다. 인터럽트는 PicBasic Pro 프로그램이 다른 일을 하고 있을때 시리얼 포트로부터 데이터를 읽어 버퍼에 저장하도록 할수 있습니다. PicBasic Pro 컴파일러는 인터럽트를 처리하는 2 가지 메커니즘을 가지고 있습니다. 첫번째는 어셈블리 언어에서 인터럽트 처리 프로그램을 작성하는것입니다. 두번째로는 ON INTERRUPT를 PicBasic Pro 프로그램에서 사용하는것입니다.

9.1. Interrupts in General

인터럽트가 발생하면 PIC마이크로MCU는 인터럽트가 발생한 바로 다음 명령의 어드레스를 스택에 저장하고 프로그램 어드레스 4 로 점프합니다. 이것은 먼저 하드웨어 스택에서 여분의 위치가 필요하다는것을 의미합니다. 14-비트 코어 MCU에서 8 단계의 하드웨어 스택 깊이 입니다. PicBasic Pro라이브러리 루틴은 4 스택을 사용합니다. 나머지 4 개는 (PIC17Cxxx는 12 개, PIC18Xxxx는 27개)CALL 과 BASIC GOSUB를 위하영 예약됩니다. 명심해야 하는것은 GOSUB는 3 단계 내포(PIC17Cxxx에서 11, PIC18Xxxx에서 26)가 가능합니다. 리턴 어드레스를 저장하여야 하므로 어셈블리 명령 CALL 을 사용하지 않아야 합니다.

When an interrupt occurs, the PICmicro MCU stores the address of the next instruction it was supposed to execute on the stack and jumps to location 4. The first thing this means is that you need an extra location on the hardware stack, which is only 8 deep on the 14-bit core devices to begin with. The PicBasic Pro library routines can use up to 4 stack locations themselves. The remaining 4 (12 for PIC17Cxxx and 27 for PIC18Xxxx)

are reserved for CALLs and nested BASIC GOSUBs. You must make sure that your GOSUBs are only nested 3 (11 for PIC17Cxxx and 26 for PIC18Xxxx) deep at most with no CALLs within them in order to have a stack location available for the return address. If your interrupt handler uses the stack (by doing a Call or GOSUB itself for example), you'll need to have additional stack space available. Once you have dealt with the stack issues, you need to enable the appropriate interrupts. This usually means setting the INTCON register. Set the necessary enable bits along with Global Interrupt Enable. For example:
INTCON = %10010000
enables the interrupt for RB0/INT. Depending on the actual interrupt desired, you may also need to set one of the PIE registers. Refer to the Microchip PICmicro MCU data books for additional
information on how to use interrupts. They give examples of storing processor context as well as all the necessary information to enable a particular interrupt. This data is invaluable to your success. Finally, select the best technique with which to handle your particular interrupts.

9.2. Interrupts in BASIC

PicBasic pro 프로그램에서 가장 ON INTERRUPT 문을 사용하는 것이 쉽게 인터럽트 핸들러를 작성하는 방법입니다. ON INTERRUPT는 PBP에게 내부 인터럽트 핸들러를 동작시키고 인터럽트가발생하면 프로그래머가 BASIC으로 작성한 인터럽트 핸들러가 동작하도록 합니다. ON INTERRUPT를 사용하면 인터럽트가 발생했을 때 PBP는 플래그를 설정하고 무엇을 하고 있었든지 일단 돌아갑니다. 프로그래머가 작성한 인터럽트 핸들러로 즉시 들어가지 않습니다. PBP 명령은 재 사용형(PBP는 새로운 명령이 시작되기 전에 현재의 명령 실행이 완료되어야 함)이 아니기 때문에 인터럽트가 핸들러가 시작하기 까지는 약간의 지연시간이 필요합니다.
예를들어 PicBasic Pro 프로그램에서 Pause 10000을 실행하기 시작했을 때 인터럽트가 발생했다고 하면 인터럽트 핸들러는 10초 후에나 실행됩니다. 즉 PBP는 인터럽트 신호가 발생하였을 때 플래그만 세트해 놓고 PAUSE 명령을 계속 실행합니다. 만약 시리얼 포트로부터 문자를 수신하여 버퍼에 저장하여야 한다면 많은 문자를 놓치게 될것입니다. 이러한 문제플 피하기 위하여 실행 시간이 큰 명령은 사용하지 않는 것 입니다. 예를들어 PAUSE 10000대신에 Pause1 과 FOR .. NEXT 루프를 사용합니다. 이 방법은 인터럽트 반응이 빠르게 되게 합니다. ON INTERRUPT 보다 더 빠른 인터럽트 반응시간을 원한다면 어셈블리 언어를 사용합니다.

The easiest way to write an interrupt handler is to write it in PicBasic Pro using the ON INTERRUPT statement. ON INTERRUPT tells PBP to activate its internal interrupt handling and to jump to your BASIC interrupt handler as soon as it can after receiving an interrupt. Which brings us the first issue. Using ON INTERRUPT, when an interrupt occurs PBP simply flags the
event and immediately goes back to what it was doing. It does not immediately vector to your interrupt handler. Since PBP statements are not re-entrant (PBP must finish the statement that is being executed before it can begin a new one) there could be considerable delay (latency) before the interrupt is handled. As an example, lets say that the PicBasic Pro program just started execution of a Pause 10000 when an interrupt occurs. PBP will flag the interrupt and continue with the PAUSE. It could be up to 10 seconds later before the interrupt handler is executed. If it is buffering characters from a serial port, many characters will be missed. To minimize the problem, use only statements that don't take very long to execute. For example, instead of Pause 10000, use Pause 1 in a long FOR..NEXT loop. This will allow PBP to complete each statement more quickly and handle any pending interrupts. If interrupt processing needs to occur more quicky than can be provided by ON INTERRUPT, interrupts in assembly language should be used. Exactly what happens when ON INTERRUPT is used is this: A short interrupt handler is placed at location 4 in the PICmicro MCU. This interrupt handler is simply a Return. What this does is send the program back to what it was doing before the interrupt occurred. It doesn't require any processor context saving. What it doesn't do is reenable Global Interrupts as happens using an Retfie. A Call to a short subroutine is placed before each statement in the PicBasic Pro program once an ON INTERRUPT is encountered. This short subroutine checks the state of

the Global Interrupt Enable bit. If it is off, an interrupt is pending so it vectors to the users interrupt handler. If it is still set, the program continues with the next BASIC statement, after which, the GIE bit is checked again, and so forth. When the **RESUME** statement is encountered at the end of the BASIC interrupt handler, it sets the GIE bit to re-enable interrupts and returns to where the program was before the interrupt occurred. If **RESUME** is given a label to jump to, execution will continue at that location instead. All previous return addresses will be lost in this case. **DISABLE** stops PBP from inserting the Call to the interrupt checker after each statement. This allows sections of code to execute without the
possibility of being interrupted. **ENABLE** allows the insertion to continue. A **DISABLE** should be placed before the interrupt handler so that it will not keep getting restarted by checking the GIE bit. If it is desired to turn off interrupts for some reason after **ON INTERRUPT** is encountered, you must not turn off the GIE bit. Turning off this bit tells PBP an interrupt has happened and it will execute the interrupt handler forever. Instead set:
INTCON = $80
This disables all the individual interrupts but leaves the Global Interrupt Enable bit set.

9.3. Interrupts in Assembler

Interrupts in assembly language are a little trickier. Since you have no idea of what the processor was doing when it was interrupted, you have no idea of the state of the W register, the STATUS flags, PCLATH or even what register page you are pointing to. If you need to alter any of these, and you probably will, you must save the current values so that you can restore them before allowing the processor to go back to what it was doing before it was so rudely interrupted. This is called saving and restoring the processor context. If the processor context, upon return from the interrupt, is not left exactly the way you found it, all kinds of subtle bugs and even major system crashes can and will occur. This of course means that you cannot even safely use the compiler's internal variables for storing the processor context. You cannot tell which variables are in use by the library routines at any given time. You should create variables in the PicBasic Pro program for the express purpose of saving W, the STATUS register and any other register that may need to be altered by the interrupt handler. These variables should not be otherwise used in the BASIC program. While it seems a simple matter to save W in any RAM register, it is actually somewhat more complicated. The problem occurs in that you have no way of knowing what register bank you are pointing to when the interrupt happens. If you have reserved a location in Bank0 and the current register pointers are set to Bank1, for example, you could overwrite an unintended location. Therefore you must reserve a RAM register location in each bank of the device at the same offset. As an example, let's choose the 16C74(A). It has 2 banks of RAM registers starting at $20 and $A0 respectively. To be safe, we need to reserve the same location in each bank. In this case we will choose the first location in each bank. A special construct has been added to the **VAR** command to allow this:

```
wsave var byte $20 system
wsave1 var byte $a0 system
```

This instructs the compiler to place the variable at a particular location in RAM. In this manner, if the save of W "punches through" to another bank, it will not corrupt other data.
The interrupt routine should be as short and fast as you can possibly make it. If it takes too long to execute, the Watchdog Timer could timeout and really make a mess of things. The routine should end with an Retfie instruction to return from the interrupt and allow the processor to pick up where it left off in your PicBasic Pro program.
A good place to put the assembly language interrupt handler is at the very beginning of your PicBasic Pro program. A **GOTO** needs to be inserted before it to make sure it won't be executed when the program starts. See the example below for a demonstration of this. If a 14-bit core PICmicro MCU has more than 2K of code space, an interrupt stub is automatically added that saves the W, STATUS and PCLATH registers into the variables wsave, ssave and psave, before going to your interrupt handler. Storage for these variables must be allocated in the BASIC program:

```
wsave var byte $20 system
wsave1 var byte $a0 system ‘   If device has
RAM in bank1
wsave2 var byte $120 system ‘   If device has
RAM in bank2
wsave3 var byte $1a0 system ‘   If device has
RAM in bank3
ssave var byte bank0 system
psave var byte bank0 system
```

In any case, you must restore these registers at the end of your assembler interrupt handler. If the 14-bit core PICmicro MCU has 2K or less of code space, or it is an PIC18Xxxx device, the registers are not saved. Your interrupt handler must save and restore any used registers. Finally, you need to tell PBP that you are using an assembly language interrupt handler and where to find it. This is accomplished with a **DEFINE**:

DEFINE **INTHAND** *Label*

Label is the beginning of your interrupt routine. PBP will place a jump to this *Label* at location 4 in the PICmicro MCU.

```
' Assembly language interrupt example
led var PORTB.1
wsave var byte $20 system
ssave var byte bank0 system
psave var byte bank0 system
Goto start ' Skip around interrupt handler
‘   Define interrupt handler
define INTHAND myint
‘   Assembly language interrupt handler
asm
; Save W, STATUS and PCLATH registers
myint movwf wsave
swapf STATUS, W
clrf STATUS
movwf ssave
movf PCLATH, W
movwf psave
; Insert interrupt code here
; Save and restore FSR if used
bsf _led ; Turn on LED (for example)
; Restore PCLATH, STATUS and W registers
movf psave, W
```



```
movwf PCLATH
swapf ssave, W
movwf STATUS
swapf wsave, F
swapf wsave, W
retfie
endasm
‘   PicBasic Pro program starts here
start: Low led ‘   Turn LED off
‘   Enable interrupt on PORTB.0
INTCON = %10010000
loop: Goto loop ‘   Wait here till interrupted
```

Appendix A : Serin2/Serout2 Mode Examples

Baud Rate	Bit 15 (Output)	Bit 14 (Conversion)	Bit 13 (Parity)	Mode Number
300	Driven	True	None	3313
300	Driven	True	Even*	11505
300	Driven	Inverted	None	19697
300	Driven	Inverted	Even*	27889
300	Open	True	None	36081
300	Open	True	Even*	44273
300	Open	Inverted	None	52465
300	Open	Inverted	Even*	60657
600	Driven	True	None	1646
600	Driven	True	Even*	9838
600	Driven	Inverted	None	18030
600	Driven	Inverted	Even*	26222
600	Open	True	None	34414
600	Open	True	Even*	42606
600	Open	Inverted	None	50798
600	Open	Inverted	Even*	58990
1200	Driven	True	None	813
1200	Driven	True	Even*	9005
1200	Driven	Inverted	None	17197
1200	Driven	Inverted	Even*	25389
1200	Open	True	None	33581
1200	Open	True	Even*	41773
1200	Open	Inverted	None	49965
1200	Open	Inverted	Even*	58157
2400	Driven	True	None	396
2400	Driven	True	Even*	8588
2400	Driven	Inverted	None	16780
2400	Driven	Inverted	Even*	24972
2400	Open	True	None	33164
2400	Open	True	Even*	41356
2400	Open	Inverted	None	49548
2400	Open	Inverted	Even*	57740
4800	Driven	True	None	188
4800	Driven	True	Even*	8380
4800	Driven	Inverted	None	16572
4800	Driven	Inverted	Even*	24764
4800	Open	True	None	32956
4800	Open	True	Even*	41148
4800	Open	Inverted	None	49340
4800	Open	Inverted	Even*	57532
9600 baud may be unreliable with 4MHz clock				
9600	Driven	True	None	84
9600	Driven	True	Even*	8276
9600	Driven	Inverted	None	16468
9600	Driven	Inverted	Even*	24660
9600	Open	True	None	32852
9600	Open	True	Even*	41044
9600	Open	Inverted	None	49236
9600	Open	Inverted	Even*	57428
baud rates below require 8MHz clock or faster				
14400	Driven	True	None	49
14400	Driven	True	Even*	8241

14400	Driven	Inverted	None	16433
14400	Driven	Inverted	Even*	24625
14400	Open	True	None	32817
14400	Open	True	Even*	41009
14400	Open	Inverted	None	49201
14400	Open	Inverted	Even*	57393
baud rates below require 10MHz clock or faster				
19200	Driven	True	None	32
19200	Driven	True	Even*	8224
19200	Driven	Inverted	None	16416
19200	Driven	Inverted	Even*	24608
19200	Open	True	None	32800
19200	Open	True	Even*	40992
19200	Open	Inverted	None	49184
19200	Open	Inverted	Even*	57376
baud rates below require 16MHz clock or faster				
28800	Driven	True	None	15
28800	Driven	True	Even*	8207
28800	Driven	Inverted	None	16399
28800	Driven	Inverted	Even*	24591
28800	Open	True	None	32783
28800	Open	True	Even*	40975
28800	Open	Inverted	None	49167
28800	Open	Inverted	Even	57359
33600	Driven	True	None	10
33600	Driven	True	Even*	8202
33600	Driven	Inverted	None	16394
33600	Driven	Inverted	Even*	24586
33600	Open	True	None	32778
33600	Open	True	Even*	40970
33600	Open	Inverted	None	49162
33600	Open	Inverted	Even	57354
baud rates below require 20MHz clock or faster				
38400	Driven	True	None	6
38400	Driven	True	Even*	8198
38400	Driven	Inverted	None	16390
38400	Driven	Inverted	Even*	24582
38400	Open	True	None	32774
38400	Open	True	Even*	40966
38400	Open	Inverted	None	49158
38400	Open	Inverted	Even	57350

Parity is odd when DEFINE SER2_ODD 1 is used.

Appendix B : Defines

```
DEFINE ADC_BITS 8 'Number of bits in ADCIN result
DEFINE ADC_CLOCK 3 'ADC clock source (rc = 3)
DEFINE ADC_SAMPLEUS 50 'ADC sampling time in microseconds
DEFINE BUTTON_PAUSE 10 'Button debounce delay in ms
DEFINE CCP1_REG PORTC 'Hpwm 1 pin port
DEFINE CCP1_BIT 2 'Hpwm 1 pin bit
DEFINE CCP2_REG PORTC 'Hpwm 2 pin port
DEFINE CCP2_BIT 1 'Hpwm 2 pin bit
DEFINE CHAR_PACING 1000 'Serout character pacing in us
DEFINE DEBUG_REG PORTB 'Debug pin port
DEFINE DEBUG_BIT 0 'Debug pin bit
DEFINE DEBUG_BAUD 2400 'Debug baud rate
DEFINE DEBUG_MODE 1 'Debug mode: 0 = True, 1 = Inverted
DEFINE DEBUG_PACING 1000 'Debug character pacing in us
DEFINE DEBUGIN_REG PORTB 'Debugin pin port
DEFINE DEBUGIN_BIT 0 'Debugin pin bit
DEFINE DEBUGIN_MODE 1 'Debugin mode: 0 = True, 1 = Inverted
DEFINE HPWM2_TMR 1 'Hpwm 2 timer select
DEFINE HPWM3_TMR 1 'Hpwm 3 timer select
DEFINE HSER_BAUD 2400 'Hser baud rate
DEFINE HSER_CLROERR 1 'Hser clear overflow automatically
DEFINE HSER_SPBRG 25 'Hser spbrg init
DEFINE HSER_RCSTA 90h 'Hser receive status init
DEFINE HSER_TXSTA 20h 'Hser transmit status init
DEFINE HSER_EVEN 1 'Use only if even parity desired
DEFINE HSER_ODD 1 'Use only if odd parity desired
DEFINE I2C_HOLD 1 'Pause I2C transmission while clock held low
DEFINE I2C_INTERNAL 1 'Use for internal EEPROM on 16CExxx and 12CExxx
DEFINE I2C_SCLOUT 1 'Set serial clock to bipolar instead of open-collector
DEFINE I2C_SLOW 1 'Use for >8MHz OSC with standard speed devices
DEFINE I2C_SDA PORTA,0 'Data pin for I2C (12-bit core only)
DEFINE I2C_SCL PORTA,1 'Clock pin for I2C (12-bit
```

		core only)
DEFINE LCD_DREG	PORTA	'LCD data port
DEFINE LCD_DBIT	0	'LCD data starting bit 0 or 4
DEFINE LCD_RSREG	PORTA	'LCD register select port
DEFINE LCD_RSBIT	4	'LCD register select bit
DEFINE LCD_EREG	PORTB	'LCD enable port
DEFINE LCD_EBIT	3	'LCD enable bit
DEFINE LCD_RWREG	PORTE	'LCD read/write port
DEFINE LCD_RWBIT	2	'LCD read/write bit
DEFINE LCD_BITS	4	'LCD bus size 4 or 8
DEFINE LCD_LINES	2	'Number lines on LCD
DEFINE LCD_COMMANDUS	2000	'Command delay time in us
DEFINE LCD_DATAUS	50	'Data delay time in us
DEFINE LOADER_USED	1	'meLoader Used
DEFINE NO_CLRWDT	1	'Forces manual use of CLRWDT
DEFINE ONINT_USED	1	'Serves as LOADER_USED for versions fore 2.33
DEFINE PULSIN_MAX	1000	'Maximum counts allowed before pulsin times out
DEFINE OSC	4	'Oscillator speed in MHz: 3(3.58) 4 8 10 12 16 20 24 25 32 33 40
DEFINE OSCCAL_1K	1	'Set OSCCAL for PIC12C671/CE673
DEFINE OSCCAL_2K	1	'Set OSCCAL for PIC12C672/CE674
DEFINE SER2_BITS	8	'Set number of data bits for Serin2 and Serout2
DEFINE SER2_ODD	1	'Use odd parity instead of even parity
DEFINE SHIFT_PAUSEUS	50	'Slow down the Shiftin and Shiftout clock
DEFINE USE_LFSR	1	'Use 18Cxxx LFSR instruction
DEFINE XINXLAT_OFF	1	'Don't translate Xin commands to BS2 format
DEFINE XOUTXLAT_OFF	1	'Don't translate Xout commands to BS2 format

Appendix C : Reserved Words

ABS	ADCIN	AND
ANDNOT	ASM	AUXIO
BANK0	BANK1	BANK2
BANK3	BANK4	BANK5
BANK6	BANK7	BANK8
BANK9	BANK10	BANK11
BANK12	BANK13	BANK14
BANK15	BANKA	BIN
BIN1	BIN2	BIN3
BIN4	BIN5	BIN6
BIN7	BIN8	BIN9
BIN10	BIN11	BIN12
BIN13	BIN14	BIN15
BIN16	BIT	BRANCH
BRANCHL	BUTTON	BYTE
CALL	CASE	CLEAR
CLEARWDT	CON	COS
COUNT	DATA	DCD
DEBUG	DEBUGIN	DEC
DEC1	DEC2	DEC3
DEC4	DEC5	DEFINE
DIG	DISABLE	DIV32
DTMFOUT	EEPROM	ELSE
ENABLE	END	ENDASM
ENDIF	EXT	FOR
FREQOUT	GET	GOSUB
GOTO	HEX	HEX1
HEX2	HEX3	HEX4
HEX5	HIGH	HPWM
HSERIN	HSEROUT	I2CREAD
I2CWRITE	IBIN	IBIN1
IBIN2	IBIN3	IBIN4
IBIN5	IBIN6	IBIN7
IBIN8	IBIN9	IBIN10
IBIN11	IBIN12	IBIN13
IBIN14	IBIN15	IBIN16
IDEC	IDEC1	IDEC2
IDEC3	IDEC4	IDEC5
IF	IHEX	IHEX1
IHEX2	IHEX3	IHEX4
IHEX5	INCLUDE	INPUT
INTERRUPT	IS	ISBIN
ISBIN1	ISBIN2	ISBIN3

ISBIN4	ISBIN5	ISBIN6	READ	READCODE	REM
ISBIN7	ISBIN8	ISBIN9	REP	RESUME	RETURN
ISBIN10	ISBIN11	ISBIN12	REV	REVERSE	SBIN
ISBIN13	ISBIN14	ISBIN15	SBIN1	SBIN2	SBIN3
ISBIN16	ISDEC	ISDEC1	SBIN4	SBIN5	SBIN6
ISDEC2	ISDEC3	ISDEC4	SBIN7	SBIN8	SBIN9
ISDEC5	ISHEX	ISHEX1	SBIN10	SBIN11	SBIN12
ISHEX2	ISHEX3	ISHEX4	SBIN13	SBIN14	SBIN15
ISHEX5	LCDIN	LCDOUT	SBIN16	SDEC	SDEC1
LET	LIBRARY	LOOKDOWN	SDEC2	SDEC3	SDEC4
LOOKDOWN2	LOOKUP	LOOKUP2	SDEC5	SELECT	SERIN
LOW	MAX	MIN	SERIN2	SEROUT	SEROUT2
MOD	NAP	NCD	SHEX	SHEX1	SHEX2
NEXT	NOT	OFF	SHEX3	SHEX4	SHEX5
ON	OR	ORNOT	SHIFTIN	SHIFTOUT	SIN
OUTPUT	OWIN	OWOUT	SKIP	SLEEP	SOUND
PAUSE	PAUSEUS	PEEK	SQR	STEP	STOP
PEEKCODE	POKE	POKECODE	STR	SWAP	SYMBOL
POLLIN	POLLMODE	POLLOUT	SYSTEM	THEN	TO
POLLRUN	POLLWAIT	POT	TOGGLE	USBIN	USBINIT
PULSIN	PULSOUT	PUT	USBOUT	VAR	WAIT
PWM	RANDOM	RCTIME	WAITSTR	WEND	WHILE

WORD WRITE WRITECODE

XIN XOR XORNOT

XOUT

Appendix D :ASCII Table

ASCII Control Characters

Decimal	Hex	ASCII Function	Key
0	0	NUL (null)	Ctrl-@
1	1	SOH (start of heading)	Ctrl-A
2	2	STX (start of text)	Ctrl-B
3	3	ETX (end of text)	Ctrl-C
4	4	EOT (end of transmission)	Ctrl-D
5	5	ENQ (enquiry)	Ctrl-E
6	6	ACK (acknowledge)	Ctrl-F
7	7	BEL (bell)	Ctrl-G
8	8	BS (backspace)	Ctrl-H

Standard ASCII Character Set

Decimal	Hex	Display / Key
32	20	Space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)

Decimal	Hex	Display / Key
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I

Decimal	Hex	Display / Key
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i

41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	₩
93	5D	]
94	5E	^
95	5F	_

105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

Appendix E : Summary of Microchip Assembly Instruction Set

ADDLW	k	IORLW	k
ADDWF	f,d	IORWF	f,d
ANDLW	k	MOVF	f,d
ANDWF	f,d	MOVLW	k
BCF	f,b	MOVWF	f
BSF	f,b	NOP	
BTFSC	f,b	RETFIE	
BTFSS	f,b	RETLW	k
CALL	k	RETURN	
CLRF	f	RLF	f,d
CLRW		RRF	f,d
CLRWDT		SLEEP	
COMF	f,d	SUBLW	k
DECF	f,d	SUBWF	f,d
DECFSZ	f,d	SWAPF	f,d
GOTO	k	XORLW	k
INCF	f,d	XORWF	f,d
INCFSZ	f,d		

b - bit address

d - destination; 0 = w, 1 = f

f - register file address

k - literal constant

Appendix F

Contact Information

Technical support and sales may be reached at:

microEngineering Labs, Inc.
Box 60039
Colorado Springs CO 80960
(719) 520-5323
(719) 520-1867 fax
http://www.melabs.com
email:support@melabs.com

PICmicro data sheets, CD-ROMs and literature may be obtained from:

Microchip Technology Inc.
2355 W. Chandler Blvd.
Chandler AZ 85224-6199
(602) 786-7200
(602) 899-9210 fax
http://www.microchip.com
email: literature@microchip.com

READ THE FOLLOWING TERMS AND CONDITIONS CAREFULLY BEFORE OPENING THIS PACKAGE.

microEngineering Labs, Inc. ("the Company") is willing to license the enclosed software to the purchaser of the software ("Licensee") only on the condition that Licensee accepts all of the terms and conditions set forth below. By opening this sealed package, Licensee is agreeing to
be bound by these terms and conditions.

Disclaimer of Liability

THE COMPANY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE AND THE IMPLIED WARRANTY OF MERCHANTABILITY. IN NO EVENT SHALL THE COMPANY OR ITS EMPLOYEES, AGENTS, SUPPLIERS OR CONTRACTORS BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR IN CONNECTION WITH LICENSE GRANTED UNDER THIS AGREEMENT, INCLUDING WITHOUT LIMITATION, LOST PROFITS, DOWNTIME, GOODWILL, DAMAGE TO OR REPLACEMENT OF EQUIPMENT OR PROPERTY, OR ANY COSTS FOR RECOVERING, REPROGRAMMING OR REPRODUCING ANY DATA USED WITH THE COMPANY'S PRODUCTS.

Software License

In consideration of Licensee's payment of the license fee, which is part of the price Licensee paid for this product, and Licensee's agreement to abide by the terms and conditions on this page, the Company grants Licensee a nonexclusive right to use and display the copy of the enclosed software on a single computer at a single location. Licensee owns only the enclosed disk on which the software is recorded or fixed, and the Company retains all right, title and ownership (including the copyright) to the software recorded on the original disk copy and all subsequent copies of the software. Licensee may not network the software or otherwise use it on more than one computer terminal at the
same time. Copies may only be made for archival or backup purposes. The enclosed software is licensed only to the Licensee and may not be transferred to anyone else, nor may copies be given to anyone else. Any violation of the terms and conditions of this software license shall result in the immediate termination of the license.

MicroCode Studio ICD

OVERVIEW

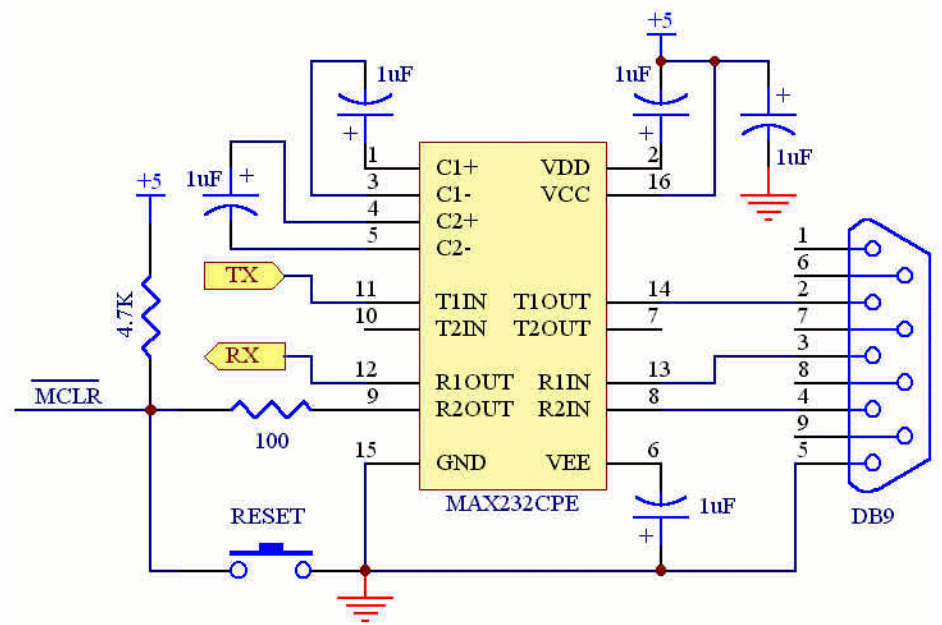
Micro Code Studio ICD(In Circuit Debugger)는 PICBasic Pro 프로그램을 PC 에서 실행하면서 PIC 마이크로 컨트롤러의 변수값, 특수기능 레지스터 (SFR), 메모리 그리고 EEPROM 의 내용을 호스트 PC 에서 관찰할수 있도록 합니다. 여러개의 브레이크 포인트를 설정할수 있으며 PICBASIC Pro 프로그램을 1 라인씩 스텝 실행이 가능합니다. MicroEngineering Labs 의 디버그 구조에 그래픽 기능이 추가된 것으므로 효율적이며 강력한 디버그 환경이 구성됩니다.

SYSTEM REQUIREMENT / 시스템 요구사항

ICD 를 구동하려면 MicroCode Studio 를 설치하여야 합니다. 그리고 microEngineering 의 PICBasic Pro 컴파일러 2.20 이나 그 이후 버젼이 설치되어 있어야 합니다. Micro Code Studio 는 PIC Basic Standard 컴파일러는 사용할 수 없습니다.

Hardware Setup / 하드웨어 설정

MicroCode Studio ICD 는 호스트 PC 와 마이크로컨트롤러에게 특별한 하드웨어 장치를 요구하지 않습니다.



MicroCode Studio ICD 는 사용자의 하드웨어에 USART 와 통신합니다. USART 를 구성할때 Microchip 데이터를 확인하여야 합니다. 예를들어 PIC16F628 은 PORTB.1 을 RX 로 PORTB.2 를 TX 로 사용합니다. PIC16F877 은 PORTC.7 을 RX 로 PORTC.6 을 TX 로 사용합니다.

ICD 는 여러가지의 클록 스피드로 동작합니다. 현재 지원되는 스피드는 4, 8, 10, 12, 16, 20, 24, 25, 32, 33 그리고 40 MHz 입니다. 그러나 PIC Basic Pro 소스 프로그램에서 클록 속도를 정의하여야 합니다. 디폴트로 4MHz 가 사용됩니다. 예를들어 마이크로 컨트롤러가 20 MHz 로 동작한다면 아래의 프로그램을 맨 처음에 삽입합니다.

```
DEFINE OSC 20
```

정확한 오실레이터 주파수 설정에 실패하면 ICD 는 동작하지 않습니다.

MicroCode Studio Plus

MicroCode Studio Plus 의 ICD 는 다음의 PIC 마이크로 컨트롤러 디바이스를 지원합니다.

16F627(A), 16F628(A), 16F73, 16F74, 16F76, 16F77, 16F870, 16F871, 16F873(A), 16F874(A), 16F876(A), 16F877(A), 18F242, 18F248, 18F252, 18F258, 18F442, 18F448, 18F452, 18F458, 18F1220, 18F1320, 18F2220, 18F2320, 18F4220, 18F4320, 18F6620, 18F6720*, 18F8620 ,18F8720*

* ICD 는 64K 블록에서만 동작합니다. 128K 모두 가능하게 하려면 PICBASIC Pro 에 포함된 파일 'pbppic18.mac'을 약간 수정하여야 합니다.

Reducing the ICD Code Overhead

ICD 컴파일이나 ICD 컴파일후 프로그램 버튼을 이용하여 프로젝트를 구성할 때 특별한 헤더가 빌드에 포함됩니다. 이 추가적인 코드는 마이크로 컨트롤러와 Micro Code Studio 의 통신을 가능하게 합니다. ICD 탐색기와 주 편집 화면을 새롭게 갱신하기 위하여 특수한 명령이 PIC Basic 명령의 앞에 놓입니다. 그 결과 최종 HEX 파일의 크기가 커지게 됩니다. 파일 크기는 프로세서의 종류와 사용자의 소스 코드 크기에따라 다릅니다. 최소로 추가되는 코드는 150 –200 워드정도 입니다. 이것은 ICD 헤더 코드 입니다. 즉 아무것도 없는 파일을 ICD 빌드 하였다면 150 – 200 워드가 사용될것입니다. 이 ICD 헤더 코드 입니다. 추가로 8 바이트의 데이터 메모리가 사용되며 Bank 0 이 됩니다.

다행히 일정량의 디버그 정보가 PIC BASIC 에서 DISABLE DEBUG 과 ENABLE DEBUG 지시자를 사용하여 추가되는 것을 제어할수 있습니다. 예를들어 sort.bas 프로그램에서 다음과 같이 추가합니다.

```
DISABLE DEBUG
   FOR Index = 0 TO ArraySize - 1
      RANDOM RandomValue
      Array(Index) = RandomValue
   NEXT Index
ENABLE DEBUG
```

이렇게 하면 DISABLE DEBUG 와 ENABLE DEBUG 사이에는 디버그 코드가 포함되지 않으므로 빌드 사이즈가 작아집니다. 유용한 팁으로 프로그램의

시작위치에 DISABLE DEBUG 를 두는것입니다. 그러면 ENABLE DEBUG 와 DISABLE DEBUG 가 함께 ICD 를 제어하기 원하는 코드를 둘러 싸게 합니다. 이것은 큰 프로그램에서 코드 추가를 줄여줍니다. 예를들어 이 방법으로 sort.bas 의 전 리스트를 보기로 합니다.

```
 ' *** disable debug information ***
DISABLE DEBUG
DEFINE OSC 10

' constants
ArraySize CON 10
True CON 1
False CON 0

' variables
Array VAR WORD(ArraySize)
Index VAR BYTE
InnerIndex VAR BYTE
OuterIndex VAR BYTE
ListSorted VAR BYTE
RandomValue VAR WORD
TmpValue VAR WORD

ProgramStart:

' *** enable debug information for code block ***
ENABLE DEBUG
   FOR Index = 0 TO ArraySize - 1
      RANDOM RandomValue
      Array(Index) = RandomValue
   NEXT Index
DISABLE DEBUG
' *** debug information now disabled ***

' sort the array - simple bubble sort
ListSorted = False
OuterIndex = 0

WHILE NOT ListSorted AND (OuterIndex < ArraySize)
   ListSorted = True

   ' check for exchange
   FOR InnerIndex = 0 TO ArraySize - 2
      Index = InnerIndex + 1
      IF Array(InnerIndex) < Array(Index) THEN
         TmpValue = Array(InnerIndex)
         Array(InnerIndex) = Array(Index)
         Array(Index) = TmpValue
         ListSorted = False
      ENDIF
   NEXT InnerIndex
   OuterIndex = OuterIndex + 1
WEND

' repeat forever
GOTO ProgramStart
```

이 예에서 ICD 는 어레이가 초기화 되는동안 ICD 가 동작할 것입니다. DISABLE DEBUG 와 ENABLE DEBUG 는 컴파일러 지시어임을 기억하여야 합니다. 이것은 소스 프로그램에서 제거할 수 있습니다.

Increasing ICD Execution Speed

ICD 컴파일이나 ICD 컴파일후 프로그램 버튼을 이용하여 프로젝트를 구성할 때 특별한 헤더가 빌드에 포함됩니다(ASM 파일을 시험하려하지 않는한 정상적인 상황에서는 원치 않을것입니다.). 이 추가적인 코드는 마이크로 컨트롤러와 Micro Code Studio 의 통신을 가능하게 합니다. ICD 탐색기와 주 편집화면을 새롭게 갱신하기 위하여 특수한 명령이 PIC Basic 명령의 앞에 놓입니다. ICD 는 각각의개별적인 PICBASIC 명령의 실행속도를 낮추지 않습니다. PULSEIN 이나 PAUSE 과 같은 PIC BASIC 명령을 사용하여도 프로젝트를 ICD 컴파일러를 사용하든지 그렇지 않든지 관계없이 동일한 속도로 동작할 것입니다. 그러나 중복 실행 속도응 여러가지 경우의 수가 있습니다. ICD 는

타겟 디바이스의 오실레이터 설정과 호스트 PC 의 속도에 따라 그래픽 시뮬레이션 동작속도가 다를 것입니다. OSC 주파수를 높게 하므로 더 빠른 속도가 가능합니다. 이상적인 경우에 초당 10 회의 명령 실행 속도를 기대할 수 있습니다. 애니매이션 동작을 정지하면 초당 수백회의 PICBasic 명령이 실행됩니다.
어떤 경우에 PICBasic 프로그램에서 ICD 의 간섭을 원치 않을 것입니다. 예를들면 정밀한 시간발생과 같은 까다로운 코드 블록입니다. ICD 를 사용할 때 그러한 까다로운 부분은 기능 정상화로부터 코드를 보호합니다. 이렇게 하기 위하여 PICBasic 의 DISABLE DEBUG 와 ENABLE DEBUG 지시어를 사용합니다. 예를들어 sort.bas 프로그램에서 다음과 같이 추가합니다.

```
DISABLE DEBUG
      FOR Index = 0 TO ArraySize - 1
            RANDOM RandomValue
           Array(Index) = RandomValue
        NEXT Index
     ENABLE DEBUG
```

위의 예에서 DISABLE DEBUG 와 ENABLE DEBUG 지시어 사이에는 디버깅 코드가 들어 가지 않습니다.
이 부분은 풀스피드로 동작합니다. 프로그램이 컨트롤 블록을 빠져나가면 ICD 는 타겟 프로세서의 제어권을 다시 갖습니다. 이 방법으로 ICD Compile 이나 ICD Compile and Program 을 이용하여 프로젝트를 만들 때 코드를 줄일 수 있습니다. 더 자세한 것은 Reducing the ICD Code Overhead 를 참조합니다.

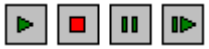
The ICD Toolbar
인서킷 디버거(ICD) 툴바는 적절한 ICD 모델이 가능할 때 활성화 되며 선택한 마이크로 프로세서에 따릅니다.
ICD Compile

 ICD 컴파일 버튼은 MicroCode Studio 컴파일 툴바 버튼과 정확히 같은 방법으로 동작합니다. 그러나 추가적인 디버그 정보가 빌드에 자동으로 추가됩니다. ICD 컴파일은 헥사 파일을 출력하며 마이크로 프로세서에 프로그램합니다. 다른 방법으로 F11 을 누르면 자동적으로 선택한 프로그래머가 컴파일이 완료되면 자동적으로 기동됩니다. 자세한것은 Using a Programmer with MicroCode Studio 를 참고합니다. ICD 는 프로젝트를 ICD 컴파일이나 ICD 컴파일 그리고 프로그램 이후에 연결되는 것을 명심하여야 합니다.
Note: 컴파일 버튼을 누르면 모든 오픈된 파일은 저장됩니다. 이것은 편집한 현재 파일을 PICBasic 컴파일러에게 전달하는 것을 확실히 합니다. 발생한 *.Hex 파일은 MCU 에 프로그램하며 ICD 에 의하여 동작합니다. ICD 가 혼자서 동작하지 않습니다. ICD 사용을 원치 않으면 compile 버튼을 사용합니다.

ICD Compile and Program

 ICD 컴파일 버튼은 MicroCode Studio 컴파일 툴바 버튼과 정확히 같은 방법으로 동작합니다. 그러나 추가적인 디버그 정보가 빌드에 자동으로 추가됩니다. ICD 컴파일은 헥사 파일 을 출력하며 자동적으로 선택한 프로그래머를 시작합니다. 자세한 것은 Using a Programmer with MicroCode Studio 를 참고합니다. ICD 는 프로젝트를 ICD 컴파일이나 ICD 컴파일 그리고 프로그램 이후에 연결되는 것을 명심하여야 합니다.
NOTE : 컴파일 버튼을 누르면 모든 오픈된 파일은 저장됩니다. 이것은 편집한 현재 파일을 PICBasic 컴파일러에게 전달하는 것을 확실히 합니다. 발생한 *.Hex 파일은 MCU 에 프로그램하며 ICD 에 의하여 동작합니다. ICD 가 혼자서 동작하지 않습니다. ICD 사용을 원치 않으면 compile and Program 버튼을 사용합니다.

ICD Control

 컨트롤 버튼은 ICD 에서 Run, Stop, Pause 그리고 Step 을 가능하게 합니다. Run 버튼을 클릭하기 이전에 ICD compile 이나 ICD compile and program 으로 프로젝트를 빌드하여야 합니다. ICD 가 스타트되면 ICD 의 동작을 중지하기 위하여 pause 버튼을 사용할수 있습니다. 다른 방법으로 ICD 를 시작하기 이전에 다운 포지션에서 pause 버튼을 사용할수 있습니다. 만약 break point 에 다다르면 MicroCode Studio 는 자동적으로 ICD 를 정지합니다.

ICD Communications Port

통신포트는 새로운 윈도우가 생기며 ICD 와 타겟 마이크로 컨트롤러와 ICD 의 통신을 시리얼 포트에 의하여 가능하도록 합니다. ICD 는 PIC 마이크로컨트롤러와 하드웨어 USART 를 이용하여 통신합니다. ICD 를 사용하기 위하여 하드웨어를 설정하는 자세한 것은 Setting Up Your Hardware 를 참고합니다.

ICD Animate


애니메이트 버튼이 Down 상태이면 MicroCode Studio 는 프로그램 실행과정을 애니메이트 합니다. 각 프로그램 라인이 하이라이트 된느것을 볼 수 있습니다. ICD Exploer 창 에서 모든 변수, SFR, Memory, Eeprom 이 갱신되는 것을 볼수 있습니다.
애니메이트가 Up 상태이면 MicroCode Studio 는 프로그램을 애니메이트 하지 않습니다. 편집기 창이나 ICD 탐색창이 변화하지 않습니다.

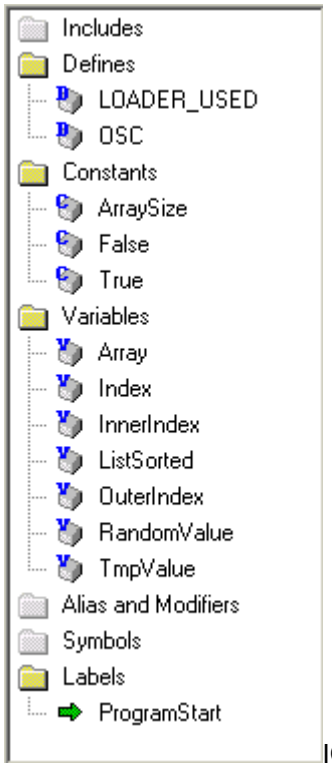
```
' initialise the array with
' random values for sorting
FOR Index = 0 TO ArraySize - 1
    RANDOM RandomValue
    Array(Index) = RandomValue
NEXT Index
```

ICD 는 타겟 디바이스의 오실레이터 설정과 호스트 PC 의 속도에 따라 애니메이트 속도가 다를 것입니다. OSC 주파수를 높게 하므로 더 빠른 속도가 가능합니다. 이상적인 경우 애니메이트 버튼이 Down 위치일때 초당 10 회의 명령 실행 속도를 기대할 수 있습니다. 애니매이션 동작을 정지하면 초당 수천회의 PICBasic 명령이 실행됩니다.
그러나 ICD 동작조건을 잘 설정하면 중요한 부분은 실제의 동작속도가 가능합니다. 자세한것은 ICD 를 구성하는 방법을 설명한 Reducing the ICD Code Overhead 와 Increasing the Speed of the ICD 를 참고합니다.

ICD Breakpoint


Breakpoint 는 어떤 위치이든 프로그램 실행을 정지합니다. MicroCode StudioICD 는 여러 개의 브레이크 포인트 설정을 허용합니다. PicBasic 내포 파일에도 브레이크 설정이 가능합니다. 브레이크 포인트를 설정하려면 브레이크 포인트 설정을 원하는 라인으로 커서를 가져갑니다. 다음에 ICD 브레이크 버튼을 클릭합니다.브레이크 포인트를 취소하려면 다시 브레이크 버튼을 누르면 됩니다. 더 쉽게 브레이크 포인트를 설정하거나 취소하는 방법이 있습니다. 주 편집창의 왼쪽에 회색 영역을 클릭하면 브리이크 포인트가 설정/취소 됩니다.

The ICD Explorer



ICD 가 실행되면 ICD 탐색창은 자동으로 열립니다. ICD 탐색창의 위의있는 탭은 선택한 변수가 표시되는 것을 가능하게 합니다. 탭은 변수, 특수기능 레지스터(SFR)메모리, EEPROM 가 있습니다. EEPROM 탭은 선택한 마이크로 프로세서에 EEPROM 이 내장되어 있을 때 표시됩니다. 데이터는 십진수, 헥사데시멀, 바이너리 포맷으로 표시됩니다. EEPROM 이 표시될 때 ASCII 값이 같이 표시됩니다.
PICBasic 명령이 실행되기 전에 ICD 는 마이크로 컨트롤러로 부터 뱅크된 메모리의 모든 덤프 데이터를 가져옵니다. 이것은 사용한 변수와 SFR 이름이 ICD 탐색창에 대응하여 표시됩니다. 변수와 레지스터 탭이 대응한 값을 표시합니다. 메모리 탭은 마이크로컨트롤러부터 넘어온 매핑되지 않은 모든

데이터를 표시합니다. EEPROM 은 예외여서 ICD 에 의하여 덤프 요청이 있을때만 표시됩니다. EEPROM 을 읽는것은 매우 느리기 때문에 PICBASIC 프로그램이 EEPROM 을 사용하지 않으면 EEPROM 의 내용을 덤프하기 위하여 ICD 가 늦게 동작하는것은 낮은 순서가 됩니다. (This is because reading EEPROM can be quite slow, so if your PICBasic program does not use EEPROM, there is little point in slowing down the ICD by dumping the contents of EEPROM.)

Reset MCU

ICD 탐색창에서 ResetMCU 를 클릭하면 ICD 는 소프트웨어 적으로 타겟 MCU 를 리셋할수 있습니다. 이것은 파워온 리셋과는 다릅니다. 내부 특수기능 레지스터(SFR)은 Microchip 사의 데이터 쉬트 처럼 초기화되지 않습니다. SFR 이 데이터 쉬트의 디폴트 값으로 초기화 되기 원하면 하드 리셋을 하여야합니다. 소프트웨어 리셋이든 하드웨어 리셋이든 ICD 는 프로그램의 처음 명령부터 실행합니다.

Update All

ICD 탐색기 창에서 "UpdateAll"을 마우스의 오른쪽 버튼으로 클릭하면 모든 데이터 메모리와 EEPROM 을 덤프할 수 있습니다. ICD 에게 데이터 메모리와 EEPROM 을 덤프하도록 하기때문에 애니매이션 버튼이 다운 위치에 있을때 약간 느리게 동작할 것입니다.

Bank Optimisation

일부 마이크로컨트롤러는 특별히 많은 양의 사용자 메모리가 가능하므로 ICD 애니매이션이 느리게 동작합니다. MicroCode Studio 는 애니메이션 버튼아 Down 위치에 있을때 특수기능 레지스터(SFR)와 사용자 변수만을 덤프하도록 뱅크 최적화를 이용합니다. ICD 탐색창에서 마우스 오른족을 클릭하여 뱅크 최적화를 선택할수 있습니다. 이는 토글 동작입니다. 뱅크 최적화가 Off 일때 모든 메모리가 덤프 됩니다.

Clear EEPROM

ICD 탐색 창에서 EEProm 탭을 선택한 상태에서 ICD 탐색창과 'Clear EEProm' 옵션을 마우스 오른쪽 버튼으로 클릭하면 EEProm 의 내용을 제로(0) 로 리셋할수 있습니다. EEPom 에 라이트하는것은 RAM 에 라이트하는 것보다 늦게 동작합니다. 이것은 많은 EEProm 을 내장한 대형 디바이스는 0 을 라이트 할때 시간이 필요합니다.

State Indicators

값이 변하지 않는 상태는 녹색원으로 표시됩니다. ICD 가 상태변화를 감지하면 가뵤은 적색원으로 배경은 회색으로 하이라이트 됩니다.

EEPROM 탭을 스위치 하거나 변수, 레지스터, 메모리탭을 스위치하면 값은 노란원으로 보입니다. 이것은 ICD 가 데이터 덤프 요구되었다는 것이며 업데이트가 실시됩니다. 업데이트는 update all 이 비 활성화 되어있을때만 보여집니다. 이것은 디폴트 상태 입니다.

어레이는 ICD 탐색기에서 많은 공간을 차지 합니다. 왜냐하면 각 어레이 위치는 각자의 슬롯을 가지기 때문입니다. 그러므로 ICD 탐색기는 플러스 기호를 사용하여 어레이가 확장되거나 축소되어 보이게 합니다. ICD 가 시작할 때에 어레이는 축소되어 있스빈다. 플러스 기호를 클릭하면 어레이의 개별 요소가 확장되어 표시됩니다.

한개 또는 그이상의 위치가 변화되었을때 어레이는 축소됩니다. 확장되었을 때 어레이의 개별위치의 상태 변화를 볼수 있습니다.

Breakpoints

Breakpoint 는 어떤 위치이든 프로그램 실행을 정지합니다. MicroCode StudioICD 는 여러 개의 브레이크 포인트 설정을 허용합니다. PicBasic 내포 파일에도 브레이크 설정이 가능합니다. 브레이크 포인트를 설정하려면 브레이크 포인트 설정을 원하는 라인으로 커서를 가져갑니다. 다음에 ICD 브레이크 버튼을 클릭합니다.브레이크 포인트를 취소하려면 다시 브레이크 버튼을 누르면 됩니다. 더 쉽게 브레이크 포인트를 설정하거나 취소하는 방법이 있습니다. 주 편집창의 왼쪽에 회색 영역을 클릭하면 브리이크 포인트가 설정/취소 됩니다.

ICD 가 동작할 때 PICBasic 프로그램은 브레이크 포인트 위치까지 실행된다. 브레이크 포인트를 만나면 ICD 는 프로그램의 실행을 정지하고 브레이크 포인트 원쪽에 작은 적색 화살표가 표시됩니다.

애니메이션 버튼이 업 위치일때 (금지상태) 브레이크 포인트를 만났을때 ICD 는 애니메이션을 자동적으로 가능상태로 만듭니다. ICD 는 모든 데이터를 덤프하게 되며 ICD 탐색기를 업데이트 합니다. Step 버튼을 클릭하면 각 라인을 실행하고 브레이크가 걸려 정지합니다. 다른 방법으로 Pause 버튼을 누르면 프로그램 실행이 정지됩니다. 브레이크 포인트는 적색 바로 표시됩니다. 그러나 ICD 를 시작하면 어떤 브레이크 포인트는 회색으로 됩니다. 닿지 않는 위치의 브레이크 포인트는 ICD 에서 문제되지 않습니다. 브레이크 포인트 위치의 프로그램 실행 정지 위치는 경고 메시지로 됩니다. 분명한 예는 주석문에 브레이크 포인트가 있을때입니다. ICD 는 프로그램 실행을 정지하지 않습니다. 주석문은 프로그램 명령이 아닙니다.

다른 예는 데이터 선언입니다.

```
Index VAR BYTE
```
프로그램 명령이아니므로 브레이크 포이트를 설정하여도 도달하지 않습니다.

브레이크 포인트가 DISABLE DEBUG 와 ENABLE DEBUG 지시어 사이에 있으면 ICD 는 프로그램에 제어 코드를 포함시키지 않으므로 브레이크 포인트는 동작하지 않습니다.

ENABLE DEBUG 와 DISABLE DEBUG 에 관한 자세한 정보는 ICD Code Overhead 감소와 Increasing the Speed of the ICD 을 참고합니다.

Breakpoint Properties

브레이크 포인트 특성을 보려면 브레이크 포이트에서 마우스 오른쪽 버튼을 클릭합니다.
지정한 횟수가 되면 프로그램이 정지하도록 패스 카운트를 설정할수 있습니다. ICD 는 브레이크 포인트를 만날 때마다 인덱스 값을 증가시킵니다. 예를들어 패스 카운트가 10 일때 프로그램의 실행은 10 번째 되는 라인에서 정지합니다. 패스 카운트는 디폴트로 0 이며 매번 프로그램이 정지합니다. 'count again when break point reached' 가 체크되면 패스 인덱스가 자동으로 리셋되며 카운트가 다시 시작합니다.

See Also

ICD Editing

ICD 가 동작할때 변수, SFR, 메모리, EEPROM 을 편집할수 있습니다. 동작중에 MCU 의 내부 변수를 변경하는 것은 매우 강력한 사양입니다. ICD 탐색기에서 편집하기 원하는 값을 클릭하면 작은 창이 생깁니다. 새로운 값을 입력하고 엔터키를 누르면 됩니다. 헥사값을 입력하려면 $ 접두사를 상요합니다. 바이너리 숫자는 % 를 사용합니다. 아무런 접두사를 사용하지 않으면 10 진수로 인식합니다. 만약 잘못된 숫자가 입력되면 엔터키가 입력된후 적색 폰트로 변경됩니다.
바이트 사이즈 변수에서 단지 8 비트만 사용 됩니다. 예를들어 $FF0F 를 바이트로 선언된 변수에 입력하면 '$0F'만 입력됩니다.

String Data

ICD 에서 EEPROM 을 볼때 EEPROM 으로 스트링을 입력할수 있습니다. 이렇게 하려면 EEPROM 스타트 어드레스를 클릭합니다. 편집 윈도우는 2 개의 탭을 보여줄것입니다.'String Data' 탭을 클릭하고 EEPROM 에 입력하기 원하는 스트링을 키인하고 엔터키를 입력합니다.
몇가지 선택사양이 있으며 스트링 데이터 탭에서 마우스 오른족 버튼을 클릭하면 됩니다.

Line Terminator

엔터키를 입력할대 라인 종료 문자를 자동적으로 붙입니다. 선택 가능한것으로 None, Null ($00), CR ($0D), LF ($0A), CR 과 LF ($0D0A), LF 과 CR ($0A0D).입니다.

Auto Increment EEPROM Address

이 옵션은 스트링 데이터 포인터를 자동적으로 다음 가능한 EEPROM 위치로 설정합니다. 예를들어 스타트 어드레스가 $00 일때 'hello'입력하면 다음 스타트 어드레스는 $00+길이('hello') + 길이(스트링 종료 기호)가 됩니다.

Auto Clear String Data

이 옵션은 다음 데이터 스트링을 빠르게 타이핑 하기 위하여 엔터키를 누른 후에 데이터 편집 박스에서 스트링 데이터를 자동으로 클리어 합니다.

An Example ICD Project

아마도 가장 좋은 방법은 예제 프로젝트를 ICD 를 사용하여 시작하는 것일 것입니다. Sort.bas 를 불러옵니다. 이 프로그램은 ICDSamples 폴더에 잇습니다. 간단한 튜토리얼은 시스템 사양과 마이크로 컨트롤러 하드웨어가 정확하다고 가정합니다.

Preparing the Source Program

Sort.bas 가 MicroCode Studio 로 읽은 후에 정확한 OSC 설정이 되었는지 확인 하여야 합니다. 디폴트로 4MHz 입니다. 만약 다른 주파수로 동작한다면 PICBasic 코드에서 정확한 오실레이터 설정 정의하는 것이 중요합니다. 예를들어 PICF628 을 10MHz 에서 동작시킨다면 다음과 같이 프로그램 초기에 설정하여야 합니다.

```
DEFINE OSC 10
```

정확한 오실레이터 주파수 설정을 하지 않으면 ICD 는 동작하지 않습니다. 즉 소스코드에서 설정한 오실레이터 주파수가 실제 도작 하드웨어와 일치하지 않으면 ICD 는 동작하지 않습니다.

```
DEFINE LOADER_USED 1
```

Compiling Your Program

다음으로 프로그램을 ICD Compile 이나 ICD Compile and Program 버튼을 클릭하여 빌드하여야 합니다. 이 버튼은 메인 ICD 툴바에 있습니다. ICD Compile 과 ICD Compile and Program 버튼은 MicroCode Studio 메인의

Compile 과 Compile and Program 과 같습니다. 그러나 빌드시 추가적인 디버깅 정보가 추가됩니다. ICD Compile 은 Hex 파일을 출력합니다. 이 파일은 별도의 프로그램 방법으로 프로그램합니다. ICD Compile and Program 은 Hex 파일을 만들어내며 자동적으로 선택한 프로그래머를 이용하여 프로그램 합니다. 프로그래머 설정 방법에 관하여는 Using a Programmer with MicroCode Studio 부분을 참고합니다. 이 버튼중 하나를 이용하여 빌드한 프로젝트를 하드오ㅣ어와 연결할수 있습니다.

Starting the ICD

ICD 를 시작하기 전에 선택한 시리얼 포트가 정확한지 확인합니다. ICD 와 타겟 마이크로 컨트롤러를 연결하는 시리얼ㄹ 포트는 communications port drop down box 에서 선택합니다. ICD 는 하드웨어 UART 를 내장한 PIC 마이크로 컨트롤러와 통신합니다. 하드웨어를 설정하는 자세한 것은 Setting Up Your Hardware 를 참고합니다.

ICD control button 은 Run, Stop, Pause 그리고 Step 이 있습니다. Run 버튼을 클릭하기전에 반드시 ICD Compile 또는 ICD Compile and Program 으로 프로젝트를 빌드하여야 하는것을 기억하여야 합니다. ICD 가 시작되면 한개의 라인만 실행하고 멈추는 step 실행하기 위하여 Pause 버튼을 이용하여 ICD 를 멈추게 할수 있습니다.

다른 방법으로 ICD 를 시작하기 전에 pause 버튼을 down 위치로 합니다. 프로그램 실행중 breakpoint 를 만나면 MicroCode Studio 는 자동적으로 ICD 를 정지합니다. Animate 버튼을 down(디폴트)하면 MicroCode Studio 는 프로그램을 애니메이트 합니다.

즉 프로그램 실행중에 main editor window 에는 각 프로그램 라인이 하이라이트 되는것을 볼수 있습니다. ICD 탐색기는 현재의 모든 변수, SFR, 메모리 그리고 EEPROM 을 업데이트 합니다. 애니메이트 버튼이 up 일때 MicroCode Studio 는 프로그램 실행을 애니매이트 하지 않습니다. Main editor window 또는 ICD Explorer 는 업데이트 되지 않습니다.

ICD 는 타겟 디바이스의 오실레이터 설정과 호스트 PC 의 속도에 따라 애니메이트 속도가 다를 것입니다. OSC 주파수를 높게 하므로 더 빠른 속도가 가능합니다. 이상적인 경우 애니메이트 버튼이 Down 위치일때 초당 10 회의 명령 실행 속도를 기대할 수 있습니다. 애니매이션 동작을 정지하면 초당 수천회의 PICBasic 명령이 실행됩니다.

그러나 ICD 동작조건을 잘 설정하면 중요한 부분은 실제의 동작속도가 가능합니다. 자세한것은 ICD 를 구성하는 방법을 설명한 Reducing the ICD Code Overhead 와 Increasing the Speed of the ICD 를 참고합니다.

Problems

앞서 설명한대로 하면 ICD 가 동작하지만 그러나 문제가 생기면 ICD Problems 을 참고합니다. 그래도 해결되지 않으면 Mecanique 에 연락하여야 합니다.

ICD Serial Communications Overview

MicroCode Studio 는 ICD 가 동작중일 때에도 시리얼 데이터를 수신하거나 송신하는 것을 허용합니다. 시리얼 데이터는 PICBasic 의 HSERIN 명령을 이용하여 ICD Serial In 윈도우에서 보냅니다. HSEROUT 명령에 의하여 보낸 시리얼 데이터는 수신되며 ICD Serial Out 윈도우를 통하여 디스프레이 됩니다. MicroCode Studio 는 마이크로 컨트롤러 하드웨어를 완전히 제어하여 ICD 기능을 정확히 유지합니다. USART 를 정상적으로 설정하지 않았더라도 ICD 는 정상적으로 실행됩니다. 어떤 상요자 정의는 ICD Compile 이나 ICD Compile and Program 과 내포과정에서 금지됩니다. (Certain user defines are therefore disabled during an ICD Compile or ICD Compile and Program and include:)

```
HSER_BAUD     'baud rate
HSER_SPBRG    'spbrg init
HSER_RCSTA    'receive status init
HSER_TXSTA    'transmit status init
HSER_EVEN     'even parity
HSER_ODD      'odd parity
```

ICD 는 메인 프로그램에 금지로 되어 있다면 하드웨어 시리얼 정의 는 금지됩니다. MicroCode Studio 는 내포파일에 있을때 정의를 금지하지 않습니다. 어떤 경우에는 인클루드 파일에서 하드웨어 시리얼 정의는 ICD 의 실행을 실패로 합니다. 추가적으로 MicroCode Studio ICD 는 메인 프로그램 블록에서 사용되었다면 HSERIN 과 HSEROUT 을 지원합니다.

MicroCode Studio ICD 는 디폴트 PICBasic 시리얼 데이터 포맷인 8 데이터 비트, 노 패리티, 원 스톱비트, (8N1)으로 컴퓨터와 통신합니다. 아래의 예를 사용합니다.

```
DEFINE HSER_SPBRG Value  'spbrg init
DEVICE HSER_TXSTA Value  'transmit status init
```

이 값을 보는것을 원치 않을것입니다. 그러나 프로그래머는 ICD 탐색창에서

'ICD Serial Settings...' 를 마우스 오른쪽 버튼으로 클릭하여 MicroCode Studio ICD 에서 사용된 실제의 값을 볼수 있습니다. ICD 는 다음의 TCSTA 를 사용할수 있으며 이것은 PICBasic 의 디폴트 입니다.

```
DEFINE HSER_RCSTA 90H     'receive status init
```

이것은 ICD 를 사용할때 HSERIN 과 HSEROUT 을 구성할 필요가 없음을 의미합니다. 일반적으로 ICD 는 자동으로 실행합니다. 그러나 마지막 코드 빌드에서 HSERIN 과 HSEROUT 이 사용되었다면 하나 또는여러개의 define 이 필요합니다.

Example

여기에 간단한 예를 보입니다. HSERIN 을 사용하여 값을 읽고 HSEROUT 을 이용하여 반송합니다. 마이크로 컨트롤러는 20MHz 로 동작하며 부트로더 소프트웨어가 프로그램 되어 있는것으로 가정합니다.

```
DEFINE LOADER_USED 1
DEFINE OSC 20
CR CON 13
Char VAR BYTE
HSEROUT ["Program Starting...",CR]
ProgramStart:
   HSERIN [Char]
   HSEROUT ["Char = '", Char,"' is ASCII = ",DEC Char,CR]
   GOTO ProgramStart
```

위의 예제는 ICD 에서만 동작합니다. 그러나 일반적인 컴파일 그리고 프로그램에서 동작하지 않습니다. (즉 ICD Compile 또는 ICD Compile and Program 을 사용하지 않았을 때) ICD 탐색창에서 'ICD Serial Setting...'을 마우스 오른쪽 버튼으로 클릭하면 MicroCode Studio 의 정의를 볼수 있습니다. 최종 프로그램에서 HSERIN 과 HSEROUT 을 사용하기 원하면 위의 정의를 소스 코드에 아래의 예와 같이 추가합니다.

```
DEFINE HSER_TXSTA 24H ' add TXSTA value, high speed
DEFINE HSER_SPBRG 0AH ' add SPBRG value

DEFINE LOADER_USED 1
DEFINE OSC 20
CR CON 13
Char VAR BYTE
HSEROUT ["Program Starting...",CR]
ProgramStart:
   HSERIN [Char]
   HSEROUT ["Char = '", Char,"' is ASCII = ",DEC Char,CR]
   GOTO ProgramStart
```

ICD Serial In

ICD Serial In 윈도우는 PICBasic HSERIN 명령의 시리얼 명령과 통신하는것을 허용합니다.

The ICD Serial In window 는 HSERIN 문을 만나면 자동으로 나타납니다. ICD 탐색 창에서 'ICD Serial In..'을 마우스 오른쪽 버튼으로 클릭하면 ICD Serial 윈도우가 오픈됩니다. 단순히 타이프 하고 엔터키를 누르면 시리얼 데이터가 출력 됩니다. 편집기 문자는 ICD 에서 마이크로 컨트롤러로 전송될 때마다 Blue 에서 Grey 로 변경됩니다.

시리얼 데이터 중에 제어 문자도 포함할수 있습니다. 예를들어 개행 문자를 스트링에 포함 되길 원할 것입니다.

ICD Serial In Control

컨트롤 버튼은 ICD 툴바 메뉴와 비슷한 방법으로 동작합니다. 메시지를 보내거나, 정지시키거나, 일시 멈춤하거나 또는 싱글 스텝으로 시리얼 데이터를 보낼수 있습니다. Send message 버튼을 누르면 에디터의 스트링을 로드하여 트랜스미션 버퍼로 보냅니다. HSERIN 명령이 실행될때 트랜스미션 버퍼는 HSERIN 피드하기 위해 사용됩니다. Stop 을 누르면 트랜스미션 버퍼를 클리러 합니다. Pause 버튼을 선택하면 송신을 중지하며 HSERIN 으로 한개의 바이트만 보내려면 step 을 사용합니다.

ICD Serial In Message Loop

The transmission buffer feeds a HSERIN instruction until the buffer is empty, at which point transmission to HSERIN will stop. Rather than manually having to send the message again, you can use the loop button to automatically re-transmit. Looping is enabled when the button is in the down position.

HSERIN 명령은 송신 버퍼에 버퍼가 비워지기 전까지 데이털르 보내며 버퍼가 비워지면 HSERIN 은 정지합니다. 매뉴얼 방식으로 메시지를 다시 보내기 보다는 루프 버튼을 사용하여 자동적으로 재 송신 되게 합니다. 루핑은 버튼이 down 위치일 때 가능상태가 됩니다.

ICD Serial In History

Send 메시지 버튼을 누를때마다 메시지 스트링은 히스토리 리스트에 저장됩니다. 전진 또는 후진 버튼을 사용하여 히스토리를 빨리 움직여 볼수 있습니

다. 빠르게 앞서 보낸 메시지를 확인 해 볼수 있습니다. 다시 타이핑 할 필요가 없습니다. 스태터스 바는 현재 선택된 히스토리를 디스프레이 합니다. 히스토리를 지우려면 ICD Serial In editor 윈도에서 마우스 오른쪽 버튼을 클릭하고 'Delete History Item'을 클릭하면 됩니다.

ICD Serial Out

ICD Serial Out 윈도우는 PICBasic HSEROUT 명령으로 부터 시리얼 정보를 수신하고 표시하는것을 허용합니다. ICD Serial 윈도우는 HSEROUT 명령을 만나면 자동으로 나타납니다. 다른 방법으로 ICD 탐색기 창에서 마우스 오른쪽 버튼을 클릭하고 'ICD Serial Out..'을 클릭하면 됩니다. ICD Serial Out 윈도우는 마이크로컨트롤러로 부터 수신된 시리얼 데이터를 디스프레이하며 필요하면 단어를 모읍니다. 캐리지 리턴 제어 코드(십진 13, 헥사 $D)를 시리얼 스트림으로 보내면 새 라인이 됩니다. 또한 라인피드(십진 10, 헥사 $A) 또는 CRLF 를 사용할수 있습니다.

MicroCode Loader

Overview

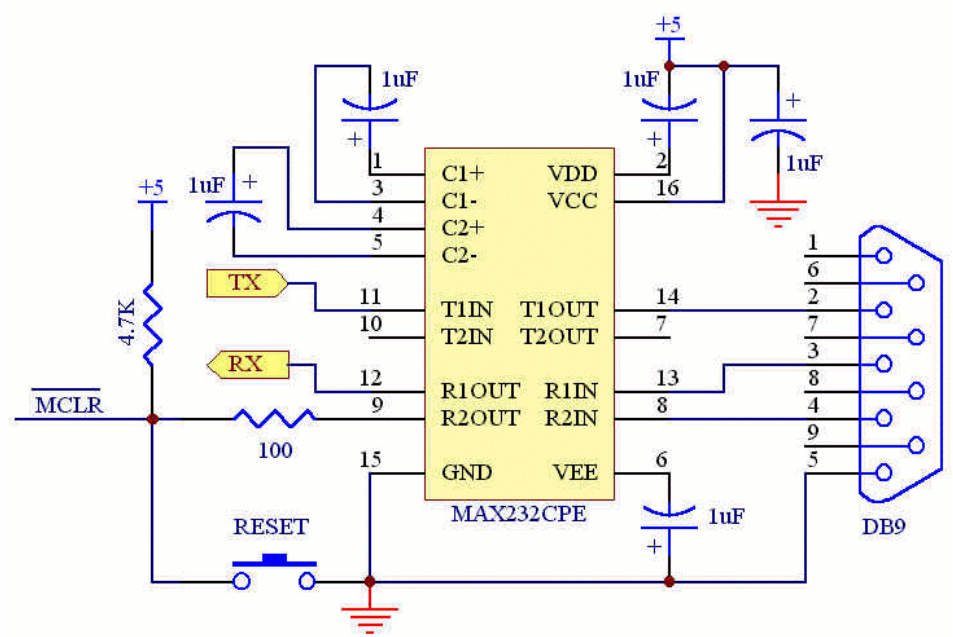
PIC16F87x(A) 와 PIC18Fxxx(x) 시리즈 마이크로 컨트롤러는 하드웨어 프로그래머가 없이 자신의 프로그램 메모리에 라이트 가능한 기능을 가지고 있습니다. 부트 로더라고하는 작은 작은 크기의 소프트웨어가 타겟 마이크로컨트롤러에 놓여지게 되며 상요자 코드와 EEPROM 데이터가 시리얼 케이블을 통하여 마이크로컨트롤러에 라이트 합니다. MicroCode Loader Application 은 컴퓨터에서 실행되는 프로그램입니다. 이 두 요소는 사용자에게 프로그램과 EEPROM 의 Write, Verify, Read 가 가능하게 합니다.

The Bootloader Software

MicroCode Loader 를 사용하기 이전에 부트로더 프로그램이 타겟 마이크로컨트롤러에 프로그램 되어 있는지 확인하여야 합니다. 부트로더가 마이크로컨트로러에 프로그램 되어 있어야 시리얼 케이블을 통한 타겟 마이크로컨트롤러의 프로그래밍이 가능합니다. 부트로더 *.HEX 파일은 타겟 마이크로컨트롤러와 클럭 주파수가 동일한지 확인하여야 합니다. 예를들어 16F877_20.HEX 는 20MHz 로 동작하는 마이크로컨트롤러 회로에 프로그램 되어야 합니다. 이렇게 하지 않으면 MicroCode Lader 는 타겟마이크로컨트롤러와 통신할수 없습니다. 사용자는 MicroCode Loader Application 에서 바우드레이트를 자동으로 인식하므로 설정할 필요가 없습니다. MicroCode Loader 는 미리 만들어진 많은 *.HEX 파일이 있습니다. 마이크로 컨트롤러에 파워가 공급되면 (또는 리셋) 부트로더는 무었을 할것인지 결정합니다.(예를들어 타겟 디바이스에 사용자 코드 프로그램) 부트로더는 빠져나오기 전에 MicroCode Loader 를 제어합니다. 그러나 부트로더가 수백 밀리초 이내에 어떤 명령도 수신하지 못하면 부트로더는 빠져나오며 앞서 라이트된 코드를 실행하기 시작합니다. 부트로더 소프트웨어는 256 워드(18Fxxx 디바이스에서 336 워드)의 프로그램 메모리를 점유합니다. 나머지 마이크로컨트롤러 코드 스페이스가 사용자 프로그램 영역이 됩니다. 모든 EEPROM 데이터 메모리와 마이크로컨트롤러 레지스터가 사용자 프로그램에서 사용가능합니다. MicroCode Loader 는 단지 프로그램 코드 영역과 EEOROM 데이터 영역만 Write, Verify 그리고 Read 가 가능합니다. 마이크로컨트롤러의 ID 영역과 configuration fuse 는 MicroCode Loader 에서 관여하지 않습니다. Configuration Fuse 는 타겟 마이크로컨트롤러에 부트로더 소프트웨어가 프로그램될때 설정하여야 합니다.

Hardware Requirements

MicroCode Loader 는 타겟 마이크로컨트롤러의 하드웨어 USART(Universal Synchronous Asynchronous Receiver Transmitter)와 통신합니다. PIC 16F87x 패밀리 마이크로 컨트롤러는 PORTC.7 에 RX 그리고 PORTC.6 에 TX 가 배치되어 있습니다. MicroCode Loader 를 사용하려면 개발 보드에 RS-232 시리얼 통신 드라이버 회로가 있어야 합니다. 보드에 9 핀 시리얼 컨넥터를 가지고 있으면 MAX232 를 사용하여 쉽게 RS-232 드라이버 회로를 만들수 있습니다. 이것이 타겟 마이크로 컨트롤러와 컴퓨터를 시리얼 케이블로 연결하여 MicroCode Loader 가 통신하는 적합한 방법입니다. 아래에 회로를 나타냅니다.



Connecting a microcontroller to MicroCode Loader

with optional software reset. Courtesy Reynolds Electronics

Programming Requirements

부트로더 소프트웨어는 256 워드(18Fxxx 디바이스에서 336 워드)의 프로그램 메모리를 점유합니다. 나머지 마이크로컨트롤러 코드 스페이스가 사용자 프로그램 영역이 됩니다. Loader 가 프로그램 메모리의 상위를 점유합니다. Loader 가 프로그램 메모리의 상위를 점유하므로 처음 파워가 공급될때와 마이크로컨트롤러가 리셋될때 부트로더의 시작 번지로 점프한느것이 필요합니다. 이렇게 하려면 4 개의 프로그램 워드 첫번째(리셋 벡터)를 부트로더 코드로 점프하도록 합니다. MicroCode Loader 를 사용하여 타겟마이크로컨트롤러를 프로그램할때 프로그램 명령은 처음 4 개의 위치를 부트로더가 자동적으로 배치합니다. 만약 부트로더가 이 명령을 재배치 하지 않으면 부트로더 소프트웨어는 다음번 파워가 공급될때 또는 마이크로컨트롤러가 리셋될때 시작하지 않습니다. (If the bootloader did not relocate these instructions, the important 'jump to bootloader' would get overwritten and the bootloader software would not start next time power was applied or the microcontroller was reset.) 그러므로 사용자 프로그램은 이를 위하여 수정할 필요가 있습니다. 약간만 변경하며 아래에 예를 보여줍니다.

PICBasic Pro, prior to version 2.33

```
DEFINE ONINT_USED 1
```

PICBasic Pro, 2.33 and above

```
DEFINE LOADER_USED 1
```

Assembler programs can use something like this:

```
   ORG 0
   GOTO StartOfProgram
   ORG 5
StartOfProgram
   ; your program goes here
```

Supported Devices

MicroCode Loader 는 다음의 마이크로 컨트롤러를 지원합니다:

16F870, 16F871, 16F873(A), 16F874(A), 16F876(A), 16F877(A), 18F242, 18F248, 18F252, 18F258, 18F442, 18F448, 18F452, 18F458, 18F1220, 18F1320, 18F2220, 18F2320, 18F4220, 18F4320, 18F6620, 18F6720, 18F8620 and 18F8720.

MicroCode Loader 는 타겟 마이크로 컨트롤러에 프로그램할 *.HEX 파일이 있습니다.

Devices running at 4 MHz	Devices running at 20 MHz
16F870_04.hex	16F870_20.hex
16F871_04.hex	16F871_20.hex
16F873_04.hex	16F873_20.hex
16F874_04.hex	16F874_20.hex
16F876_04.hex	16F876_20.hex
16F877_04.hex	16F877_20.hex
16F873A_04.hex	16F873A_20.hex
16F874A_04.hex	16F874A_20.hex
16F876A_04.hex	16F876A_20.hex
16F877A_04.hex	16F877A_20.hex
18F242_04.hex	18F242_20.hex
18F248_04.hex	18F248_20.hex
18F252_04.hex	18F252_20.hex
18F258_04.hex	18F258_20.hex
18F442_04.hex	18F442_20.hex
18F448_04.hex	18F448_20.hex
18F452_04.hex	18F452_20.hex

18F458_04.hex	18F458_20.hex
18F1220_04.hex	18F1220_20.hex
18F1320_04.hex	18F1320_20.hex
18F2220_04.hex	18F2220_20.hex
18F2320_04.hex	18F2320_20.hex
18F4220_04.hex	18F4220_20.hex
18F4320_04.hex	18F4320_20.hex
18F6620_04.hex	18F6620_20.hex
18F6720_04.hex	18F6720_20.hex
18F8620_04.hex	18F8620_20.hex
18F8720_04.hex	18F8720_20.hex

Quick Start Guide

Using the MicroCode Loader is very easy. This quick start guide assumes you already have a target microcontroller programmed with the necessary bootloader software and that you have a suitable development board for the target microcontroller. You will also need to make a slight modification to your source code before using MicroCode Loader.

Using the Loader with MicroCode Studio Plus

- Connect a serial cable between your computer and development board. Apply power to the board.
- From the ICD toolbar, select the serial port you will be using to communicate with the development board.
- Ensure that the MicroCode Loader option is selected by using the drop down selection box to the right of 'Compile and Program' or 'ICD Compile and Program'.
- Press either 'Compile and Program' or 'ICD Compile and Program'. At this point, the MicroCode Loader application software may ask you to reset the development board in order to establish communications with the resident microcontroller bootloader. Press reset to establish communications.

Using the Loader as a Stand Alone Application

- Connect a serial cable between your computer and development board. Apply power to the board.
    - Start the MicroCode Loader application software.
    - From the main toolbar, select the serial port you will be using to communicate with the development board.
    - Open the *.hex file you want to program into the target microcontroller.
    - Press the program button. At this point, the MicroCode Loader application software may ask you to reset the development board in order to establish communications with the resident microcontroller

Loader Toolbar

Open Hex File

The open button loads a *.hex file ready for programming.

Program

The program button will program the loaded hex file code and EEPROM data into the target microcontroller. When programming the target device, a verification is normally done to ensure the integrity of the programmed user code and EEPROM data. You can override this feature by un-checking either Verify Code When Programming or Verify Data When Programming. You can also optionally verify the complete *.hex file after programming by selecting the Verify After Programming option.

Pressing the program button will normally program the currently loaded *.hex file. However, you can load the latest version of the *.hex file immediately before programming by checking Load File Before Programming option. You can also set the loader to start running the user code immediately after programming by checking the Run User Code After Programming option. When programming the target device, both user code and EEPROM data are

programmed by default (recommended). However, you may want to just program code or EEPROM data. To change the default configuration, use the Program Code and Program Data options.

Should any problems arise when programming the target device, a dialog window will be displayed giving additional details. If no problems are encountered when programming the device, the status window will close at the end of the write sequence.

Verify

The verify button will compare the currently loaded *.hex file code and EEPROM data with the code and EEPROM data located on the target microcontroller. When verifying the target device, both user code and EEPROM data are verified by default. However, you may want to just verify code or EEPROM data. To change the default configuration, use the Verify Code and Verify Data options.

Should any problems arise when verifying the target device, a dialog window will be displayed giving additional details. If no problems are encountered when verifying the device, the status window will close at the end of the verification sequence.

Read

The read button will read the current code and EEPROM data from the target microcontroller.

Should any problems arise when reading the target device, a dialog window will be displayed giving additional details. If no problems are encountered when reading the device, the status window will close at the end of the read sequence.

Erase

The erase button will erase all of the code memory on a PIC18Fxxx microcontroller.

Loader Information

The loader information button displays the loader firmware version and the name of the target microcontroller, for example PIC16F877.

Run User Code

The run user code button will cause the bootloader process to exit and then start running the program loaded on the target microcontroller.

Loader Serial Port

The loader serial port drop down box allows you to select the com port used to communicate with the target microcontroller.

See Also

Configuration Options

Loader Options

Loader options can be set by selecting the OPTIONS menu item, located on the main menu bar.

Program Code

Optionally program user code when writing to the target microcontroller. Uncheck this option to prevent user code from being programmed. The default is ON.

Program Data

Optionally program EEPROM data when writing to the target microcontroller. Uncheck this option to prevent EEPROM data from being programmed. The default is ON.

Verify Code When Programming

Optionally verify a code write operation when programming. Uncheck this option to prevent user code from being verified when programming. The default is ON.

Verify Data When Programming

Optionally verify a data write operation when programming. Uncheck this option to prevent user data from being verified when programming. The default is ON.

Verify Code

Optionally verify user code when verifying the loaded *.hex file. Uncheck this option to prevent user code from being verified. The default is ON.

Verify Data

Optionally verify EEPROM data when verifying the loaded *.hex file. Uncheck this option to prevent EEPROM data from being verified. The default is ON.

Verify After Programming

Performs an additional verification operation immediately after the target microcontroller has been programmed. The default is OFF.

Run User Code After Programming

Exit the bootloader process immediately after programming and then start running the target user code. The default is ON.

Load File Before Programming

Optionally load the latest version of the *.hex file immediately before programming the target microcontroller. The default is OFF.

Baud Rate

Select the speed at which the computer communicates with the target microcontroller. By default, the Auto Detect option is enabled. This feature enables MicroCode Loader to determine the speed of the target microcontroller and set the best communication speed for that device. If you select one of the baud rates manually, it must match the baud rate of the loader software programmed onto the target microcontroller. For devices running at less that 20MHz, this is 19200 baud. For devices running at 20MHz, you can select either 19200 or 115200 baud.

See Also

The Main Toolbar