Microchip

PIC®16CXX MICROCONTROLLER ASSEMBLER GUIDE

(한 글 판)

현명전자

서울시 종로구 장사동 173
세운상가 가동 바-443
〒 110-430

대표 전화: 275-4455
277-7061
기술개발실: 278-0282
F A X: 277-8470

Microchip

PIC®16C5X MICROCONTROLLER ASSEMBLER GUIDE

목 차

1 장 : The PIC16C5X Cross Assemler

이 책에서는 Microchip Technology Inc. 에서 개발한 PIC Assembler (PICALC) 의 테크니컬한 면을 설명하고 있으며 주요특성들, 프로그램 구조, 지시어 요약, 매크로 기능, I/O 파일들 그리고 명령어에 대해서도 자세히 설명하고 있다.

참고로 S/W Tools 디스켓의 "PICALC.DOC" 라는 파일에는 User's guide가 출판 되었을때 누락, 첨가된 내용이나 수정사항들이 들어있다.

1.1 어셈블러 방법

어셈블러는 어셈블러 명령(소스코드)을 기계가 이해할 수 있는 코드(기계어)로 번역해주는 소프트웨어 툴이다. 어셈블러 명령들은 프로그래머가 기계어 명령을 효율적으로 수행할 수 있도록 쉬운 형태로 되어 있다.

어셈블리어는 일반적으로 니모닉과 지시어로서 되어있다. 니모닉들은 직접 기계어로 번역되는 명령들이다. 이들은 메모리나 레지스터에 있는 데이터를 가지고 산술연산과 논리연산을 수행한다. 또한 이들은 메모리나 레지스터로 데이터를 넣거나 꺼낼 수 있으며 지정된 번지로 프로그램을 분기시킬 수도 있다. 지시어들은 메모리의 어떤 영역으로도 분기하는 것을 쉽게 해주며 기호를 사용하여 메모리 영역을 초기화 시킬 수도 있다. 지시어는 또, 다른 파일이나 라이브러리의 소스코드를 포함시키기도 한다. PICALC 어셈블러는 위에 언급된 특징들은 물론 다른 많은 특징들을 가지고 있다.

1.2 PICALC 의 주특성들 - 개요

- MS-DOS 3.0 또는 그이상을 사용하는 IBM PC 나 그 호환기종 그리고 MS-DOS 3.3A 를 사용하는 NEC 9800 시리즈 컴퓨터에서 PIC® 어셈블리어로 씌어진 프로그램(소스코드)을 실행 가능한 기계어로 번역한다.
- 마이크로칩사의 PIC 개발장비 뿐 아니라 타사의 EPROM 프로그래머에도 Write가 될 수 있도록 4가지 형태의 목적코드를 만들수 있다.
- PIC16C54, PIC16C55, PIC16C56, PIC16C57 을 포함하여 마이크로칩사의 PIC® 마이크로콘트롤러 패밀리의 소스코드를 지원하고 목적코드로 만든다.
- 매크로의 모든기능 사용가능.
- Hex, Decimal, Octal 의 소스코드와 리스팅 포맷을 지원하며 디폴트로 Hex를 지원한다.
- 조건적인 어셈블리 가능.

1.3 PICALC 설정

어셈블러는 아래의 코맨드와 함께 수행된다. :

```
PICALC  [-?]
        [-P <processor>]
        [-f <object_format>]
        [-w <warning level>]
        [-w-<error code>]
        [-r <radix>]
        [-i]
```

위의 코맨드에서 :

<source_file>	-	파일 확장명을 포함한 소스파일의 이름.
<object_format>	-	목적코드의 출력 포맷을 설정 (setion 1.3.1 참조).
<processor>	-	프로세서 타입 설정 (16C54, 16C55, 16C56, 16C57).
<warning level>	-	표시할 경고레벨의 설정.
<error code>	-	제거 하고싶은 에러번호의 설정.
<radix>	-	어셈블될 상수의 기수값.
<tab width>	-	탭 스페이스의 설정.

1.3.1 코맨드 옵션

-? 코맨드 옵션을 설명하는 간단한 도움말 화면을 디스플레이 한다. 어셈블러는 도움말 화면을 표시한후 즉시 빠져 나온다.

-p <processor> 어셈블리시에 사용할 프로세서의 타입을 지정한다. <processor>는 **PIC16C54, 16C55, 16C56, 16C57**이며 디폴트로는 **16C54**로 되어있다.

-f <object format> 목적파일의 출력 포맷을 지정한다. <object format>은 **INHX16, INHX8S, INHX8M** 또는 **PICICE** 이며 디폴트는 **PICICE** 이다.

-w <warning level> 리스팅시에 표시되어야 할 경고레벨을 지정한다. PICALC에는 다음 4가지의 메시지 레벨이 있다 : comments, warnings, fatals, criticals. <warning level> 은 **0**(Comments), **1**(Warnings), **2**(Fatals), **3**(Criticals)의 4가지이다. 만일 유저가 메시지 표시레벨을 설정하면 그 표시레벨 이상의 메시지만 표시된다. 예를들어 '-w 2' 가 지정되면 "FATAL"과 "CRITICAL"메시지만 표시된다.

-w-<error code> 리스팅으로 부터 지우고 싶은 에러 메시지를 지정한다. **주의:두번째 '-'와 에러코드 사이에는 공백이 없어야 한다.** <error code>는 번호 0에서 64중 하나이다. 유저는 공간이 허락하는한 에러코드당 1개의 -w- 옵션을 가능한 많이 사용할 수 있다. 에러 메시지에 대한 사항은 부록 B 를 참고하기 바란다.

-r <radix> 어셈블될 상수의 기수값을 지정한다. <radix>는 **DEC, OCT, HEX** 이고 디폴트는 **HEX** 이다.

t <tab width>　소스파일의 탭 스페이스를 지정한다. <tab width> 는 행의 스페이스 수를 재설정 해준다.

-i　이 옵션은 대,소문자를 구분하지 않는다. 디폴트로는 대,소문자를 구분한다.

괄호 [ ] 안의 코맨드는 선택사용할 수 있다.

각 옵션사이는 띄어 써야한다.

모든 소스 파일들은 확장명 'ASM'으로 가정하며 그외 다른 확장명을 사용할 경우 따로 주어져야 한다.

어셈블러는 아래와 같이 다양한 확장명을 가지는 출력파일들을 생성한다.

- OBJ - 목적 파일.
- LST - 리스팅 파일.
- SYM - 심볼파일.
- REF - 심볼 참고 파일(PICICE포맷이 선택된 경우)
- MAP - 맵 파일(PICICE 포맷이 선택된 경우).

1.3.2 오브젝트 포맷

PICICE - PIC-ICE 에뮬레이터에 사용할 수 있는 파일을 생성한다. 이 포맷에서 TITLE 와 SUBTITLE 를 포함한 리스팅파일에 어셈블러 메시지는 없다. 그렇지만 디스플레이를 통하여 경고 메시지는 표시된다.

INHX8S - 이 포맷은 2개의 8비트 Hex 파일을 생성하는데 하나는 상위 8비트이고 다른 하나는 하위 8비트이다. 목적 코드의 파일 확장명은 상위 파일일 경우 '.obh'가 되고 하위 파일일 경우는 '.obl'이 된다. 이 포맷은 특히 NMOS PIC Demo 보드 (PFD 보드)를 위한 EPROM을 write 하는데 유용하게 쓰인다.

INHX8M - 상위 바이트와 하위 바이트가 하나로 통합된 Intellect Hex 파일을 제공하며 목적파일의 확장명은 '.obj'가 된다. 이 포맷은 특히 PIC16C5X 의 코드를 타사의 EPROM 프로그래머(DATA I/O 의 Unisite 40, Logical Devices 의 ALLPRO)에 다운로딩할 때 유용하다.

INHX16 - 하나의 Intellect Hex파일을 제공하며 목적 코드의 파일 확장명은 '.obj' 가 된다. 이 포맷은 마이크로칩사의 "PICPRO II" 프로그래밍을 위해 사용된다.

출력 포맷은 또한 소스 코드에 LIST 라는 지시어를 사용함 으로써 직접 선택할 수 있다. 이 선택은 코맨드 옵션에 의해 변경 될수 있음에 주의해야 한다.

만일 출력옵션이 주어지지 않으면 PICICE 포맷이 디폴트로 주어진다.

2장 : 프로그램 구조

프로그램 즉, 소스코드는 니모닉, 지시어, 매크로, 심볼, 수식, 상수 등으로 이루어져 있다. 이 모든 것들은 순차적 형식으로 실행되고 END 라는 지시어에 의해 끝마친다. 이들 각각은 이 부분 이후 에서 상세하게 다룰 것이다.

2.1 문자 세트

아래에 나열된 문자들은 어셈블러가 인식할 수 있는 문자들이다. 만약 이외의 다른 문자들을 사용하면 어셈블러는 에러 메시지를 발생시킨다.

그러나 코맨트는 어떤 문자를 써도 무방한데, PICALC는 이것들을 무시해 버리기 때문이다.

알파벳 문자들

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

주의 : PICALC는 대문자, 소문자 모두를 구분하며 디폴트로 이 옵션이 설정되어 있다.

숫 자

0 1 2 3 4 5 6 7 8 9

범용의 특수문자 :

? 물음표

_ 밑줄

파운드 기호

@ at 기호

공백문자(space)

특수 사용을 위한 예약문자 :

/	슬래쉬	%	퍼센트 기호
'	단일 인용 부호	*	별표
"	따옴표	(	왼 괄호
+	더하기	)	오른 괄호
-	빼기	=	등호
;	세미콜론	$	달러 기호
<	부등호	>	부등호
~	물결 표시	!	느낌표
|	파이프 기호	&	and 기호
^	탈자기호		

2.2 소스 라인 포맷

소스 코드는 소스라인의 정해진 시퀀스로 구성되어 있다. 소스라인의 구성은 다음과 같다.
[<label>] <command> [<operand>] [<comment>]

Example :

```
LOOP    CLRF  3          ;Clear file register 3.
```

위의 구문에서 주석 부분은 syntax에서 제외된다.. 이 주석 라인들은 첫 세미콜론(;)으로 표시되며, 이 코맨트 라인은 어셈블러에 의해 모두 무시된다. 세미콜론이 코맨트의 제일 앞에 와야 한다는 것에 주의한다.

이 어셈블러는 자유포맷 언어이다.즉 각각의 필드들이 특정칼럼 범위로 제한되지 않는다는 것을 의미한다. 자유로운 포맷을 사용하면 어떤 에러들은 디버깅 과정 에서 빠질수도 있다. 예를들어 유저가 아래의 라인을 썼다고 하자 :

```
RET ; return to calling program
```

그러면 어셈블러는 문자 'RET'를 틀린 니모닉으로 인식지지 않고 레이블로 인식하여 번역한다. 이러한 코드에러를 찾아내기 위하여 유저는 심볼 테이블이나 cross-reference 리스트에서 사용하지 않은 레이블들을 확인하여야 한다.

2.2.1 레이블 필드

레이블 필드는 어셈블러 니모닉보다 앞서 위치하며 공백 문자(공백은 스페이스나 탭 또는 end-of-line 문자)에 의해 끝난다. 레이블들은 선택적으로 쓸수 있으며 'A'-'Z','_' ,'@' ,'#',':' 를 포함한 최대 132 문자를 쓸수있다. 레이블들은 일반적으로 소스코드의 위치에 대한 참고적인 기호로 사용된다. 레이블은 라인의 모든 위치에서 시작될 수 있다.

2.2.2 코맨드 필드

코맨드 필드는 소스라인의 모든 위치에서 시작될 수 있고 그 다음의 공백문자에 의해 중단된다. 코맨드 필드는 어셈블러 니모닉, 지시어, 매크로 등을 포함한다.

2.2.3 오퍼랜드 필드

오퍼랜드 필드는 코맨드 필드 다음의 공백 다음 에서부터 시작하며 그 다음의 첫 공백문자에 의해 끝이난다. 그렇지만 몇몇 명령들은 오퍼랜드를 사용하지 않는다. 이러한 경우 주석필드는 오퍼랜드 필드의 공간에서 시작할 수도 있다. 오퍼랜드 필드는 쉼표에 의해 분리된 한개 또는 그 이상의 상수나 수식들을 포함한다. 상수나 수식의 갯수는 코맨드 종류에 따라 결정된다. 이 필드 사이에는 공백문자부터 포함할 수 있으나 중간에 캐리지 리턴을 포함 해서는 안된다.

Exmple :

```
FIVE EQU   3+2          ; Assign Values 5 to label FIVE.
```

이 표현은

```
FIVE EQU   3    +2      ; Assign Values 5 to label FIVE.
```

와 동일하다.

2.2.4 주석(comments) 필드

주석 필드는 세미콜론으로 부터 소스라인의 끝까지 이다. 주석은 반드시 세미콜론으로 부터 시작하여야 하며 만일 그렇지 않으면 어셈블러가 이것을 니모닉이나 지시어로 생각하여 어셈블 하려고 할것이다.

2.3 상수

오퍼랜드 필드에 나타나는 상수들은 여러가지 다른 형식으로 표시될 수 있다. 디폴트의 기수값은 Hex 이며 유저가 특별히 기수들을 설정하지 않으면 디폴트값으로 간주한다.

상수 앞에는 + 나 - 표시를 할 수 있으며 아무 표시가 없다면 양의값으로 간주된다.

상수값은 32 비트의 부호가 없는 정수로 취급된다.

필드에서 상수의 자리수가 너무크면 PICALC는 '자리잘림' 경고를 표시할 것이다. 상수를 선언하는 방식이 아래에 나타나있다. **기수의 표현은 대문자든 소문자든 관계없다.** 디폴트 기수값과 상수 포맷과의 비교를 위하여 부록 C 를 참조하기 바란다.

TYPE	SYNTAX	EXAMPLE
Decimal	<digits>	100
	<digits>D	100d
	.digits	.100
Hexadecimal	H'<hex digits>'	H'9F'
	<hex digits>H	900h
	0x<hex digits>	0x9FF
Octal	O'<octal digits>'	o'777'
	Q'<octal digits>'	Q'777'
	<octal digits>o	777o
	<octal digits>q	777Q
	\<octal digits>	\777
Binary	B'<binary digits>'	b'00111001'
	<binary digits>b	00111001B
Character	'<character>'	'C'
	A'<character>'	A'C'

2.4 심볼

심볼은 하나 혹은 그 이상의 알파뉴메릭 문자열('A'-'Z', 'a'-'z', '?', '_', '#', ':', '@' 등으로 시작되는)로서 표현된다. 심볼들은 소스라인에서 레이블이나 오퍼랜드로 사용된다. 심볼들은 레이블 필드에서 현 프로그램의 번지나 EQU 나 SET 의 결과치로서 정의된다. 이 값들은 오퍼랜드 필드에서 심볼로서 사용될 수 있다.

2.4.2 예약된 심볼들

어셈블러는 레지스터와 목적지 지정을 위하여 내부적으로 3 개의 심볼을 예약해 놓았다. "LOCAL"을 제외한 다른 심볼들은 유저가 재정의할 수 있다. 그것들은 다음과 같다 :

"W" 와 "w" 는 '0' 이고

"F" 와 "f" 는 '1' 이고

"LOCAL"은 매크로 정의에서 사용하기 위해 예약되어 있다.

2.5 수 식

수식은 소스라인의 오퍼랜드 필드에 사용되며 상수와 심볼 또는 산술연산자에 의해 분리되는 심볼과 상수의 모든 조합들을 포함한다. 각각의 상수나 심볼들은 아래의 표시들중 하나가 덧붙여진다 :

'+' 는 양의 값을 표시한다(디폴트).
'-' 는 음의 값을 표시한다(2의 보수)

2.5.1 산술 연산자들

수식에서 사용되는 산술연산자들은 아래와 같다 :

연산자	설명	사용예
+	Addition	3 + 4
+	Unary Plus	+x
-	Subtraction	3 - 4
-	Unary Minus	-x
%	Modulus	3 % 5
*	Multiplication	3 * 4
/	Division	3 / 4
<<	Left Shift	3 << 4
>>	Right Shift	3 >> 4
()	Parentheses	((3 + 4) / 5)
==	Logical Equal	x == y
!=	Logical Not Equal	x != y
<=	Less Than or equal	3 <= 5
>=	Greater Than or equal	3 >= 5
!	Not	! (x == y)
~	Compliment	~x
|	Inclusive OR	x | y
&	Inclusive AND	x & y
||	Logical OR	x || y
&&	Logical AND	x && y
^	Exclusive OR	x ^ y

연산순서는 왼쪽에서 오른쪽으로 이루어지며 괄호는 얼마든지 사용할 수 있다.

모든 연산은 부호가 없는 32 비트 정수값을 사용하므로 나눗셈으로 인한 소수부분은 잘려 나간다. 만일 수식이 오버플로우 상태를 만들면 경고가 표시된다.

 수식에 사용되는 모든 레이블은 완전히 2번 PASS를 하여 그 값이 결정된다.

3장 : 어셈블러 명령어 세트

3.1 일반적인 명령 형식

PIC16C5X시리즈의 명령어 세트는 33개의 기본명령어 목록으로 이루어져 있으며 3개의 범주로 나뉜다 :

- 범용 파일 레지스터 동작 (바이트 단위)
- 파일 레지스터의 비트 레벨 동작
- 상수나 제어 동작

각각의 PIC 명령어들은 12비트 워드로 되어 있으며 opcode, 소스 레지스터, 목적지 레지스터 혹은 상수로 나뉜다.

바이트 단위 명령에서 'f'는 파일 레지스터를 나타내고 'd'는 목적지를 나타낸다. 파일 지정자 'f'는 명령에서 사용될 PIC 파일들중 하나를 지정한다. 목적지 지정자 'd'는 명령 실행에 의한 연산결과가 위치할 파일 레지스터를 지정한다. 만일 'd'가 0이면 결과는 W 레지스터로 가고 결과가 1이면 명령에서 지정한 곳으로 간다. 그리고 만일 'd'를 생략하면 PICALC는 그값을 1로 인식한다.

비트단위 명령에서 'b'는 수행후 영향을 받을 비트 필드를 나타내며 'f'는 그 비트를 포함하고 있는 파일 레지스터를 나타낸다.

상수/제어 명령에서 'k'는 8 에서 11 비트의 상수를 나타낸다.

각각의 명령 워드는 12비트로 이루어진다. 2진으로 표현된 명령워드는 기계어나 목적코드 이다. 명령워드에서 몇개의 비트들은 opcode로 할당되는데 OPCODE는 실행될 연산타입을 지정한다. 명령어의 나머지 부분은 명령연산을 나머지를 지정하기 위해 하나 또는 그 이상의 오퍼랜드를 나타낸다.

범용 파일 레지스터 연산에서 6비트는 OPCODE 로 할당된다. 파일 레지스터의 비트연산을 위해 4비트가 할당되고 상수/제어 명령을 위해서는 3 또는4 비트가 할당된다.
오퍼랜드 필드는 아래의 정보를 포함한다 :

- 데이터를 구할 파일 레지스터의 번지
- W 레지스터로 부터 데이터를 받을 파일 레지스터의 번지
- 연산 결과치의 목적지 (파일 레지스터나 W 레지스터)
- 비트연산에 의해 영향을 받을 비트 번호
- 프로그램 카운터가 이동해야할 명령 번지
- 프로그램 메모리에 저장된 상수값

PIC 명령의 예를보자 ,W 레지스터에 hex 16을 넣는 목적코드는 110000010110이고 OPCODE와 Operand는 다음과 같다 :

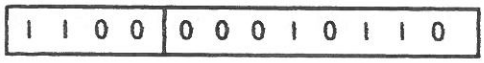
OPCODE	Operand
1 1 0 0	0 0 0 1 0 1 1 0

OPCODE 1100은 W 레지스터에 상수가 들어감을 지정한다. 2진수의 오퍼랜드는 hex 값 16 과 같다. 프로세서는 명령을 실행하기 위해 12 비트의 2진값을 사용한다.

일반적으로 이러한 2진 형태의 명령은 프로그래머가 읽거나 쓰기에 매우 불편하다. 그러므로 프로그램들은 프로그래머가 쉽게 이해하고 PIC Cross Assembler 가 실행할 수 있도록 심볼로서 작성된다.

심볼들을 사용하여 OPCODE는 니모닉으로 표현된다. 오퍼랜드는 8진수, 2진수, 16진수, 10진수, 또는 심볼로서 표현된다. 그러나 각 오퍼랜드는 특별히 지정되지 않으면 hex로 간주되고 그러지 않기 위해서는 따로 지정이 되어져야 한다.

PIC 목적코드 명령



은

아래와 같이 설명된다 :

OPCODE	Operand
MOVLW	16

같은 명령이면서 각기 다른 형식의 오퍼랜드로 표현된 예들이 아래에 있다 :

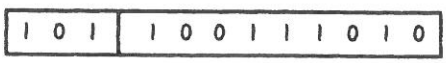
```
MOVLW   B'00010110'
        여기서 B'00010110'은 2진수로 나타낸 hex값 16이다.

MOVLW   026
        여기서 026은 8진수로 나타낸 hex값 16이다.

MOVLW   .22
        여기서 .22는 10진수로 나타낸 hex값 16이다.

MOVLW   SAMPLE
        여기서 SAMPLE은 상수를 대신하는 심볼이다. 이 심볼은 프로그램내의 적절한
        곳에서 정의되어  어셈블러가 "SAMPLE" 이란 표시를 만났을때 그에 상응하는
        정확한 2진값으로 대치할 수 있게 하여야 한다.
```

어셈블러를 사용하면 명령어나 명령어 그룹을 레이블화할 수 있다. 오퍼랜드로 분기될 명령의 번지를 제공하는 분기와 CALL명령 에서 레이블은 오퍼랜드로서 번지를 대신해 준다. 어셈블러는 레이블을 포함하는 오퍼랜드 필드에서 적절한 번지로 대치된다.

프로그램 위치 13A로의 분기명령은 다음과 같이 표현된다 :

```
GOTO    13A  또는
GOTO    OVFLO
```

여기서 OVFLO는 번지를 나타내는 레이블이나 심볼 이름이다. PIC 목적코드에서 명령은 다음과 같이 씌어진다 :



어셈블러는 주석 필드를 제공하는데 이 필드를 사용함으로써 프로그래머가 자신의 프로그램을 알아보기 쉽게 할 수 있다. 그러므로 보통의 어셈블리어 라인은 아래와 같다 :

Label	OPCODE	Operand	Comments

레이블과 코맨트 필드는 항상 쓰이는 것은 아니다.
서브루틴 호출과 조건에 의한 스킵, 분기들은 2기계 사이클 내에 실행되며 그외 모든 명령들은 1기계 사이클 내에 실행된다.

명령어 세트 요약

BYTE 단위 파일 조작 명령

(11-6)	(5)	(4-0)
OPCODE	d	f(FILE#)

d=0 : W 에 저장 d=1 : f 에 저장

BINARY	HEX	내용	니모닉	OP	동작	STATUS	비고
0000 0000 0000	000	동작 안함발	NOP	-	-	없음	
0000 001f ffff	02f	W에서 f로 옮김	MOVWF	f	W→f	없음	1,4
0000 0100 0000	040	W 클리어	CLRW	-	0→W	Z	
0000 011f ffff	06f	f 클리어	CLRF	f	0→f	Z	4
0000 10df ffff	08f	f에서 W빼기발	SUBWF	f,d	f-w→d[f+w+1→d]	C,DC,Z	1,2,4
0000 11df ffff	0cf	f 감소발	DECF	f,d	f-1→d	Z	2,4
0001 00df ffff	10f	W와 f의 OR	IORWF	f,d	W∪f→d	Z	2,4
0001 01df ffff	14f	W와 f의 AND	ANDWF	f,d	W&f→d	Z	2,4
0001 10df ffff	18f	W와 f의 XOR	XORWF	f,d	W⊕f→d	Z	2,4
0001 11df ffff	1cf	W와 f 더하기	ADDWF	f,d	W+f→d	C,DC,Z	1,2,4
0010 00df ffff	20f	f 옮기기발	MOVF	f,d	f→d	Z	2,4
0010 01df ffff	24f	f의 보수	COMF	f,d	f→d	Z	2,4
0010 10df ffff	28f	f 증가	INCF	f,d	f+1→d	Z	2,4
0010 11df ffff	2cf	f 감소,0이면 스킵	DECFSZ	f,d	f-1→d, skip if zero	없음	2,4
0011 00df ffff	30f	f를 우로 시프트	RRF	f,d	f(n)→d(n-1),C→d(7),f(0)→C	C	2,4
0011 01df ffff	34f	f를 좌로 시프트	RLF	f,d	f(n)→d(n+1),C→d(0),f(7)→C	C	2,4
0011 10df ffff	38f	니블 교체	SWAPF	f,d	f(0-3)↔f(4-7)→d	없음	2,4
0011 11df ffff	3cf	f 증가,0이면 스킵	INCFSZ	f,d	f+1→d, skip if zero	없음발	2,4

BIT 단위 파일 조작 명령

(11-8)	(7-5)	(4-0)
OPCODE	b(BIT#)	f(FILE#)

BINARY	HEX	내용	니모닉	OP	동작	STATUS	비고
0100 bbbf ffff	4bf	f의 비트 클리어	BCF	f,b	0→f(b)	없음	2,4
0101 bbbf ffff	5bf	f의 비트 셋트	BSF	f,b	1→f(b)	없음	2,4
0110 bbbf ffff	6bf	test bit, 0=스킵	BTFSC	f,b	test bit(b) in file(f),skip if clear	없음	
0111 bbbf ffff	7bf	test bit, 1=스킵	BTFSS	f,b	test bit(b) in file(f), skip if set	없음발	

(11-8)	(7-0)
OPCODE	k(LITERAL)

상수 / 제어 명령

BINARY	HEX	내용	니모닉	OP	동작	STATUS	비고
0011 00df ffff	30f	옵션 정의	OPTION	-	W→OPTION register	없음	
0011 01df ffff	34f	standby 모드 전환	SLEEP	-	0→WDT, stop oscillator	TO, PD	
0011 10df ffff	38f	WDT 클리어	CLRWDT	-	0→WDT (and prescaler, if assigned)	TO, PD	
0010 00df ffff	20f	포트 정의	TRIS	f	W→I/O control register f	없음	3
0010 01df ffff	24f	복귀 (W에 문자)	RETLW	k	k→W, Stack→PC	없음	
0010 10df ffff	28f	서브루틴 호출	CALL	k	PC+1→Stack, k→PC	없음	1
0010 11df ffff	2cf	분기 (k=9비트)	GOTO	k	k→PC (9bit)	없음	
0011 00df ffff	30f	문자를 W로 옮김	MOVLW	k	k→W	없음	
0011 01df ffff	34f	문자와 W의 OR	IORLW	k	k∪W→W	Z	
0011 10df ffff	38f	문자와 W의 AND	ANDLW	k	k&W→W	Z	
0011 11df ffff	3cf	문자와 W의 XOR	XORLW	k	k⊕W→W	Z	

참고 1: 8 비트로 이루어지는 모든 명령의 목적지가 PC이면 PC의 9번 비트는 클리어 된다. PIC16C56, PIC16C57 에서 STATUS 레지스터의 상위 3비트인 PA2-PA0는 PC의 상위 3비트(11-9번 비트)로 로드된다. GOTO 명령의 경우 PC의 하위 9비트는 목적지 번지로 로드되고 STATUS 레지스터의 PA2-PA0 비트는 PC의 상위 3비트로 로드된다.

참고 2: I/O 레지스터가 자신의 값을 read, write 하여 바꾸면(MOVF6, 1) 그 사용된 값은 핀 자체 상태의 값으로 된다. 예를들어 데이터 래치가 1인 입력 핀의 값이 외부 디바이스에 의해 low로 구동되면 그핀은 low 상태로 재 래치 된다.

참고 3: "TRIS f"(f=5,6,7)명령은 W 레지스터의 내용을 지정된 포트의 tristate 래치로 로드한다. "1"의 값은 포트를 Hi 임피던스 상태로 만들어 버리고 더이상 출력버퍼로 쓸수가 없다.

참고 4: 만일 명령이 f1 파일 레지스터에 실행되고 만약 프리스케일러가 RTCC에 할당 되었다면 그 값은 클리어 된다.

참고 5: 이 명령어들은 PIC16C5X 시리즈 에서만 사용할 수 있다.

3.2 범용 파일 레지스터 운용

명령어들은 I/O 레지스터를 포함한 모든 레지스터에 있는 데이터에 대해 아래의 명령어들을 수행할 수 있다 :

- 2개의 데이터 이동 명령
- 6개의 산술연산
- 6개의 논리연산
- 3개의 로테이트 동작

범용 레지스터에서 직접, 간접의 어드레스 모드를 사용할 수 있으며 일반적으로 직접 어드레싱 방법을 많이 사용한다.
직접 어드레스 모드는 1에서 1FH 내의 파일 레지스터 번지를 지정함으로서 이루어지고 OPCODE에 의해 지정되는 동작은 지정된 파일 레지스터의 데이터에 수행된다.

간접 어드레스 모드는 오퍼랜드 필드에 0번지의 파일 어드레스를 지정한다. 수행하면 파일선택 레지스터(F4, FSR)가 지정하는 레지스터의 데이터가 실행되어 진다. 파일 선택 레지스터는 프로그램 수행중에 로드 되어져야만 하기 때문에 간접 어드레스를 사용하기전에 먼저 FSR에 해당 파일 어드레스가 로드 되어져야 한다.

예제 : FSR에 파일 레지스터 번지인 F0A가 로드되어 있다고 가정하자. 간접 어드레스 명령인 MOVF 0, W가 씌어지면 FSR은 파일 레지스터 F0A를 지정하므로 따라서 0A 번지의 파일 레지스터 내용은 W 레지스터로 이동한다.

범용 레지스터 파일 명령은 아래와 같다 :

(11-6)	5	(4-0)
OPCODE	d	f

f = 파일 레지스터 번지 (일반적으로 hex로 표시)

d = 결과값이 보내질곳
    0 = w 레지스터
    1 = 파일 레지스터

명령은 심볼로써 표현된다 :

OPCODE f, d
여기서 f는 2진, 8진, 16진,10진 또는 심볼로서 표시할 수 있고 d는 W 레지스터를 나타내는 0또는 W로 표시하거나 파일 레지스터를 나타내는 1 또는 공백으로 표현할 수도 있다.

목적지를 명시하지 않으면 그 값을 1로 간주하며 따라서 목적지는 파일 레지스터가 된다.

예 : 파일 내용의 증가

```
INCF  6, 0  ;  (F6)+1→W

INCF  6, W  ;

INCF  6, 1  ;  (F6)+1→F6

INCF  6,    ;
```

3.2.1 데이터 이동 연산

PIC 명령어 세트에는 2개의 이동 명령이 있다. 그중 하나는 데이터를 W에서 파일 레지스터로 옮기는(MOVWF) 것이고 다른 하나는 데이트를 파일 레지스터에서 W 레지스터로 옮기는(MOVF) 것이다. MOVWF 명령은 또한 NOP으로도 사용한다. 이 NOP명령은 아무런 기능도 하지 않으면서 1기계 사이클만 소비한다.

3.2.2 산술 연산

PIC 명령어 세트는 6개의 산술명령을 제공한다 : 가산(ADDWF), 감산(SUBWF), 증가(INCF), 감소(DECF) 증가 또는 감소후 그 값이 0이면 스킵(INCFSZ/DECFSZ). 각 연산의 결과는 W레지스터로 갈수도 있다. 모든 산술연산은 STATUS 레지스터에 영향을 준다. 또 감산 명령인 SUBWF 명령을 실행하고 STATUS 레지스터를 조사함으로써 비교 연산을 할 수 있다. INCFSZ 와 DECFSZ 는 루프운용에 보통 사용된다.

3.2.3 논리 연산

PIC 명령어 세트에는 6개의 논리 명령어가 있다 : W레지스터의 내용을 지우는 (CLRW), 파일 레지스터의 내용을 지우는 (CLRF), W 레지스터의 내용과 파일 레지스터의 내용을 AND 연산하는 (ANDWF), W 레지스터의 내용과 파일 레지스터와의 내용을 Exclusive OR 연산하는 (XORWF)와 Inclusive OR 연산하는 (IORWF), 그리고 파일 레지스터 내용을 보수 연산하는 (COMF)의 6개이다.

3.2.4 로테이트 연산

PIC 명령어 세트에는 3개의 로테이트 명령이 있다. 이들 명령어들은 모든 파일 레지스터에 있는 데이터를 왼쪽이나 오른쪽으로 회전 시킬수 있다. 이들 명령어들은 2진 곱셈이나 나눗셈 그리고 연속적인 데이터 출력 등을 포함한 넓은 범위의 응용에 사용된다. 특수한 로테이트 명령 으로는 파일 레지스터의 내용의 상위반과 하위반을 맞바꾸는 SWAP 명령이 있다. 이 명령은 데이터를 packing 하거나 unpacking 할때 유용하게 쓰이면서 BCD 연산에 유용된다.

3.3 BIT 단위의 파일 레지스터 운용

이 그룹의 명령들은 모든 어드레스 가능한 파일 레지스터의 개별적인 비트들을 조작하고 test할 수 있다. 이 명령들 역시 직접 또는 간접의 어드레스 모드를 사용할 수 있다.

비트단위 파일 레지스터 명령의 형식은 :

(11-8)	(7-5)	(4-0)
OPCODE	b	f

f = 파일 레지스터 번지

b = 비트 번호

명령은 심볼로서도 나타낼 수 있다.

OPCODE f, b

여기서 f와 b는 2진, 8진,16진, 10진, 또는 심볼로써 표시할 수 있다.

3.3.1 비트 조작

PIC 명령어에는 파일 레지스터의 개별 비트를 조작하는 2개의 명령어가 있는데 그 중 하나는 비트를 클리어 하는 (BCF) 명령이고 다른 하나는 비트를 세트시키는 (BSF) 명령이다.

3.3.2 비트 TEST에 의한 조건적인 스킵 명령

PIC 명령어에는 개별적인 비트를 TEST하기 위한 2개의 명령어가 있다. 그 하나는 비트 TEST후 그 비트가 클리어(0) 이면 스킵하는 (BTFSC) 이고 다른 하나는 비트 TEST후 그 비트가 세트(1)이면 스킵하는 (BTFSS) 명령이다. 이들 명령들은 STATUS와 플래그 비트들을 쉽게 조사해 볼 수 있고 그 결과에 의하여 프로그램의 다른 위치로 이동할 수 있다.

3.4 상수 / 제어 연산

이 그룹의 명령들은 프로그램 메모리 내의 상수에 대해 동작하거나 프로그램 메모리 내에 있는 CALL명령을 하거나 분기를 위하여 사용된다.

동작은 상수명령들을 사용 함으로써 수행된다 :

- 상수 값을 W 레지스터로 옮긴다.
- 상수를 논리 연산한다.

또한 동작은 제어 명령에 의해서도 이루어 진다 :

- Jump 명령
- Call과 Return 명령

상수와 제어 명령은 immediate 어드레싱을 사용한다. 명령 워드는 8 또는 9 비트의 상수로 이루어진 OPCODE로 이루어져 있다. 이 상수들은 산술이나 논리 연산에서의 오퍼랜드로 사용된다.

3.4.1 상수 연산

4 개의 상수 명령어가 PIC 명령어중에 있다. 그 중 하나는 상수를 W 레지스터에 옮기는 (MOVLW) 명령이고 다른 3명령은 (IORLW, XORLW, ANDLW) 상수와 W 레지스터간에 논리 연산을 수행한다.

상수 명령의 형식은 다음과 같다 :

11-8	7-0
OPCODE	k

명령은 다음과 같이 심볼로 표시할 수도 있다 ;

OPCODE k 여기서 k는 8진, 2진, 16진, 10진, 또는 심볼 표시이다.

3.4.2 제어 연산

4개의 제어 명령이 jump, call, return 을 위하여 PIC 명령어 세트에 있다. 그중 하나는 무조건적인 jump(분기)인 GOTO명령이고 분기될 명령번지는 프로그램 카운터로 로드 .

call 과 return 명령은 서브루틴 호출과 메인 프로그램으로의 복귀를 위하여 사용된다. CALL 명령은 호출될 서브루틴의 번지가 PC로 로드되기 이전에 CALL 명령의 다음 번지(PC+1)를 스택에 넣는다. RETLW 명령은 스택에 있는 맨위의 번지를 PC로 넣고 지정된 상수값을 W 레지스터로 로드한다.

3.5 목적지로 지정될 때의 STATUS (F3) 레지스터

STATUS 레지스터(F3)의 내용을 바꿀때는 주의를 요한다. 비록 F3 이 파일 레지스터로 값을 써넣는 명령의 목적지가 된다 하더라도 STATUS 레지스터의 Z, DC, C 비트들은 그 명령이 수행된후 다시 조정된다. 또한 F3의 'TO와'PD' 비트(CMOS PIC 에서만)들은 읽을 수만 있으며 을 써넣을 수는 없다. 그러므로 F3 을 목적지로 하는 명령의 결과는 의도했던 것과는 다를수 있다.

예를들어 CLRF F3 명령은 F3에 'TO'와 'PD'비트를 제외하고 '0'을 로드할 것이고 'Z'비트(비트2)를 세트시켜 그 결과는 : 00XX100 이 된다.

그러므로 다음과 같은 명령들은 STATUS 비트에 영향을 줄 수 없으므로 STATUS 레지스터를 바꾸는데에 BCF, BSF 와 MOVWF 같은 명령의 사용을 권한다.

3.6 Opcode Map 과 사용되지 않은 Opcode들

그림 3.6.1 은 PIC16C5X 의 OPCODE 를 표시한 것이다. 사용되지 않은 OPCODE들은 다음과 같다 :

001 : NOP

008 - 01F : NOP

041 - 05F3 : CLRW

Microchip은 사전통보 없이 사용치 않는 opcode 들의 기능을 바꿀수 있는 권리가 있읍니다.

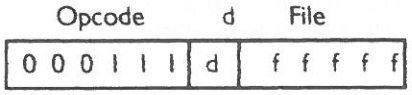
OPCODE <7:5> →

OPCODE <11:8>	0	1	2	3	4	5	6	7
0	참고1	MOVWF	040 CLRW 041-05F Unused	CLRF	SUBWF	SUBWF	DECF	DECF
1	IORWF	ANDWF	ANDWF	ANDWF	XORWF	XORWF	ADDWF	ADDWF
2	MOVF	MOVF	COMF	COMF	COMF	INCF	DECFSZ	DECFSZ
3	RRF	RLF	RLF	RLF	RLF	SWAPF	INCFSZ	INCFSZ
4	BCF							
5	BSF							
6	BTFSC							
7	BTFSS							
8	RETLW							
9	CALL							
A	GOTO							
B	GOTO							
C	MOVLW							
D	IORLW							
E	ANDLW							
F	XORLW							

참고 1 : 000 NOP 005 TRIS 5
002 OPTION 006 TRIS 6
003 SLEEP 007 TRIS 7
004 CLRWDT 001,008-01F Unused

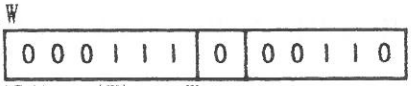
그림 3.6.1 PIC16C5X OPCODE MAP

ADDWF F,d	W 레지스터의 내용과 파일 레지스터의 내용을 가산한다.	ADDWF F,d



(f) + W → d 영향받는 status bit: C, DC, Z

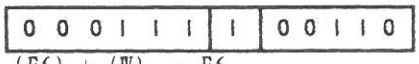
예 : ADDWF 6,W



(F6) + (W) → W

W 레지스터의 내용은 파일 레지스터의 내용과 더해지고 결과치는 W 래지스터로 가며(d=6), F6의 내용은 변하지 않는다.

예 : ADDWF 6



(F6) + (W) → F6

W 레지스터의 내용은 파일 레지스터의 내용과 더해지고 결과치는 F6로 가며(d=1), W 레지스터는 변하지 않는다.

f6 에 A2가 있고 W 레지스터에 4F 가 있다고 가정하여 ADDWF 명령을 수행하면 STATUS 레지스터의 비트들은 아래와 같이 영향을 받는다 :

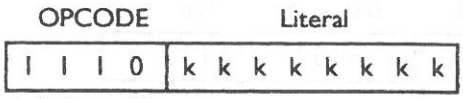
```
  1 0 1 0 0 0 1 0
+ 0 1 0 0 1 1 1 1
-----------------
  1 1 1 1 0 0 0 1   → F6
```

STATUS

C	DC	Z
0	1	0

위의 표에서 carry 비트가 리셋되어 있는것은 연산에서 오버플로우가 없음을 나타낸다(W 레지스터와 파일 레지스터간의 가산결과는 255). Digit Carry 비트는 digit 자리넘침이 생겼으므로 세트된다(W 레지스터와 파일 레지스터의 하위 4 비트의 연산이 15 보다 크다). Zero 비트가 리셋되어 있는것은 가산 결과가 0 이 아님을 나타낸다.

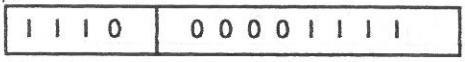
ANDLW k	상수와 W 레지스터를 AND 연산 (비트 대 비트) 한다.	ANDLW k



k & W → W

Status 비트의 변화 : Z

예 : ANDWF 17,W

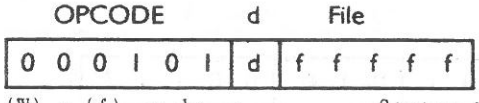


0F & W → W

W 레지스터의 상위 4 비트는 상수의 상위 4 비트와의 AND 연산에 의해 0 이 되고 하위 4 비트는 상수의 하위 4 비트가 1이므로 변화되지 않는다.

W 레지스터의 내용이 ANDLW 0F 명령이 실행되기전 01001001 이었다고 가정하자. 이때 ANDLW 0F 명령이 실행되면 W 레지스터의 값은 00001001 이 된다.

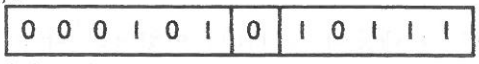
ANDWF f,d	W 레지스터와 파일 레지스터와 AND 연산한다.	ANDWF f,d



(W) • (f) → dn

Status 비트의 변화 : Z

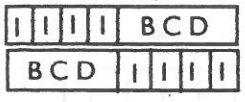
예 : ANDWF 17,W



(W) • (F17) → W

W 레지스터의 내용은 파일 레지스터 17 의 내용과 AND 연산되고 결과치는 W 레지스터로 간다(d=0). 그리고 f17 의 내용은 변하지 않는다.

BCD 라고 표시된 데이터를 하나의 레지스터로 모은다고 하자, W 레지스터의 상위 비트가 1로 채워져 있고 F17의 하위 비트들도 1로 되어있다. 그러면 두 레지스터를 AND 연산하여 :

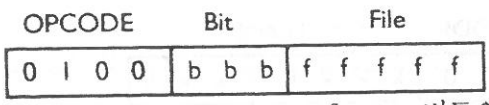


W • F17

결과는 :

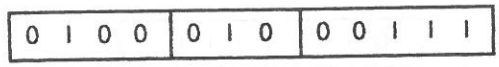
→ W

BCF f,b 파일 레지스터의 특정비트를 클리어 한다. **BCF f,b**



0→f(b) Status 비트의 영향 : 없음

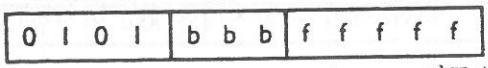
예 : BCF 7,2



0→F7(2)

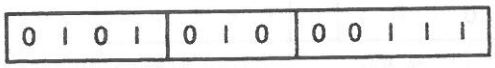
BCF 명령이 실행되기 전에 F7 의 내용이 11111111 이었다고 하자 BCF 명령이 실행되면 F7 의 내용은 11111011 으로 된다.

BSF f,b 파일 레지스터의 특정비트를 세트 시킨다. **BSF f,b**



1→f(b) Status 비트의 영향 : 없음

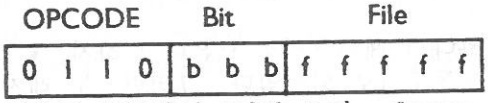
예 : BSF 7,2



0→F7(2)

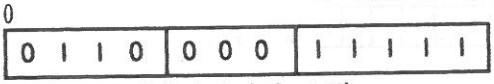
BCF 명령이 실행되기 전에 F7 의 내용이 11111011 이었다고 하자 BSF 명령이 실행되면 F7 의 내용은 11111111 으로 된다.

BTFSC f,b 파일 레지스터의 비트를 TEST한후 클리어이면 스킵한다. **BTFSC f,b**



F(b)를 TEST하여 0이면 스킵 Status 비트의 영향 : 없음

예 : BTFSC 01F,0



F 1F를 TEST하여 0이면 스킵

파일 레지스터 1F 의 0번 비트를 TEST하여 만일 그 값이 0 이면 다음 명령은 스킵한다.

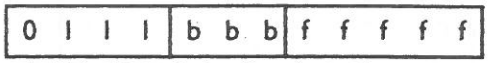
파일 1F 의 0번 비트가 오버플로우 비트라고 하면 프로그램은 아래와 같이 코딩되어야 할 것이다:

```
BTFSC 01F,0
INCF  10
GOTO  SCAN
```

만일 오버플로우가 있으면 F10 은 SCAN 루틴으로 가기전에 하나 증가할 것이다. 그러나 오버플로우가 없으면 F10 은 증가되지 않는다.

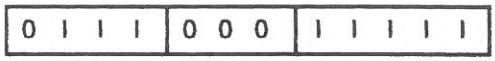
BTFSS f,b	파일 레지스터의 비트를 TEST하여 세트이면 스킵한다.	BTFSS f,b

OPCODE Bit File



F(b)를 TEST하여 세트면 스킵, Status 비트의 변화 : 없음

예 : BTFSS 7,1



F 7(1)를 TEST하여 세트면 스킵

파일 레지스터 7 을 시험하여 값이 1 이면 다음 명령은 스킵한다.

F7 의 1번 비트가 입력 플래그 비트라고 하면 코딩은 아래와 같이 될것이다 :

```
BTFSS 7,1
GOTO  CALC
GOTO  INPUT
```

만일 1번 비트가 세트면 프로그램은 INPUT 루틴으로 점프한다. 그러나 클리어면 프로그램은 CALC 루틴으로 점프한다.

CALL k	k 어드레스에 있는 서브루틴을 호출한다.	CALL k

OPCODE Address

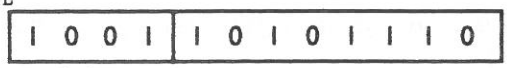
(PC)+1→STACK

k→PC , STATUS 비트의 변화 : 없음

이 명령은 PC 의 값을 하나 증가 시켜 그 증가된 값(PC+1)을 스택에 넣고 지정된 서브루틴 번지를 PC에 넣는다. 그러면 프로그램은 이 위치에서부터 실행된다.

참고 : 0FF 까지의 모든 명령번지들은 8 비트의 2진수로 표시된다(0FF=11111111). 0FF 를 넘는 번지들은 9비트가 필요한데 프로그램 카운터의 9번째 비트는 CALL 명령에서 항상 0으로 리셋되기 때문에 **서브루틴은 프로그램 메모리 000-0FF 이내에 있어야한다**. 그러나 서브루틴은 스택이 9비트 이기 때문에 모든 프로그램 메모리로 부터 호출될 수 있다. 16C56과 16C57 에서 STATUS 레지스터의 PA2:PA0 비트는 PC 의 <11:9>비트로 로드된다. 그러므로 16C56 과 16C57에서 서브루틴은 512 워드 페이지의 처음 256워드에만 존재하여야 한다.

예 : CALL 0AE

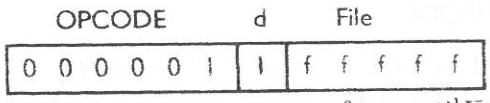


프로그램이 1F0에 있다고 하자 :

1F0+1→STACK

AE→PC

CLRF f — 파일 레지스터의 내용을 클리어 한다. — **CLRF f**

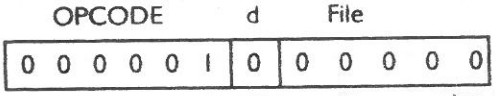


0→f　　Status 비트의 변화 : Z

예 : CLRF 0F

0→F0F

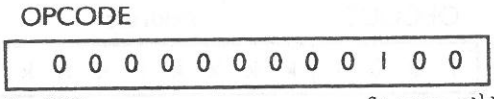
CLRW — W 레지스터의 내용을 클리어 한다. — **CLRW**



0→W　　Status 비트의 변화 : Z

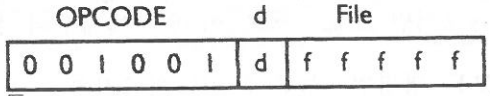
CLRWDT — Watchdog 타이머를 클리어 한다. — **CLRWDT**

Watchdog 타이머를 클리어 한다. 이때 만일 프리스케일러가 WDT 에 할당 되었다면 프리스케일러도 클리어 된다.



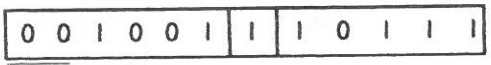
0→WDT　　Status 비트의 변화 : TO, PD

COMF f,d — 파일 레지스터의 내용에 보수를 취한다. — **COMF f,d**



$(\overline{F})$→d　　Status 비트의 변화 : Z

예 : COMF 17

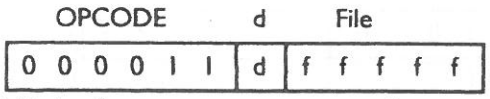


$(\overline{F17})$ →F17

파일 레지스터 17의 내용은 보수로 바뀌었다. d=1 이므로 결과치는 F17로 가고 W 레지스터의 내용은 변화되지 않는다.

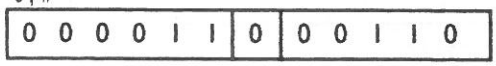
COMF 명령이 실행되기전 F17 의 내용이 01110110 이었다고 가정하자 여기서 COMF 명령이 실행되면 F17 의 내용은 10001001 이 된다.

DECF f,d — 파일 레지스터의 내용을 감소 시킨다. — **DECF f,d**



(f)-1→d　　Status 비트의 변화 : Z

예 : DECF 6,W

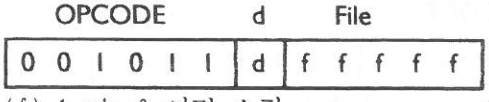


(F6)-1→W

파일 레지스터6의 내용은 감소되고 결과치는 W 레지스터로 간다. F6 의 내용은 바뀌지 않는다.

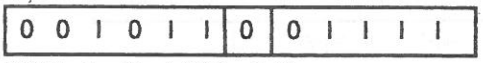
F6 의내용이 DECF 명령 이전에 00000001 이었다고 하자 명령이 실행되면 내용은 감소되어 결과치는 00000000 이 되고 STATUS 의 Z 비트는 세트된다.

DECFSZ f,d — 파일 레지스터의 내용을 감소시킨후 0이면 스킵한다. — **DECFSZ f,d**



(f)-1→d　0 이면 스킵　　Status 비트의 변화 : 없음

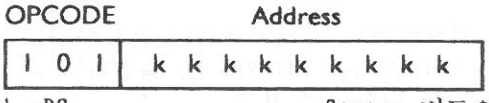
예 : DECFSZ 0F,W



(F0F)-1→W, 0이면 스킵

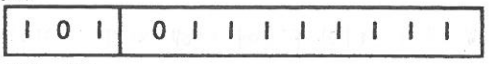
이 명령은 loop 카운트의 2의 보수라기 보다는 실제적 loop 레지스터라는 점을 제외하면 INCFSZ 명령과 비슷하다. 마지막 loop 카운트에서 레지스터의 값이 0이 되면 다음 명령으로 스킵한다.

GOTO k — k 번지로 분기한다. (주의:이 명령에서 k 는 9비트이다.) — **GOTO k**



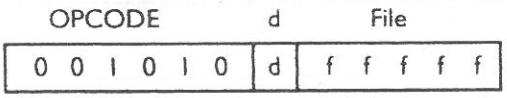
k→PC　　Status 비트의 변화 : 없음

예 : GOTO 0FF



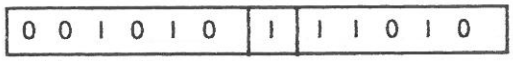
0FF→PC

INCF f,d 파일 레지스터 내용을 증가 시킨다. **INCF f,d**



(f)+1→f Status 비트의 변화 : Z

예 : INCF 1A

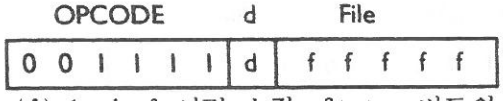


(F1A)+1→F1A

파일 레지스터 1A 의 내용을 증가 시키고 결과치는 F1A로 간다(d=1). W 레지스터는 아무런 영향을 받지 않는다.

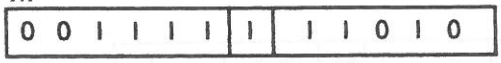
명령 INCF가 실행되기전 F1A 의 내용이 01010111 이었다고 하자 이제 INCF 1A 가 실행되면 F1A 의 내용은 01011000 로 된다.

INCFSZ f,d 파일 레지스터의 내용을 증가시킨후 0이면 스킵한다. **INCFSZ f,d**



(f)+1→d, 0 이면 스킵, Status 비트의 변화 : 없음

예 : INCFSZ 1A



(F1A)+1→F1A, 0 이면 스킵

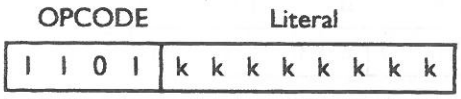
아래에는 연속적인 7개의 테이블을 순차적으로 수정 시키는 프로그램으로서 간접 어드레스에 의해 파일 선택 레지스터 F4 는 테이블의 시작번지를 로드하고 루프 통과횟수에 대한 2의 보수는 F1A 에 로드된다.

테이블 개정동작을 제외하고 2개의 동작이 loop 에서 수행된다 : (1)테이블 번지가 증가된다.; (2)통과 횟수가 증가된다.

```
        MOVLW     TABLE   ;  시작번지를 W 레지스터에 로드한다.
        MOVWF     4       ;  시작 번지를 F4(FSR) 로 옮긴다.
        MOVLW     F9      ;  7 에 대한 2의 보수.
        MOVWF     1A      ;  통과 횟수를 F1A 에 넣는다.
        --------------
        --------------
LOOP    ADDWF 0           ;  W 래지스터와 FSR에 의해 지정된 위치의 테이블과 더한다.


        --------------
        --------------
        INCF        4     ;  테이블 번지를 증가 시킨다.
        INCFSZ      1A    ;  카운터를  증가시키고 0 이면 스킵한다.
        GOTO        LOOP
        EXIT
```

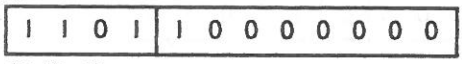
마지막 7번째 loop 에서 F1A 는 증가하여 0이 되고 GOTO LOOP 명령은 스킵한다.

IORLW k	W 레지스터와 상수와의 비트단위의 Inclusive OR 연산을 한다	IORLW k



k∨W→W　　Status 비트의 변화 : Z

예 : IORLW 80

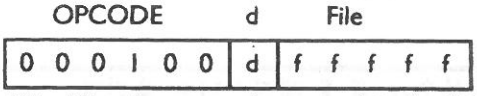


80∨W→W

산술연산을 할때 sign 비트를 양에서 음으로 바꾸어야할 필요가 있다고 가정하자 W 레지스터의 내용과 80(10000000)과 Inclusive OR 를 하면 sign 비트(MSB)가 1이된다(음의부호).

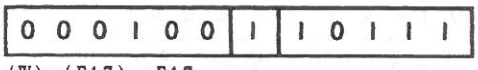
IORLW 80 명령이 실행되기전 W 레지스터값이 01101110 이었다고 하면 IORLW 80 명령을 수행한후에 W 레지스터의 값은 11101110 이 된다.

IORWF f,d	W 레지스터와 파일 레지스터와의 Inclusive OR 연산을 한다.	IORWF f,d



(W)∨(f)→d　　Status 비트의 변화 : Z

예 : IORWF 17



(W)∨(F17)→F17

파일 레지스터 17의 내용은 W 레지스터와 OR 연산된다. 결과치는 F17로 간다(d=1). W 레지스터의 내용은 변화없다.

두 바이트로 나뉘어 있는 BCD 데이터를 하나의 레지스터로 모아야 한다고 가정하자. W 레지스터의 상위 비트와 F27 의 하위 비트들은 0으로 되어있다. 이제 두 레지스터를 OR 연산하면 데이터를 하나의 레지스터로 모을 수 있다.

MOVF f,d — 파일 레지스터의 내용을 이동 시킨다. — **MOVF f,d**

OPCODE	d	File
0 0 1 0 0 0	d	f f f f f

(f)→d　　Status 비트의 변화 : Z

예 : MOVF 12,W

0 0 1 0 0 0	0	1 0 0 1 0

(F12)→W

파일 레지스터 12의 내용은 W 레지스터로 옮겨지고 만일 레지스터의 내용이 0 이면 Status 레지스터의 Zero 비트는 세트된다.

예 ; MOVF 12

0 0 1 0 0 0	1	1 0 0 1 0

(F12)→F12

파일 레지스터 12 의 내용은 ALU로 이동했다가 다시 파일 12로 돌아온다. 이 명령은 파일 레지스터의 내용을 조사하기 위해 사용할 수 있으며 만일 레지스터의 내용이 0 이라면 Status 의 Zero 비트는 세트된다.

MOVLW k — 상수 k 를 W 레지스터로 옮긴다 — **MOVLW k**

OPCODE	Literal
1 1 0 0	k k k k k k k k

k→W　　Status 비트의 변화 : 없음

부호화된 8비트상수는 W 레지스터로 이동한다.

1 1 0 0	k k k k k k k k

0FF→W

MOVWF f — W 레지스터의 내용을 파일 레지스터로 옮긴다. — **MOVWF f**

OPCODE	d	File
0 0 0 0 0 0	1	f f f f f

(W)→f　　Status 비트의 변화 : 없음

예 : MOVWF 11

0 0 0 0 0 0	1	1 0 0 0 1

(W)→F11

W 레지스터의 내용은 파일 레지스터 11 로 옮겨진다.

NOP	아무런 실행도 하지 않는다.	NOP

OPCODE

0 0 0 0 0 0 0 0 0 0 0 0

Status 비트의 변화 : 없음

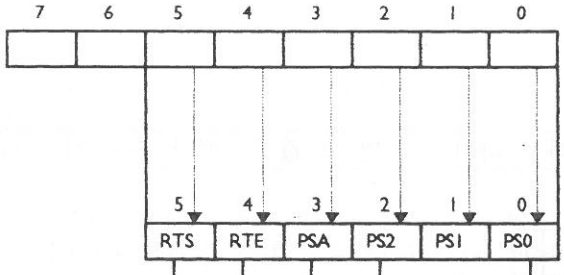
OPTION	옵션 레지스터에 값을 로드 시킨다.	OPTION

W 레지스터의 하위 6비트 내용은 OPTION 레지스터로 로드된다.

OPCODE

0 0 0 0 0 0 0 0 0 0 1 0

W→옵션 레지스터, Status 비트의 변화 : 없음



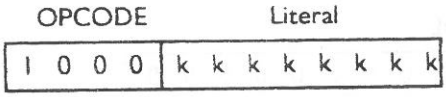
프리스케일러 값	RTCC 분주비	WDT 분주비
0 0 0	1 : 2	1 : 1
0 0 1	1 : 4	1 : 2
0 1 0	1 : 8	1 : 4
0 1 1	1 : 16	1 : 8
1 0 0	1 : 32	1 : 16
1 0 1	1 : 64	1 : 32
1 1 0	1 : 128	1 : 64
1 1 1	1 : 256	1 : 128

프리스케일러 지정 비트:
0 ... RTCC
1 ... WDT

RTCC 카운트 시점:
0 ... RTCC핀의 신호가 LOW에서 HIGH로 전환될때 증가
1 ... RTCC핀의 신호가 HIGH에서 LOW로 전환될때 증가

RTCC 신호원:
0 ... 내부의 명령 수행 클럭 (=CLKOUT)
1 ... RTCC 핀

RETLW k	메인 프로그램으로 복귀하며 W 레지스터에 상수 K를 넣는다.	RETLW k



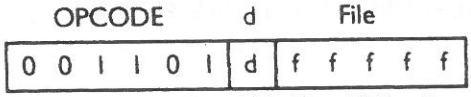
k→W

스택→PC

Status 비트의 영향 : 없음

이 명령은 서브루틴의 끝에서 CALL 명령 다음으로 즉각 복귀하기 위해 사용된다. 상위 스택의 내용은 PC 로 간다. 명령에 포함된 상수값 k 는 W 레지스터로 간다.

RLF f,d	파일 레지터의 내용을 좌로 이동한다.	RLF f,d



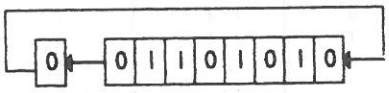
C→d(0), f(6-0)→d(7-1), f(7)→C

Status 비트의 영향 : C

예 : RLF 10

지금 파일 레지스터 10에 값이 저장되어 있다고 하자 그러면 그 값이 2배가 되고 하면 CARRY 비트는 리셋 된다 :

회전하기전



회전후

파일 레지스터 10에 있던 값은 35에서 두배로 증가된 6A가 되었다.

RRF f,d	파일 레지스터의 내용을 오른쪽으로 이동한다.	RRF f,d

OPCODE d File

0	0	1	1	0	0	d	f	f	f	f	f

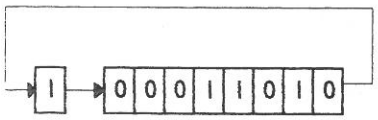
C→d(7), f(7-1)→d(6-0), f(0)→C

Status 비트의 영향 : C

예 : RRF 10

파일 레지스터 10의 내용이 오른쪽으로 이동 하여야 한다고 하면 CARRY 비트와 연결되어 :

회전하기전



회전후

CARRY 비트는 테스트 가능하며 내용은 각 회전명령 후에 출력된다.

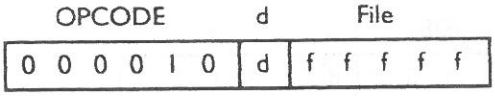
SLEEP	슬립 모드로 들어간다.	SLEEP

이 명령으로 칩은 SLEEP 모드로 들어간다. 슬립 명령이 실행되면 WDT 는 클리어 되고 오실레이터는 꺼진다. 또 F3(Status 레지스터의) "PD" 비트는 클리어 되고 "TO" 비트는 세트된다. 그렇지만 WDT는 칩 내부에있는 자신의 클록으로 계속 작동한다.

OPCODE

0	0	0	0	0	0	0	0	0	0	0	1	1

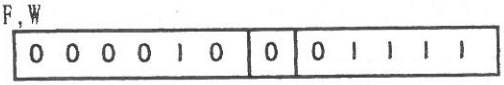
0→WDT Status 비트의 영향 : TO, PD



(f) - (W) ⊛ d[(F) + $(\overline{W})$ + 1 ⊛ d]

Status 비트의 영향 : C, DC, Z

예 : SUBWF 0F,W



(F0F)-(W)→W

W 레지스터의 내용은 파일 레지스터 0F 의 내용으로 부터 감산된다(2의 보수 덧셈을 이용). 결과치는 W 레지스터로 간다(d=0). F0F의 내용은 변하지 않는다.

F0F에 44가 들어있고 W 레지스터에 28이 들어있다고 가정하여 SUBWF 명령이 Status 비트에 미치는 영향을 살펴보면 :

```
  01000100
- 00101000
----------
  00011100    →W
```

STATUS

C	DC	Z
1	0	0

CARRY 비트가 세트되어 있는것은 오버플로우가 없음을 나타낸다(W 레지스터에 있는 절대값이 F0F의 절대값 보다 작다). DIGIT CARRY 비트가 리셋되어있는 것은 'DIGIT 오버플로우'가 있다는 것을 나타낸다(F0F의 하위 4비트의 절대값은 W 레지스터의 하위 4비트 절대값 보다 크다).

F0F 의 값이 28 이고 W 레지스터의 값이 44 라고 하면 :

```
  00101000
- 01000100
----------
  11100100    →W
```

STATUS

C	DC	Z
0	1	0

CARRY 비트가 리셋되어 있는것은 오버플로우가 있음을 나타낸다(W 레지스에 있는 절대값이 F0F의 절대값 보다 크다). DIGIT CARRY 가 세트되어 있는것은 'DIGIT 오버플로우'가 없음을 나타낸다.

작은 수에서 큰 수를 뺄때는 그 차이값의 2의 보수가 결과치가 된다. 이 경우 E4는 두 수의 차이(1C, 00011100)의 보수로 부터 얻어졌다. 따라서 C=0 은 음의 결과치를 의미하는 2의 보수 형태이다. F07에 28이 들어있고 W 레지스터에도 28이 들어있다고 하면 :

```
  00101000
- 00101000
----------
  00000000    →W
```

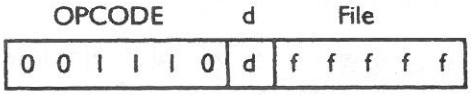
STATUS

C	DC	Z
1	1	0

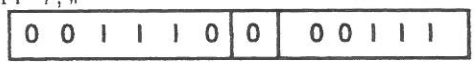
SUBWF 명령은 2개의 값을 비교할 수도 있다. 하나의 값이 W 레지스터에 있고 다른 값이 파일 레지스터에 있을때 SUBWF 명령이 사용되면 W 에 있는 값이 파일 레지스터에 있는 값보다 큰지, 작은지, 같은지를 Status를 검사함 으로써 알 수 있다 :

Condition	True	False
W>F	C=0	C=1
W<F	C=1	C=0
W=F	Z=1	Z=0

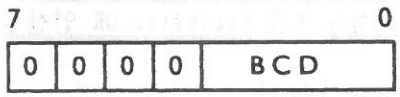
SWAPF f,d 파일 레지스터의 상,하위 반을 서로 맞바꾼다. **SWAPF f,d**



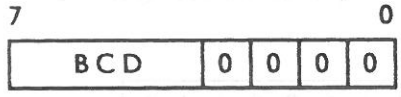
예 : SWAPF 7,W



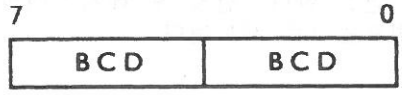
각각의 레지스터에 BCD로 표시된 데이터가 하나의 레지스터로 모아져야 한다고 가정하자. F7의 하위 4비트에 BCD 데이터가 있고 나머지는 0 으로 채워져있고 F8도 그렇게 되어있다 :



명령 SWAPF 7,W 는 BCD와 0값을 서로 맞바꿔치고 결과값을 W 레지스터로 보낸다 :



이제 IORWF 8 명령을 이용하여 W 레지스터와 F8 레지스터와 OR 연산을 하면 두 BCD 데이터는 F8로 모아진다.



TRIS f — 포트의 입,출력 상태를 설정한다. — TRIS f

I/O 포트(A,B,C)의 모드를 정의한다.

OPCODE										Operand		
0	0	0	0	0	0	0	0	0	0	f	f	f

W→I/O 제어 레지스터 f

Status 비트의 영향 : 없음

I/O 제어 레지스터는 W 레지스터의 내용을 로드한다.

TRIS 명령은 I/O 포트를 입력 또는 출력으로 쓸 것인지를 결정하며 "f"에 들어갈 값은 16C54/16C56 의 경우에는 4,5가 되고 16C55/16C57 의 경우에는 4,5,6이 된다.

예 :

```
MOVLW   0F
TRIS    6
```

I/O 포트 B 는 위의 프로그램으로 인하여 하위 4비트는 입력 상위 4비트는 출력으로 정의된다.

XORLW k — W 레지스터와 상수 k를 Exclusive OR 연산한다. — XORLW k

OPCODE				Literal							
1	1	1	1	k	k	k	k	k	k	k	k

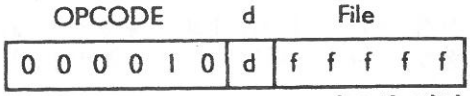
k⊕W→W, Status 비트의 영향 : Z

예 : XORLW C7

1	1	1	1	1	1	0	0	0	1	1	1

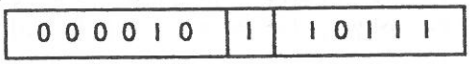
C7⊕W→W

C7은 W 레지스터와 Exclusive OR 연산을 한다. 만일 두 레지스터의 내용이 같다면 결과치는 0이되고 STATUS 레지스터의 ZERO 비트는 1로 세트된다.



(W)⊕(f)→d,　　Status 비트의 변화 : Z

예 : XORWF 17



(W)⊕(F17)→F17

W 레지스터와 파일 레지스터와 Exclusive OR 연산을 하고 결과값은 파일 레지스터(F17)로 간다(d=1). W 레지스터의 내용은 변화하지 않는다.

W 레지스터의 내용과 파일 레지스터의 내용과 비교 해야 한다고 가정하자. 만일 그 값이 서로 같다면 결과는 0이 되고 Status 레지스터의 ZERO 비트는 세트된다.

4 장 : 특수 명령 니모닉

Status 비트 테스터에 의한 조건적 스킵과 분기, 2의 보수연산, carry 와 digit carry 덧셈 등과 같이 빈번히 사용되는 명령들은 파일, 비트 ,상수, 제어명령들의 조합으로 이루어진 특수 오퍼랜드 에 의해 수행될 수 있다.

이 연산들은 PIC 어셈블러에 의해 인정되는 특수 니모닉 으로써 실행된다. 이 니모닉들은 추가적인 명령어 워드를 의미 하는것은 아니다. 각각의 이 특수 명령어들은 하나 혹은 그 이상의 PIC 명령어 들을 불러쓴다. 어셈블러는 특정 위치나 목적지를 위한 적절한 오퍼랜드를 끼워넣는다.

특수명령 니모닉들은 아래의 연산들을 수행한다.

- 파일의 데이터를 W 레지스터로 이동.
- 파일 테스트.
- 파일 레지스터의 내용에 대한 2의 보수 연산.
- 무 조건적 분기.
- 6개의 Status 비트 조작.
- 6개의 Status 비트 시험에 의한 조건적 스킵.
- 6개의 Status 비트 시험에 의한 조건적 분기.
- 4개의 Carry, Digit carry 대수 연산.

4.1.1 파일로부터 W 레지스터로 데이터 이동

특수명령 니모닉으로 파일의 내용을 W 레재스터로 옮길 수 있다.

MOVFW f — **파일의 내용을 W 레지스터로 옮긴다.** — **MOVFW f**

OPCODE	d	File
0 0 1 0 0 0	0	f f f f f

(f)→W　　Status 비트의 변화 : Zero

동일 파일 연산 :　　MOVF f,0

4.1.2 파일 테스트

특수명령 니모닉으로 파일 레지스터의 내용이 0 인지를 알아볼 수 있다. 이 명령은 파일 레지스터의 내용을 스스로에게 덧씌우는데 이 과정에서 Status의 Zero 비트의 변화에 의해 파일 레지스터의 내용을 시험해볼 수 있다.

TSTF f — **파일 레지스터의 내용을 TEST 한다.** — **TSTF f**

OPCODE	d	File
0 0 1 0 0 0	1	f f f f f

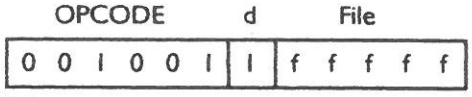
(f)→1　　Status 비트의 변화 : Zero

동일 파일 연산 : MOVF f,1

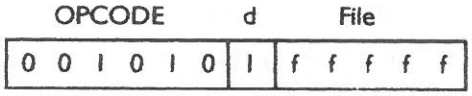
4.1.3 레지스터 내용의 2의보수 연산

특수 명령 니모닉으로 파일 레지스터의 내용에 대한 2의보수 연산을 할 수 있다. 이 니모닉은 2개의 명령을 사용한다. 첫째 명령은 파일의 내용을 보수 연산하고 둘째 명령은 LSB 에 2진수 1을 더한다.

NEGF f,d	파일 레지스터의 내용에 대한 2의보수 연산을 한다.	NEGF f,d



$\overline{(f)}$ ® d Status 비트의 변화 : Zero



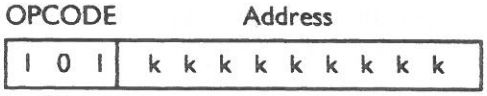
(f)+1→d Status 비트의 변화 : Zero

동일 파일 명령 : COMF f,1
INCF f,d

4.1.4 무조건적인 분기

특수 명령 니모닉으로 무조건적 분기 명령을 실행할 수 있다.

B k	k 번지(9비트 번지)로 분기 한다.	B k



k→PC

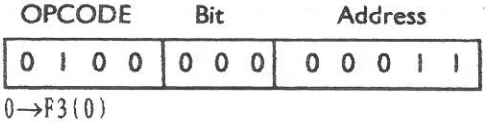
9비트의 번지가 PC로 로드되고 프로그램은 그 번지로 점프한다.

동일 제어 명령 : GOTO k

4.1.5 Status 비트 조작

Status의 Carry, Digit carry, Zero 비트들을 세트 시키거나 리셋 시키기위해 6개의 특수 명령 니모닉이 있다.

CLRC	Carry 비트를 클리어 한다.	CLRC



0→F3(0)

Status 레지스터의 0비트(Carry 비트)는 클리어 된다.

동일 비트 명령 : BCF 3,0

CLRDC — Status의 Digit carry 비트를 클리어 한다. — **CLRDC**

OPCODE	Bit	File
0 1 0 0	0 0 0	0 0 0 1 1

0→F3(1)

Status 레지스터의 1번 비트(Digit carry 비트)가 클리어 된다.

동일 비트 명령 : BCF 3,1

CLRZ — Status의 Zero 비트를 클리어 한다. — **CLRZ**

OPCODE	Bit	File
0 1 0 0	0 1 0	0 0 0 1 1

0→F3(2)

Status 레지스터의 2번 비트(Zero 비트)가 클리어 된다.

동일 비트 명령 : BCF 3,2

SETC — Status의 carry 비트를 클리어 한다. — **SETC**

OPCODE	Bit	File
0 1 0 1	0 0 0	0 0 0 1 1

0→F3(2)

Status 레지스터의 0번 비트(Carry 비트)가 세트된다.

동일 비트 명령 : BSF 3,0

SETDC — Status의 Digit carry 비트를 클리어 한다. — **SETDC**

OPCODE	Bit	File
0 1 0 1	0 0 1	0 0 0 1 1

1→F3(1)

Status 레지스터의 1번 비트(Digit Carry 비트)가 세트된다.

동일 비트 명령 : BSF 3,1

SETZ — Status의 Zero 비트를 세트 한다. — **SETZ**

OPCODE	Bit	File
0 1 0 1	0 0 1	0 0 0 1 1

1→F3(1)

Status 레지스터의 2번 비트(Zero 비트)가 세트된다.

동일 비트 명령 : BSF 3,2

4.1.6 Status 비트에 의한 조건적 스킵

Status 비트 테스트에 의한 조건적인 스킵을 위해 6개의 특수명령 니모닉이 있다.

SKPC — Status의 Carry 비트 TEST후 세트이면 다음 명령을 스킵한다. — **SKPC**

OPCODE	Bit	File
0 1 1 1	0 0 0	0 0 0 1 1

Status의 Carry 비트 TEST후 세트이면 다음 명령 스킵

Status의 0번 비트(Carry 비트)를 TEST후 세트이면 다음 명령은 스킵한다.

동일 비트 명령 : BTFSS 3,0

SKPDC — Status의 Digit Carry 비트 TEST후 세트이면 다음명령을 스킵한다. — **SKPDC**

OPCODE	Bit	File
0 1 1 1	0 0 1	0 0 0 1 1

Status의 Digit Carry 비트 TEST후 세트이면 다음 명령스킵

Status의 1번 비트(Digit Carry 비트)를 TEST후 세트이면 다음 명령은 스킵한다.

동일 비트 명령 : BTFSS 3,1

SKPNC — Status의 Carry 비트 TEST후 클리어면 다음 명령을 스킵한다. — **SKPNC**

OPCODE	Bit	File
0 1 1 0	0 0 0	0 0 0 1 1

Status의 Carry 비트 TEST후 리셋이면 다음명령 스킵

Status의 0번 비트(Carry 비트)를 TEST후 리셋이면 다음 명령은 스킵한다.

동일 비트 명령 : BTFSC 3,0

SKPNDC — Status의 Digit Carry 비트 TEST후 세트이면 다음 명령을 스킵한다. — **SKPNDC**

OPCODE	Bit	File
0 1 1 0	0 0 1	0 0 0 1 1

Status의 Digit Carry 비트 TEST후 리셋이면 다음명령 스킵

Status의 1번 비트(Digit Carry 비트)를 TEST후 리셋이면 다음 명령은 스킵.

동일 비트 명령 : BTFSC 3,1

SKPNZ — Status의 Zero 비트 TEST후 클리어면 다음 명령을 스킵한다. — **SKPNZ**

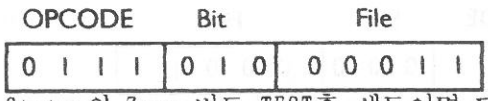
OPCODE	Bit	File
0 1 1 0	0 1 0	0 0 0 1 1

Status의 Zero 비트 TEST후 리셋이면 다음 명령 스킵

Status의 2번 비트 (Zero 비트)를 시험한후 리셋이면 다음 명령은 스킵된다.

동일 비트 명령 : BTFSC 3,2

SKPZ	Status 의 Zero 비트를 TEST후 세트이면 다음명령 스킵	SKPZ



Status의 Zero 비트 TEST후 세트이면 다음명령 스킵

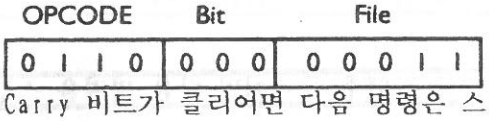
Status의 2번 비트(Zero 비트)를 TEST후 세트이면 다음 명령은 스킵한다.

동일 비트 명령 : BTFSS 3,2

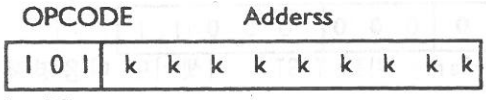
4.1.7 Status 비트에 의한 조건적인 분기

Status 비트 테스트에 의한 조건분기를 위해 6개의 특수명령 니모닉이 있다. 각각의 이 니모닉들은 2개의 명령을 호출하여 사용한다. 첫번째 명령은 Status의 비트를 TEST 한다. 만일 요구조건이 만족되면 두번째 명령은 PC가 지정하는 번지로 가고 만족되지 않으면 다음 명령을 스킵하고 프로그램은 계속된다.

BC k	Carry 비트가 세트이면 k 번지로 분기 한다.	BC k



Carry 비트가 클리어면 다음 명령은 스킵

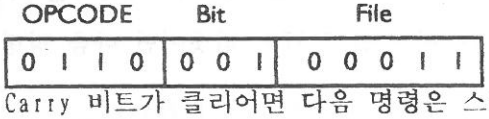


k→PC

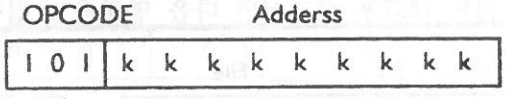
Carry 비트를 TEST하여 0이면 GOTO 명령은 스킵된다. 그러나 1 이면 9비트(k)의 명령 번지가 PC로 가서 그 번지로 점프한다.

동일한 비트와 제어 명령은 : BTFSC 3,0
GOTO k

BDC k	Digit carry 비트가 세트이면 k번지로 분기한다.	BDC k



Carry 비트가 클리어면 다음 명령은 스킵

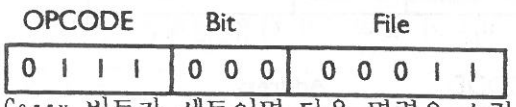


k→PC

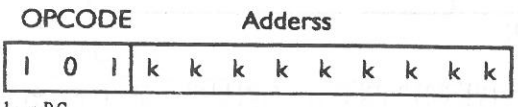
Digit Carry 비트를 TEST하여 0이면 GOTO 명령은 스킵된다. 그러나 1 이면 9비트(k)의 명령 번지가 PC로 가서 그 번지로 점프한다.

동일한 비트와 제어 명령은 : BTFSC 3,1
GOTO k

BNC k — **Carry 비트가 리셋이면 k 번지로 분기한다.** — **BNC k**



Carry 비트가 세트이면 다음 명령은 스킵

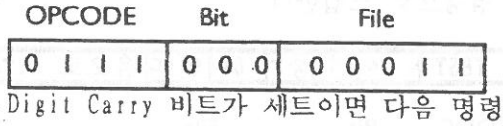


k→PC

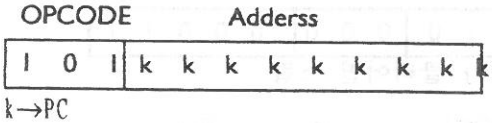
Carry 비트를 TEST하여 1이면 GOTO 명령은 스킵된다. 그러나 0이면 9비트(k)의 명령 번지가 PC로 가서 그 번지로 점프한다.

동일한 비트와 제어 명령은 : BTFSS 3,0
GOTO k

BNDC k — **Digit Carry 비트가 리셋이면 k 번지로 분기한다.** — **BNDC k**



Digit Carry 비트가 세트이면 다음 명령은 스킵

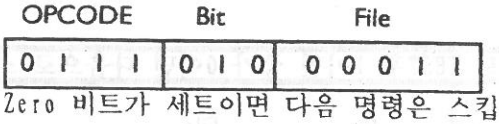


k→PC

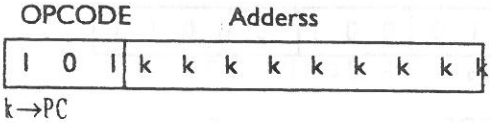
Digit Carry 비트를 TEST하여 1이면 GOTO 명령은 스킵된다. 그러나 0이면 9비트(k)의 명령 번지가 PC로 가서 그 번지로 점프한다.

동일한 비트와 제어 명령은 : BTFSS 3,1
GOTO k

BNZ k — **Zero 비트가 리셋이면 k 번지로 분기한다.** — **BNZ k**



Zero 비트가 세트이면 다음 명령은 스킵

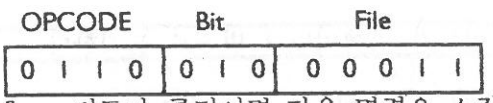


k→PC

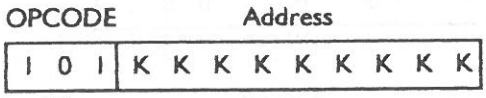
Zero 비트를 TEST하여 1이면 GOTO 명령은 스킵된다. 그러나 0이면 9비트(k)의 명령 번지가 PC로 가서 그 번지로 점프한다.

동일한 비트와 제어 명령은 : BTFSS 3,2
GOTO k

BZ k	Zero 비트가 세트이면 k 번지로 분기한다.	BZ k



Zero 비트가 클리어면 다음 명령은 스킵



k→PC

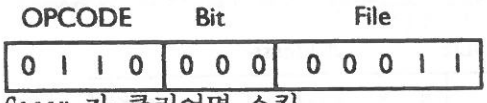
Zero 비트를 TEST하여 0이면 GOTO 명령은 스킵된다. 그러나 1이면 9비트(k)의 명령 번지가 PC로 가서 그 번지로 점프한다.

동일한 비트와 제어 명령은 : BTFSC 3,2
GOTO k

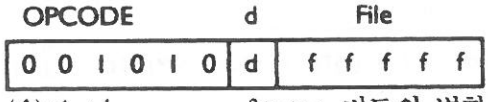
4.1.8 Carry 와 Digit carry 산술연산

파일 레지스터로 부터 Carry 나 Digit carry 비트를 더하거나 빼기 위한 4개의 특수 명령 니모닉이 있다. 각각의 이 명령들은 2개 이상의 명령을 사용한다. 첫번째 명령은 Carry 나 Digit carry 비트를 테스트하는 명령인데 1 이면 두번째 명령이 파일 레지스터를 증가 시키거나 감소 시키고 0 이면 다음 명령으로 스킵한다.

ADDCF f,d	Carry 비트 TEST후 1이면 증가 0이면 다음으로 스킵한다.	ADDCF f,d



Carry 가 클리어면 스킵

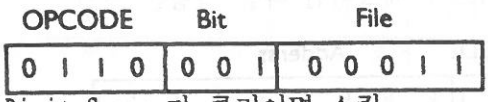


(f)+1→d Status 비트의 변화 : Zero

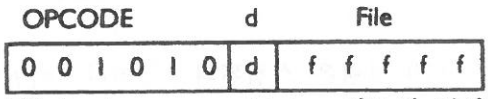
Status 의 Carry 비트가 TEST되어 0이면 증가 명령은 스킵되고 1이면 파일 레지스터는 증가된다.

동일 명령은 : BTFSC 3,0
INCF f,d

ADDDCF f,d	Digit Carry 비트 TEST후 1이면 증가 0이면 다음으로 스킵한다.	ADDDCF f,d



Digit Carry 가 클리어면 스킵

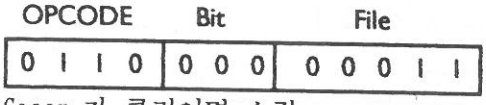


(f)+1→d Status 비트의 변화 : Zero

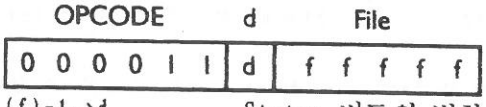
Status 의 Digit Carry 비트가 TEST되어 0이면 증가 명령은 스킵되고 1이면 파일 레지스터는 증가된다.

동일 명령은 : BTFSC 3,1
INCF f,d

SUBCF f,d — **Carry 비트 TEST후 1이면 감소 0이면 다음으로 스킵한다.** — **SUBCF f,d**



Carry 가 클리어면 스킵

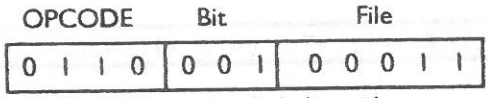


(f)-1→d Status 비트의 변화 : Zero

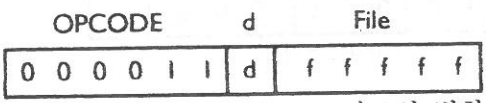
Status 의 Carry 비트가 TEST되어 0이면 증가 명령은 스킵되고 1이면 파일 레지스터는 감소된다.

동일 명령은 : BTFSC 3,0
DECF f,d

SUBDCF f,d	Digit Carry 비트 TEST후 1이면 감소 0이면 다음으로 스킵한다.	SUBDCF f,d



Digit Carry 가 클리어면 스킵



(f)-1→d　　Status 비트의 변화 : Zero

Status 의 Digit Carry 비트가 TEST되어 0이면 증가 명령은 스킵되고 1이면 파일 레지스터는 감소된다.

동일 명령은 : BTFSC 3,1
DECF f,d

4.1.9 프로그램 페이지 경계를 넘는 CALL 명령

프로그램 페이지를 넘는 CALL 명령을 쉽게 호출하기 위해 하나의 특수 명령 니모닉이 있다. 이 니모닉은 목적번지인 Status 레지스터의 페이지 비트를 세트하거나 리셋한다. CALL 명령에서 번지의 8번비트는 언제나 0 이므로 목적번지는 페이지의 하위 반 이내에 있어야 한다.

LCALL k	페이지 경계를 넘는 CALL 명령 이다.	LCALL k

이 명령은 실제로 5개의 명령으로 확장된다 : STATUS 레지스터의 페이지 비트를 세트 하거나 리셋하기 위한 2비트의 조작 명령, 목적번지로의 CALL 명령, 그리고 그들이 CALL 명령 호출 이전상태로 가기위해 STATUS 레지스터의 페이지 비트를 리셋하는 2개의 비트 조작 명령.

동일 명령 : BCF 3,5 또는 BSF 3,5
BCF 3,6 또는 BSF 3,6
CALL k
BCF 3,5 또는 BSF 3,5
BCF 3,6 또는 BSF 3,6

표 4-1 추가 명령 세트 요약

Instruction - Binary	이름	니모닉	동일 명령	Status
0100 0000 0011	Clear Carry	CLRC	BCF 3,0	-
0101 0000 0011	Set Carry	SETC	BSF 3,0	-
0100 0010 0011	Clear Digit Carry	CLRDC	BCF 3,1	-
0101 0010 0011	Set Digit Carry	SETDC	BSF 3,1	-
0100 0100 0011	Clear Zero	CLRZ	BCF 3,2	-
0101 0100 0011	Set Zero	SETZ	BSF 3,2	-
0111 0000 0011	Skip on Carry	SKPC	BTFSS 3,0	-
0110 0000 0011	Skip on No Carry	SKPNC	BTFSC 3,0	-
0111 0010 0011	Skip on Digit Carry	SKPDC	BTFSS 3,1	-
0110 0010 0011	Skip on No Digit Carry	SKPNDC	BTFSC 3,1	-
0111 0100 0011	Skip on Zero	SKPZ	BYFSS 3,2	-
0110 0100 0011	Skip on Non Zero	SKPNZ	BTFSC 3,2	-
0010 001f ffff	Test File	TSTF f	MOVF f,1	Z
0010 000f ffff	Move File to W	MOVFW f	MOVF f,0	Z
0010 011f ffff 0010 10df ffff	Negate File	NEGF f,d	COMF f,1 INCF f,d	Z
0110 0000 0011 0010 10df ffff	Add Carry to File	ADDCF f,d	BTFSC 3,0 INCF f,d	Z
0110 0000 0011 0010 11df ffff	Subtract Carry from File	SUBCF f,d	BTFSC 3,0 DECF f,d	Z
0110 0010 0011 0010 10df ffff	Add Digit Carry to File	ADDDCF f,d	BTFSC 3,1 INCF f,d	Z
0110 0010 0011 0000 11df ffff	Subtract Digit Carry from File	SUBDCF f,d	BTFSC 3,1 DECF f,d	Z
101k kkkk kkkk	Branch	B k	GOTO k	-
0110 0000 0011 101k kkkk kkkk	Branch on Carry	BC k	BTFSC 3,0 GOTO k	-
0111 0000 0011 101k kkkk kkkk	Branch on No Carry	BNC k	BTFSS 3,0 GOTO k	-
0111 0010 0011 101k kkkk kkkk	Branch on Digit Carry	BDC k	BTFSC 3,1 GOTO k	-
0110 0100 0011 101k kkkk kkkk	Branch on No Digit Carry	BNDC k	BTFSS 3,1 GOTO k	-
0111 0100 0011 101k kkkk kkkk	Branch on Zero	BZ k	BTFSC 3,2 GOTO k	-
0111 0100 0011 101k kkkk kkkk	Branch on Non Zero	BNZ k	BTFSS 3,2 GOTO k	-
---- ---- ----	Call across page boundary	LCALL k	BCF 3,5 or BSF 3,5 BCF 3,6 or BSF 3,6 CALL k BCF 3,5 or BSF 3,5 BCF 3,6 or BSF 3,6	-

4.3 I/O 프로그래밍시 주의사항

양방향 I/O 포트로의 사용과 입력 혹은 출력 전용 I/O 포트로의 사용에는 어떤 운용 규칙이 있다. 이 규칙들은 I/O 포트 동작을 위한 명령 시퀀스를 따라야만 한다.

4.3.1 양방향 I/O 포트

양방향 포트들은 입력, 혹은 출력 양쪽으로 사용할 수 있다. 입력으로 되면 이 포트들은 래치되지 않는다. 어떠한 입력 이라도 명령에 의해 읽혀지기 까지 포트에 안정되어 있어야만 한다. 출력은 출력 래치가 다시 씌어질때 까지 래치되어 바뀌지 않는다.

몇몇 명령들은 입력후 즉시 출력과 같이 내부적으로 동작하는것이 있다. 예를들어 BCF 와 BSF 명령은 포트 전체를 CPU 로 읽어 들여 비트 연산을 실행하고 다시 출력을 내보내는데, 이 명령들을 사용할 때는 주의해야 한다.

4.3.2 양방향 I/O 포트에서의 연속적 연산

같은 I/O 포트에서 명령이 연속되면 주의해야 한다. 명령의 시퀀스는 파일을 CPU로 읽혀지게 하는 명령들(MOVF, BIT SET, BIT CLEAR, BIT TEST)이 실행되기 이전에 핀 전압을 안정화 되게 끔 진행 되어야 한다. 그렇지 않으면 현재 상태보다 그 핀의 이전 상태가 CPU로 읽혀질 수가 있다. 자세한 사항은 **매뉴얼 DS30015 (EPROM Based 8-Bit CMOS Microcontroller Series)**을 참고하기
바란다.

예제 1 : C5(포트 C 의 5번 비트)를 통하여 3ms 펄스를 발생.

```
프로그램 스텝                    설명
    MOVLW   .250        ; 10 진수 250을 W 에 로드한다.
    MOVWF    11         ; 250을 F11 로 옮김.
    BSF     7,5         ; F7 의 5번 비트를 세트시킴.
A   DECFSZ   11,1       ; F11 을 감소시키고 0이면 스킵.
    GOTO      A          ; 이 GOTO 명령으로 F11 은 250번 감소하게 된다. 감소시간이
                         4μS이고 GOTO가 8μS 에 실행되므로 루프에 걸리는 시간은
                         (4+8)μS × 250 = 3ms 가 된다.
    BCF     7,5         ; F7 의 5번 비트를 리셋시킴.
```

주의 : 정확한 타이밍을 위해서 반드시 외부 오실레이터를 사용해야하며 그렇지 않으면 실제 타이밍은 외부 RC 소자의 허용오차에 따라 변한다.

예제 2 : 조건적인 테스트와 분기

F11 의 내용을 상수와 비교하여 같으면 GOTO OK 를 실행하고 다르면 GOTO NO를 실행한다.

```
프로그램 스텝                    설명
    MOVF    11,W        ; F11 의 내용을 W 레지스터로 옮긴다.
    XORLW   CONST        ; 상수 CONST와 W 의 내용과 Exclusive OR를 하여 같으면 W
                         의 값은 0이되고 Status 레지스터의 2번째 비트는
                         세트된다. 여기서 SUBWF 명령이 사용될 수 있으나 이 명령
                         을 사용하면 Carry 비트가 변할 수 있다.
    BTFSS   3,2         ; F3 의 2번 비트가 1 이면 다음으로 스킵한다.
    GOTO    NO          ; 그 값이 다를때.
    GOTO    OK          ; 그 값이 같을때.
```

예제 3 : 8비트 파일 레지스터의 시리얼 출력.

이 예제에서 파일 레지스터 F0A 의 내용은 I/O B0(F6 의 0번 비트)으로 출력된다. B1(F6 의 1번 비트)은 출력의 상승 edge 와 동기하기 위해 사용된다.

```
프로그램 스텝                    설명
    MOVLW   .8          ; decimal 8을 W 레지스터로 넣는다.
    MOVWF   11          ; W 값을  F11 로 옮긴다.
LOOP BCF    6,1         ; 출력 동기신호를  클리어 한다.
    RRF     0A,1        ; F10을 오른쪽으로 회전 시킨다. 0번 비트는 Carry가 된다.
    BTFSS   3,0         ; F3 의 carry 비트를 시험하여 1이면 스킵한다.
    GOTO    A           ; Carry 비트가 클리어면 A 로 간다.
    BSF     6,0         ; Carry 비트가 1이면 C0를 세트시킨다.(출력은 high)
    GOTO    B           ; B 로 간다.
A   BCF     6,0         ; Carry 비트가 0이면 C0를 클리어 한다.(출력은 low)
B   BSF     6,1         ; 동기신호를 세트한다.
    DECFSZ  11          ; 8비트 모두를 출력했나 ?
    GOTO    LOOP        ; 아니면 다음 비트를 출력하러 LOOP로 돌아간다.
    BCF     6,1         ; 맞으면 동기신호를 리셋 시킨다.
    END
```

예제 4 : F0A 하위 4비트의 7-세그먼트 변환

상위 4비트는 0이라고 가정한다. 7-세그먼트는 F7을 통해 출력된다. 이 프로그램은 계산된 GOTO 명령을 보여준다. 이 코드 예제는 전형적인 7-세그먼트 디스플레이 이다. LED 포지션은 B'0abcdefg' 이다.

```
프로그램 스텝                          설명
        MOVF    0A,W        ; F0A 값(BCD number)을 W 레지스터로 옮긴다.
        CALL    CONVRT      ; 변환 서브루틴 호출.
        MOVWF   7           ; 7-세그먼트 출력값을 출력포트 F7 로 보낸다.
        •
        •
        •
CONVRT  ADDWF   2           ; W 값과 PC 값을 더한값이 새로운 PC 번지가 된다.

        RETLW   B'00000001' ; 7-세그먼트 코드인 0의 보수
        RETLW   B'01001111' ; 7-세그먼트 코드인 1의 보수
        RETLW   B'00010010' ; 7-세그먼트 코드인 2의 보수
        RETLW   B'00000110' ; 7-세그먼트 코드인 3의 보수
        RETLW   B'01001100' ; 7-세그먼트 코드인 4의 보수
        RETLW   B'00100100' ; 7-세그먼트 코드인 5의 보수
        RETLW   B'01100000' ; 7-세그먼트 코드인 6의 보수
        RETLW   B'00001111' ; 7-세그먼트 코드인 7의 보수
        RETLW   B'00000000' ; 7-세그먼트 코드인 8의 보수
        RETLW   B'00001100' ; 7-세그먼트 코드인 9의 보수
```

RETLW 명령어는 CALL 다음 (MOVWF 7)으로 복귀하기 전에 W 레지스터 옆의 상수를 가지고 온다.

참고 : 위 에서 7-세그먼트는 공통 애노드 구성이며 서브루틴은 00 에서 0FF 사이에 있어야 한다.

예제 5 : 플래그 비트의 조건에 따라 상수 2개중 하나를 W 레지스터로 보낸다.

이 예제는 효율적인 코딩 방법(방법 2)을 보여준다.

```
        방법 1
        BTFSC   FLAG, BIT   ; 플래그 테스트
        GOTO    A
        MOVLW   LITERAL_1   ; 플래그 = 0
        GOTO    CONTINUE
A       MOVLW   LITERAL_2   ; 플래그 = 1


        방법 2
        MOVLW   LITERAL_2
        BTFSS   FLAG, BIT   ; 플래그 테스트
         MOVLW   LITERAL_1   ; 플래그 = 0
```

예제 6 : F9 가 지정한 파일을 F7(포트C)로 출력

F09 에 0A 가 들어있고 F24 에 5A가 들어 있다고 하자, 아래 프로그램은 F7 을 통해 5A를 출력할 것이다.

```
프로그램 스텝                          설명
        MOVF    9, W        ; F09 의 내용을 W 로 옮긴다.(W=0A)
        MOVWF   4           ; W 의 내용을 F4(FSR)로 옮긴다.
        MOVF    0, W        ; FSR 이 지정하는 파일의 내용을 W 로 보낸다.
        MOVWF   7           ; W 의 내용을 F7 로 보낸다.
```

예제 7 : F5 에서부터 F1F 까지의 파일 레지스터를 클리어 시킨다.

이 프로그램은 FSR 의 사용과 F0 를 이용한 간접번지 지정을 보여준다.

```
프로그램 스텝                          설명
        MOVLW   5           ; 상수 5를 W 에 넣는다.
        MOVWF   4           ; 상수 5를 FSR 에 넣는다.
LOOP    CLRF    0           ; FSR 에 의해 지정된 파일을 클리어 한다.
        INCFSZ  4, 1  *     ; FSR을 증가 시킨다.
        GOTO    LOOP        ; 다음 파일을 지우기 위해 루프로 돌아간다.
        END                 ; F5 에서 F1F 까지 클리어 되었다.
```

* FSR의 상위 3비트는 항상 1이며 F1F 에서는 FSR의 모든 비트가 1이 되면, INCFSZ 명령은 ALU 에서 그값을 읽고 증가 시킨다. 증가 시키면 0이되어 스킵한다. 그러나 이것을 수행후 에도 FSR을 다시 읽으면 그값은 0E0으로 읽힌다.

5 장 : 어셈블러 지시어(directives).

소스 코드에 있는 어셈블러 지시어는 직접 opcode 로 번역 되지는 않는다. 대신에 그들은 로케이션 카운터를 조정하거나 메모리 영역을 초기화하고 리스팅 출력포맷을 정하는데 사용된다.

아래에 모든 지시어들이 간단하게 나열되어 있다. 지시어 형식에서 [ ] 괄호는 옵션이다.

Directives	Effect
DATA	Reserve data in memory
ELSE	Conditional assembly converse
END	End of source code
ENDIF	End conditional assembly segment
EQU	Equate a symbol to an expression
IF	Conditional aseembly statement
INCLUDE	Insert external source file
LIST	List of options
ORG	Set program origin
PAGE	Insert page eject in listing file
RES	Reserve storage in memory
SET	Set a symbol equal to an expression
SPAC	Insert lines in listing file
TITLE	Define program heading
ZERO	Initialize storage to zero in memory

DATA — 프로그램 메모리에 데이터를 예약한다. — DATA

형식 :

```
[<label>]  DATA  <operand>  [,<operand>]
```

설명 :

각 오퍼랜드는 expression 이나 70개 까지의 문자가 된다. 어셈블러는 각각의 수식 이나 문자를 메모리에 1개씩 할당한다.

예 :

```
DATA  ONE, !+4, E"E"
DATA  "TESTING"
```

END — 어셈블리를 끝 마친다. — END

형식 :

[<label>] END [<comment>]

설명 :

어셈블러 소스의 마지막 행임을 알려주면 그다음의 추가적인 라인은 무시된다.

EQU — 심볼에 수식을 대응 시킨다 — EQU

형식 :

[<label>] EQU <expression> [<comment>]

설명 :

수식은 레이블로 할당된다.

예 :

FOUR EQU 4

IF...ELSE...ENDIF — 조건적으로 어셈블리 한다. — IF...ELSE...ENDIF

형식 :

IF <expression>

<source_lines>

ELSE

<source_lines>

ENDIF

설명:

PICALC는 다음의 3개의 코맨드에 따라 조건부 어셈블리를 지원 한다 : IF, ELSE, ENDIF. 이것들은 모두 소스라인의 코맨드 필드에 위치 하여야 한다. 그 형식은 위와 같다.

수식의 값이 0이 아니면 그다음의 소스라인은 ELSE 나 ENDIF를 만날때 까지 어셈블 된다. IF 문은 반드시 ENDIF 와 함께 씌어진다. 수식의 값이 0이면 그다음의 소스 라인은 ELSE 나 ENDIF 를 만날때까지 무시된다. 수식 값이 0이 되어 ELSE를 만나면 소스라인은 ENDIF 를 만날때 까지 어셈블된다. ELSE는 옵션이며 IF 문은 얼마든지 네스트할 수 있다.

예 :

```
IF      COND-5
MOVLW   CONST1
ELSE
MOVLW   CONST2
ENDIF
MOVWF   FILE
```

위 에서 심볼 "COND"의 값이 5이면 레지스터 "FILE"의 값은 "CONST2"의 값으로 로드된다. 그렇지 않으면 "CONST1"의 값이 사용된다.

INCLUDE — 외부 파일을 삽입 한다. — INCLUDE

형식 :

```
<label>  INCLUDE  "<filename>"  [comments]
```

설명 :

지정된 파일의 소스코드로 읽혀진다. 지정된 파일에서 EOF 를 만나면 다시 원래의 소스코드로 되돌아 온다. 8단계 까지 Include nesting을 할수 있다. 만일 레이블이 사용되면 로케이션 카운터의 현재값으로 주어진다.

예 :

```
INCLUDE  "C:\PICASM\FILE.ASM"
```

LIST — 출력 옵션을 정의 한다. — LIST

형식 :

```
[<label>]  LIST  [C=<Columns>, D, E=<warning level>, F=<type>,
                  N=<lines>, P=<part>, R=<radix>,
                  T=<ON|OFF>, X=<ON|OFF>]
```

설명 :

여러개의 옵션들이 아래에 나타나 있는데 이것들을 이용하여 디폴트의 리스트 세팅값을 바꿀 수 있다.

옵션	설명	디폴트
C=<columns>	프린트 출력을 위해 페이지당 column수를 정의한다. "**columns**"는 반드시 상수 이어야 한다.	C=80
D	PICESII 포맷에서 심볼 테이블을 만든다. (단지 호환성 문제로)	PICICE
E=<level>	에러나 경고 메시지의 레벨을 지정한다. 0=모두 표시 1=Warnings, Fatals, Criticals. 만 표시. 2=Fatals, Criticals. 만 표시. 3=Criticals. 만 표시.	1

F=<type>	오브젝트 코드 포맷 **PICICE, INHS8X, INHX8M, INHX16** 중 하나를 지정한다.(자세한 사항은 6장을 참조)	PICICE
N=<lines>	리스팅 출력을 위하여 페이지당 라인수를 정의한다.	N=<51>
T=<ON/OFF>	'C' 옵션에서 지정한 값으로 리스팅 column을 자른다.	OFF (ON for PICICE)
P=<part>	부품(디바이스)을 지정한다 : **16C54, 16C55, 16C56, 16C57**	16C54
R=<radix>	소스 파일의 기수를 지정한다.(HEX,DEC, OCT)	HEX
X=<ON/OFF>	매크로 전개를 결정한다.	ON

참고 : 오브젝트 코드 모듈과 심볼 테이블 파일들은 항상 16진수로 만들어지며 LIST 옵션과는 별개이다.

ORG — 프로그램 기점을 세트한다. — ORG

형식 :

[<label>] ORG <expression> [<comment>]

설명 :

적당한 값이 로케이션 카운트로 들어가면 프로그램의 절대기점이 정의된다. 레이블이 사용되더라도 그 값으로 주어진다. ORG 지시어가 없으면 로케이션 카운터의 디폴트값은 0 이다.

예 :

```
LBL4    ORG    D'100'      ; 10 진수 100으로 프로그램 카운터를 세트한다.
```

PAGE — 리스팅 파일의 페이지 eject 를 삽입한다. — PAGE

형식 :

[<label>] PAGE [<comment>]

설명 :

리스팅 파일에 form feed 를 삽입한다. 지시어 자체는 리스팅 파일에 나타나지 않는다.

RES — 프로그램 메모리를 예약 한다. — RES

형식 :

[<label>] RES <expression> [<comment>]

설명 :

수식의 값만큼 프로그램 카운터가 증가된다. 이렇게 reserve된 메모리의 값은 예측할 수 없음을 의미한다.

SET — 심볼을 수식에 대응 시킨다. — SET

형식 :

<label> SET <expression> [<comment>]

설명 :

주어진 레이블로 적절한 수식의 값이 할당된다. 지시어 EQU 와는 달리 지시어 SET 는 같은 심볼에 대해 여러번 정의될 수 있다. 어셈블리 도중의 어떤 위치에서도 가장 최근의 지시어 SET값이 심볼 값으로 정의된다.

예 :

```
ONE   SET   1
ONE   SET   ONE+ONE
```

SPAC — 리스팅 파일에 라인을 삽입한다. — SPAC

형식 :

<label> SPAC <expression> [<comment>]

설명 :

수식에 의해 주어진 만큼 공백 라인들을 리스팅 파일에 삽입한다. 지시어 자체는 리스팅 파일에 나타나지 않는다.

SUBTITL — 프로그램의 부제목을 정의한다. — SUBTITL

형식 :

<label> SUBTITL "<string>" [;<comment>]

설명 :

이 subtitle 문자열은 지시어에 이어서 리스팅 파일 각 페이지의 헤더에 위치한다. 만일 지시어가 소스코드의 두번째 라인에 있다면 subtitle은 첫째 페이지에 나타날 것이다. 문자<string>는 132자 까지 쓸 수 있다.

TITLE	프로그램 제목을 정의한다.	TITLE

형식 :

<label> TITLE "<string>" [;<comment>]

설명 :

이 제목은 지시어에 이어 리스팅 페이지의 헤더에 위치한다. 지시어가 소스코드의 첫째 라인에 있다면 타이틀은 첫째 페이지에 나타난다. 문자는 132자 까지 쓸 수 있다.

ZERO	프로그램 메모리를 예약하고 ZERO로 채운다.	ZERO

형식 :

<label> ZERO "<string>" [;<comment>]

설명 :

메모리를 예약하는 것에서는 지시어 RES와 같으나 메모리를 0으로 채운다는 것이 다르다.

5.1 매크로

매크로는 매크로 호출에 의해 어셈블리 소스 코드로 삽입되는 일련의 명령어들 이다. 매크로는 아래에 정해진 방법대로 정의되어야 하며 그 이후로는 소스코드 모듈 어디에서도 호출될 수 있다. 매크로는 재정의 될 수 있으며 언제나 현재 정의된 상태로 실행된다. 파라미터 전달도 매크로에서 가능하다.

5.2 매크로 정의

매크로 정의는 매크로 헤딩, 매크로 내용, 매크로 종료 의 3단계로 이루어져 있다.

5.2.1 매크로 헤딩

매크로 헤딩의 형식은 다음과 같다 :

<name> MACRO [<parameter1> ... <parametern>] [<comment>]

여기서,

<name> - 매크로 이름
<parameter> - 매크로 호출에 의해 전달 되어야 할 파라미터

참고 : 매크로 헤딩에서 코맨트는 파라미터가 있건 없건 쓸 수 있다.

매크로의 이름은 첫 문자가 'A'-'Z', 'a'-'z', '?', '@', '#', ':', '_' 등으로 시작되는 1 에서 132 까지의 알파뉴메릭 문자이어야 한다. 매크로 실행을 위해 매크로를 호출할 때 매크로 이름으로 사용한다.

만일 매크로 이름이 니모닉이나 지시어와 같으면 어셈블러는 에러 메시지를 표시한다.

5.2.2 매크로 내용

매크로 내용은 매크로 정의후에 즉시 시작하며 매크로 종료를 만날 때 까지 계속된다. 매크로 내용은 어떤 필드에서 파라미터를 포함하는 소스 라인들의 시퀀스로 이루어 진다. 매크로가 실행될때 모든 파라미터는 매크로 호출에 의해 그에 대응하는 인수들로 대치된다. 코맨트의 파라미터들은 인정되지 않는다.

5.2.3 매크로 종료

지시어 ENDM 은 매크로 정의를 끝낸다. ENDM 은 반드시 다른 매크로를 만나기 전 반드시 종료하여야 한다. 매크로 종료의 형식은 다음과 같다 :

```
[<label>]  ENDM  [<comment>]
```

5.3 매크로 호출

한번 매크로가 있의되면 아래에 나열한 것과 같이 소스 모듈의 어디서라도 실행될 수 있다 :

```
[<lavel>]  <name>  [<argument>[,<argument>] .. [<comment>]]
```

여기서,

<label> - 로케이션 카운터의 현재 값을 할당한다.
<name> - 실행될 매크로의 이름.
<argument> - 매크로에서 파라미터로서 통과 되어야할 어떤 상수나 심볼.

매크로 호출 그 자체는 메모리에 위치하지 않으나 매크로 실행은 현재 메모리 위치에서 시작된다. 콤마는 각 인수들을 구분하기 위해 사용할 수도 있다. 이 경우 인수는 NULL이 된다. 인수 리스트는 공백문자나 세미콜론 에 의해 종료된다.

5.4 파라미터

모든 인수들은 매크로 실행에서 문자로서 통과된다. 그러므로 심볼은 그 값이 아니라 이름을 통해 전달된다. 즉, 파라미터들은 매크로가 전개되기 전 까지 그 값이 결정되지 않는다. 그러므로 매크로 도중의 지시어 SET는 매크로에 의해 전달된 인수의 값을 바꿀 수도 있다.

5.4.1 매크로의 예

아래의 예제는 PIC16C56, PIC16C57 에서 프로그램 페이지 사이를 스위치 시켜주는 유용한 프로그램인 LONG GOTO(LGOTO) 라 불리는 매크로를 이용하고 있다. 이 매크로는 사용하고자 하는 STATUS 레지스터의 페이지 선택 비트를 세트시키거나 리셋시키기 위해 매크로의 내용안에 조건 어셈블리를 사용하고 있다.

5.4.1.1 매크로 정의

```
LGOTO   MACRO   P1,P0,LABEL  ; LONG "GOTO" (INCL. PAGE SELECT)
; VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
     IF P1
     BSF      STATUS,PAGE_1
     ELSE
     BCF      STATUS,PAGE_1
     ENDIF
     IF P0
     BSF      STATUS,PAGE_0
     ELSE
     BCF      STATUS,PAGE_0
     ENDIF
     GOTO     LABEL
;VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
     ENDM
```

5.4.1.2 매크로 호출

```
 LGOTO  1,0,NEXT2        ;          jump to label "NEXT" on page 2
```

5.4.1.3 매크로 코드

아래의 코드에서 "a" 는 현재 PC 의 번지이다.

```
     a                  LGOTO  1,0,NXET2
       +;VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
     a                  +      if      1
     a                  +      BSF     STATUS,PAGE1
     a+1                +      else
     a+1                +      BCF     STATUS,PAGE1
     a+1                +      endif
     a+1                +      if      0
     a+1                +      BSF     STATUS,PAGE0
     a+1                +      else
     a+1                +      BCF     STATUS,PAGE0
     a+2                +      endif
     a+2                +      GOTO    NEXT 2
     ;VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
```

5.5 Local 심볼

모든 레이블들은 프로그램 전체에서 사용되므로 한번 이상 호출된 레이블이 다시 매크로에서 사용되면 중복 레이블 에러 메시지를 표시한다. 이런 문제를 피하기 위하여 유저는 매크로에서 사용될 레이블들을 매크로 내에 Local로 정의할 수 있다. 어셈블러가 호출하는 매크로는 매번 시스템에 Local 심볼을 할당하는데 ??nnnn 형태로 된다. 그러므로 최초의 Local 심볼은 ??0001이 되고 두번째는 ??0002 ....등 이다. 모든 LOCAL 정의는 지시어 MACRO 직후에 있어야 하고 매크로 내용의 첫째 라인 전에 있어야 하며 다음의 형식을 가진다 :

```
LOCAL  <label> [,<label>] ...
```

LOCAL 명령은 코맨드 필드에 있어야 하고 LOCAL 지시어가 매크로정의 밖에 나타나면 에러 메시지를 표시할 것 이다.

예 :

```
WAIT        MACRO     T
  LOCAL     LAB1
  MOVLW     T
  MOVWF     TIME
LAB1        DECFSZ    TIME,F
  GOTO      LAB1
  ENDM

+MOVLW      D'250'
+MOVWF      TIME
+??0001     DECFSZ    TIME,F
+GOTO       ??0001
+MOVLW      0x250
+MOVWF      TIME
+??0002     DECFSZ    TIME,F
+GOTO       ??0002
```

5.6 매크로 종료

지시어 EXITM 은 매크로 전개를 종료시킨다. 매크로 실행중 지시어 EXITM 은 현재 진행중인 매크로를 중지 시키고 EXITM 과 ENDM 사이의 모든 코드는 무시된다. 매크로가 네스트 되면 EXITM 은 매크로 전개 이전의 레벨로 돌아가기 위한 코드를 발생한다.

```
[<label>]  EXITM  [<comment>]
```

6장 : 어셈블러 출력

각각의 소스파일에서 어셈블러는 2개의 출력 파일(리스팅 파일과 오브젝트 파일) 을 생성한다. 리스팅 파일은 소스 코드와 함께 출력번지에 대한 정보와 그것에 상응하는 오브젝트 코드 정보를 가지고 있다. 오브젝트 파일은 로드되고 실행되는 데이터를 가지고 있다. 이 데이터들은 여러가지 다른 포맷으로 출력될 수 있다.

또, PICALC 는 심볼 이름과 레이블, 그리고 그들의 값을 포함하는 심볼 파일도 생성한다. 이 심볼 파일은 in-circuit-emulation 시스템(PIC-ICE) 이나 시뮬레이터 소프트웨어(PICSIM) 에 의해 로드된다.

심볼 테이블 파일은 소스파일과 이름은 같으나 확장명은 "SYM" 이 된다.

6.1 리스팅 파일

리스팅 파일의 각 페이지는 지시어 TITLE 에 의해 페이지 수와 제목을 포함한 헤더와 함께 시작한다. 소스 코드의 각 라인은 리스팅 파일에 재생되며 소스 라인 번호와 로케이션 카운터, 오브젝트 코드와 함께 표시된다.

예 :

```
0018    0013    0142    CLRF2    ;Clear File Register 2.
                        •••
                        •••
                        •••
                        •••
```

첫째 필드는 소스 코드에 있는 소스라인의 번호를 나타내는 4 자리의 10진수이다.

두번째 필드는 현재의 로케이션 카운터를 나타내는 4 자리의 16진수 이다.

세번째 필드는 소스라인 으로 부터 생성되는 opcode 를 나타내는 16진수 이다. 이것은 오브젝트 코드에서도 볼 수 있는 실제적인 값이다. 라인의 나머지는 소스 코드로 부터 직접 복사된다.

다음의 지시어들은 리스팅 파일에 나타나지 않는다 : PAGE 와 SPAC

6.1.1 Cross-Reference Listing

Cross-Reference 테이블은 리스팅 파일의 맨끝에 생성된다. 이 테이블은 아래의 정보와 함께 소스 파일에 사용된 모든 심볼의 리스트를 포함한다 :

VALUE - 4 자리의 8진수로 표시되는 심볼값.
DEFN - 심볼이 정의된 소스 라인을 나타내는 4자리의 10진수.
REFERENCES - 심볼이 참고된 모든 소스라인을 나타내는 4자리의 10진수.

다음의 예는 cross-reference listing 을 보여준다 :

Cross-Reference Listing

LABEL	VALUE	DEFN	REFERENCE
AR	0001	0005	0018 0016
CR	0000	0009	0021 0021 0017
DR	0000	0011	0017
EIGHT	0008	0020	
FIVE	0005	0017	
FOUR	0004	0016	
NINE	0009	0021	
PR	0100	0007	0020 0020 0019 0016
			0015 0015 0015 0014
			0014 0014 0013
SEVEN	0007	0019	
THREE	0003	0015	

6.1.2 에러 메시지

에러 메시지는 리스팅 파일이나 터미널을 통해 나타날 수 있다. 모든 메시지는 다음의 형식을 가진다 :

```
<error level>:  [<error level count>] : <messege>
```

여기서,

<error level> - 에러 메시지의 레벨
<error level count> - 이 레벨의 에러 갯수
<messege> - 에러의 설명

에러의 정도는 다음의 범주중 하나가 된다 :

Comment 유저에게 유용한 정보를 준다. 이 에러들은 디폴트 설정시 안나타난다.
Warning 이 상태에서는 바라는 결과가 나올수도 있고 나오지 않을 수도 있다.
Fatal 이 상태는 정확하지 못한 오브젝트 코드를 생성시킨다.
Critical 이 상태에서 어셈블리를 계속할 수 없다.

에러 메시지들은 부록 B 에 번호화 되어있다.

6.2 오브젝트 코드

어셈블러는 지시어 LIST F 나 코맨드 라인 옵션에 의해 3개의 다른 오브젝트 코드 포맷을 설정할 수 있다. 이어지는 섹션에서 이 각각의 포맷들을 설명할 것이다. 또, 어셈블러는 PIC-ICE 개발장비를 위해 배타적인 특수 오브젝트 코드를 만든다.

6.2.1 분리된 8-Bit Intellec Hex 포맷

이 포맷은 INHX8S 옵션이 지시어 LIST F 와 사용되거나 코맨드 라인의 "f" 옵션이 사용되면 어셈블러에 의해 출력된다.

이 포맷은 2개의 8비트 Hex 파일을 생성한다. 한 파일은 어드레스/데이터의 상위 8비트를 포함하고 다른 파일은 하위 8비트를 포함한다. 오브젝트를 위한 파일 확장명은 상위와 하위를

고려하여 각각 '.obl' 이나 '.obh' 가 된다. 더 자세한 사항은 부록 A 를 참고하기 바란다. 이 포맷은 8-Bit Intellec Hex 포맷과도 호환되며 특히 NMOS PIC 16C7X와 PIC16C5X 를 에뮬레이트하기 위한 PFD 보드의 상/하위 바이트 EPROM 을 프로그램 하는데에 유용하다.

예 :

```
<filename>.OBL:

:0A0000000000000000000000000F6
:0A0000000000000000000000000F6
:10001900002840688A8E8C82868A989EA28086ABFAA
:10002900E0E82868BFE8C80808080343O3E8E8FFDO
:03003900FFFF19AD
:00000001FF
```

예 :

```
<filename>.OBH

:0A0000000000000000000000000F6
:0A0000000000000000000000000F6
:10001900000000000000001010101010202020CA
:10002900020203030303030404040405060707O883
:03003900008080AAA
:00000001FFF
```

6.2.2 혼합된 8-Bit Intellec Hex 포맷

이 포맷은 **INHX8M** 옵션이 지시어 **LIST P** 와 사용되거나 코맨드 라인의 'f' 옵션이 사용되면 어셈블러에 의해 출력된다.

이 포맷은 상/하위 바이트가 조합된 8-Bit Hex 파일을 생성한다. 이 포맷에서는 각각의 어드레스가 8 비트만 포함할 수 있으므로 모든 어드레스는 2배가 된다. 파일 확장명은 '.obj'가 되며 자세한 사항은 부록 A를 참고하기 바란다.

이 포맷은 PIC16C5X 시리즈의 오브젝트 코드를 타사의 EPROM 프로그래머들 (**DATA I/O**사의 **Unisite YO, Logical Devices** 사의 **ALLPRO** 등)에 옮길 때에 유용하다.

6.2.3 16-Bit Hex 포맷

이 포맷은 INHX16 옵션이 지시어 LIST F 와 사용되거나 코맨드 라인의 'f' 옵션이 사용되면 어셈블러에 의해 출력된다.

이 포맷은 하나의 16-Bit Hex 파일을 생성한다. 파일 확장명은 '.obj' 가 된다. 자세한 사항은 부록 A 를 참고하기 바란다.

이 포맷은 **PIC16C5X** 시리즈의 오브젝트를 마이크로칩사의 EPROM 프로그래머인 **"PICPRO II"** 에 옮길 때에 특히 유용하게 쓰인다.

8-Bit 워드 포맷

각 데이터 레코드는 처음 9개의 문자로 시작하여 마지막 2개의 checksum 문자로 끝난다. 각 레코드는 다음과 같은 포맷으로 되어있다 :

```
:BBAAAATTHHHH .... HHHCC
```

여기서,

BB	- 라인위에 나타나는 데이터 워드의 수를 나타내는 두 자리의 Hex 바이트이다.
AAAA	- 데이터 레코드를 위한 시작 번지를 나타내는 4자리의 16진수 이다.
TT	- '01' 로 세트되는 EOF 레코드를 제외하면 언제나 '00' 으로 되는 2자리의 레코드 타입이다.
HH	- 2자리의 16진수 데이터 워드이다.
CC	- 이전의 데이터 레코드를 더한 값의 2의 보수로서 2자리의 Hex checksum 이다.

16-Bit 워드 포맷

16 비트 워드 포맷은 기본적으로 8-Bit 워드 포맷과 같으며 HEX 데이터 워드가 4자리로 확장된다는 점이 다르다.

```
:BBAAAATTHHHHHHHH ....HHHHCC
```

여기서,

BB	- 라인위에 나타나는 데이터 워드의 수를 나타내는 두 자리의 Hex 바이트이다. PICALC 에서는 PICPRO II와 호환성을 갖기위해 "BB" 를 "08"(Hex) 까지 제한한다.
AAAA	- 데이터 레코드를 위한 시작 번지를 나타내는 4자리의 16진수 이다.
TT	- '01' 로 세트되는 EOF 레코드를 제외하면 언제나 '00' 으로 되는 2자리의 레코드 타입이다.
CC	- 레코드에서 이전의 모든 바이트를 더한 값의 2의 보수로서 2자리의 Hex checksum 이다. 4자리의 데이터 워드는 2바이트로 취급된다.

Error Number	Error Message	Error Level
0	"Undefined Error"	Comment
1	"Duplicate Label"	Fatal
2	"Missing FileReg"	Fatal
3	"Error in FileReg"	Fatal
4	"Left shift greater than 16 bits"	Warning
5	"Missing Destination"	Fatal
6	"Bad Destination"	Fatal
7	"Missing Comma"	Warning
8	"Assuming Destination of 1"	Comment
9	"Bad Expression"	Fatal
10	"Missing Label"	Fatal
11	"Unknown Character - Ignored"	Warning
12	"Unexpected EOF"	Critical
13	"Unmatched Quote"	Fatal
14	"Filename truncated"	Warning
15	"Extension truncated"	Warning
16	"Unrecognized Argument"	Warning
17	"File Nesting Error - Too Deep"	Fatal
18	"File Nesting Error - No More Files"	Critical
19	"Cannot Open File"	Fatal
20	"Missing Address Expression"	Fatal
21	"Address expression greater than 9 bits"	Fatal
22	"Unmatched Endif"	Fatal
23	"Unmatched Else"	Fatal
24	"Unmatched If"	Fatal
25	"Redefinition of a static label"	Fatal
26	"Bad string syntax"	Fatal
27	"Bad Local argument"	Fatal
28	"Bad or misplaced instruction"	Fatal
29	"Macro definitions cannot be nested"	Fatal
30	"Bit number out of range"	Fatal
31	"Extra statements ignored"	Warning
32	"Filereg truncated to 5 bits"	Warning
33	"Missing Right Paren"	Fatal
34	"Right shift greater than 16 bits"	Warning
35	"Null Object Record"	Fatal
36	"Unknown instruction type"	Fatal
37	"Unknown Intel8MDS command"	Fatal
38	"Unknown Intel8SDS command"	Fatal
39	"Unknown Intel16DS command"	Fatal
40	"Unknown PICICE command"	Fatal
41	"Unknown PICES command"	Fatal
42	"Unknown PIC type"	Fatal
43	"Unknown OUTPUT format"	Fatal
44	"TRIS must use Filereg 5 6 or 7"	Fatal

45	"Address exceeds PIC 54/55 memory limit"	Fatal
46	"Address exceeds PIC 56 memory limit"	Fatal
47	"Address exceeds PIC 57 memory limit"	Fatal
48	"Address is not within lower half page"	Fatal
49	"PICES format not supported setting format to INHX8M"	Warning
50	"Bad Macro argument type"	Fatal
51	"Value truncated to 8 bits"	Warning
52	"Illegal Constant format"	Fatal
53	"Address change across page boundary ensure page bits are set"	Comment
54	"Illegal Error Level"	Fatal
55	"Missing Bit Number"	Fatal
56	"Address greater than 9 bits truncated"	Comment
57	"Destination from reset vector not in Page 0"	Warning
58	"Unknown radix"	Fatal
59	"Bad Decimal Format"	Fatal
60	"Bad Hex Format"	Fatal
61	"Bad Octal Format"	Fatal
62	"Bad Binary Format"	Fatal
63	"Unknown Symbol"	Fatal
64	"RTCC may miss at least 1 count"	Warning

디폴트 기수 대 상수 포맷

설정한 디폴트의 기수는 다른 상수 포맷을 사용함으로써 결과가 예상과 다를수 있다. 아래의 예제는 디폴트의 기수 때문에 같은 상수가 2개의 다른 결과치를 산출하는 것을 보여준다.

디폴트 기수	상수	값(HEX)	
HEX	001b	001B	16 진수의 상수로 인식
DEC	001b	0001	2 진수의 상수로 인식

문제는 HEX 와 DEC 사이의 디폴트 기수 에서 생긴다. 아래의 테이블은 문제를 일으키는 HEX 와 DEC 의 상수 포맷의 예상값들을 설명하고 있다.

Default Radix	Example	interpreted Value(in HEX)
Decimal	100D	64
	.100	64
	100	64
	100o	40
	100q	40
	q'100'	40
	100b	4
	b'100'	4
	ox100	100
	100H	100
	H'100'	100
Hexadecimal	100D	100D
	.100	64
	100	100
	100o	40
	100q	40
	q'100'	40
	100b	100B
	B'100'	4
	ox100	100
	100H	100
	H'100'	

위 표에서 알수 있듯이 <base>'number' 로 포맷된 상수는 오해의 소지가 없으나 그렇지 않을 경우는 문제를 일으킬 수 있다.

현명전자

서울시 종로구 장사동 173
세운상가 가동 바-443
〒 110-430

대표 전화 : 275-4455
277-7061
기술개발실 : 278-0282
F A X : 277-8470