BNC	Branch if C ≠ 0
BZ	Branch if Z = 1
BNZ	Branch if Z ≠ 0
BN	Branch if N = 1
BNC	Branch if N ≠ 0
BOV	Branch if OV = 1
BNOV	Branch if OV ≠ 0

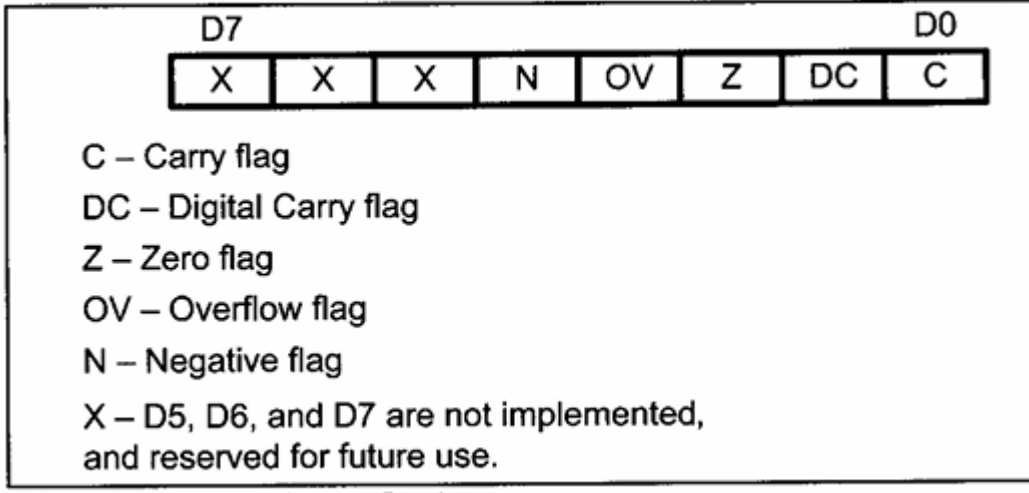


Figure 2-7. Bits of Status Register

Table 2-3: File Register Instructions Using fileReg or WREG as Destination

Instruction		
COMF	fileReg, d	Complement fileReg
DECF	fileReg, d	Decrement fileReg
DECFSZ	fileReg, d	Decrement fileReg and skip if zero
DECFSNZ	fileReg, d	Decrement fileReg and skip if not zero
INCF	fileReg, d	Increment fileReg
INCFSZ	fileReg, d	Increment fileReg and skip if zero
INCSNZ	fileReg, d	Increment fileReg and skip if not zero
MOVF	fileReg, d	Move fileReg
NEGF	fileReg, d	Negative fileReg
RLCF	fileReg, d	Rotate left fileReg through carry
RLNCF	fileReg, d	Rotate left fileReg (No carry)
RRCF	fileReg, d	Rotate right fileReg through carry
RRNCF	fileReg, d	Rotate right fileReg (No carry)
SWAPF	fileReg, d	Swap nibbles in fileReg
BTG	fileReg, d	Bit Toggle fileReg

Table 2-2: ALU Instructions Using Both WREG and fileReg

Instruction		
ADDWF	fileReg, d	ADD WREG and fileReg
ADDWFC	fileReg, d	ADD WREG and fileReg with Carry
ANDWF	fileReg, d	AND WREG with fileReg
IORWF	fileReg, d	OR WREG with fileReg
SUBFWB	fileReg, d	Subtract fileReg from WREG with borrow
SUBWF	fileReg, d	Subtract WREG from fileReg
SUBWFB	fileReg, d	Subtract WREG from fileReg with borrow
XORWF	fileReg, d	Exclusive-OR WREG with fileReg

Note: The d bit selects the destination for the operation. If d = w; the result is stored in WREG (d = 0). If d = F; the result is stored in the fileReg (d = 1). The default is F. That means "`ADDWF myfile`" is the same as "`ADDWF myfile,F.`"

GOTO: 4BYTE

BRA: 2BYTE

CALL: 4BYTE

RCALL: 2BYTE

MAX LOCATIONS IN STACK: 31

(21 BIT WIDE) FROM 01H TO 1FH

Summary of DAW action

After any instruction,

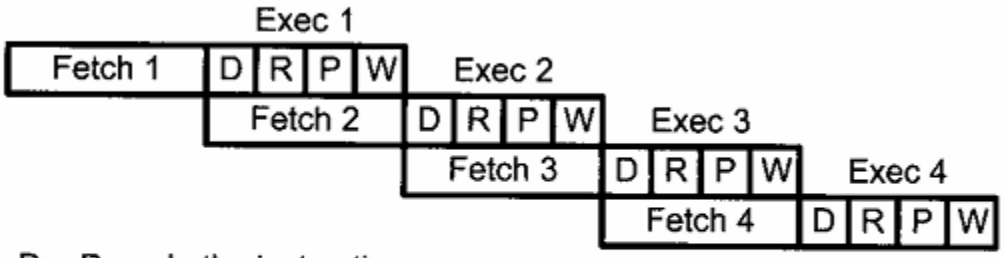
1. If the lower nibble (4 bits) is greater than 9, or if DC = 1, add 0110 to the lower 4 bits.
2. If the upper nibble is greater than 9, or if C = 1, add 0110 to the upper 4 bits.

Chapter 8). In the PIC18, one instruction cycle consists of four oscillator periods. Therefore, to calculate the instruction cycle for the PIC, we take 1/4 of the crystal frequency, then take its inverse, as shown in Example 3-14.

Example 3-14

The following shows the crystal frequency for three different PIC-based systems. Find the period of the instruction cycle in each case.
(a) 4 MHz (b) 16 MHz (c) 20 MHz

Solution:
(a) 4/4 = 1 MHz; instruction cycle is 1/1 MHz = 1 µs (microsecond)
(b) 16 MHz/4 = 4 MHz; instruction cycle = 1/4 MHz = 0.25 µs = 250 ns (nanosecond)
(c) 20 MHz/4 = 5 MHz; instruction cycle = 1/5 MHz = 0.2 µs = 200 ns



D = Decode the instruction
R = Read the operand
P = Process (eg. ADDLW)
W = Write the result to destination register

Table 4-8: Single-Bit (Bit-Oriented) Instructions for PIC18

Instruction	Function
BSF fileReg,bit	Bit Set fileReg (set the bit: bit = 1)
BCF fileReg,bit	Bit Clear fileReg (clear the bit: bit = 0)
BTG fileReg,bit	Bit Toggle fileReg (complement the bit)
BTFSC fileReg,bit	Bit test fileReg, skip if clear (skip next instruction if bit = 0)
BTFSS fileReg,bit	Bit test fileReg, skip if set (skip next instruction if bit = 1)

When is the OV flag set?

In 8-bit signed number operations, OV is set to 1 if either of the following two conditions occurs:

1. There is a carry from D6 to D7 but no carry out of D7 (C = 0).
2. There is a carry from D7 out (C = 1) but no carry from D6 to D7.

Table 5-2: PIC18 Compare Instructions

CPFSGT	Compare FileReg with WREG, skip if greater than	FileReg > WREG
CPFSEQ	Compare FileReg with WREG, skip if equal	FileReg = WREG
CPFSLT	Compare fileReg with WREG, skip if less than	FileReg < WREG

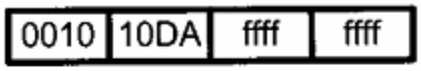
Bank switching

= 0. With A = 0, the access bank is the default bank. Now to use banks other than the access bank, two things must be done:

BCF STATUS,RP0 ; BANK 0

BSF STATUS,RP0 ; BANK 1

1. Load the BSR with the desired bank number, and
2. Make A = 1 in the instruction itself.



D = F, destination is fileReg
D = W, destination is WREG

D – destination for operation
A – bank accessed for operation

A = 0, use default access bank
A = 1, use bank pointed to by
BSR (Bank Selector Register)

$0 \leq f \leq FF$

Bank in PIC Microcontrollers

- A **bank** is a segment of memory in PIC microcontrollers used to organize **Special Function Registers (SFRs)** and **General Purpose Registers (GPRs)**.
- PIC divides memory into **banks** due to limited addressing capability (e.g., 7 or 8 bits).

Need for Bank Switching

1. **Limited Addressing Space:**
   - The microcontroller's instruction set can address only a small part of memory directly. Bank switching allows access to additional memory.
2. **Efficient Memory Management:**
   - Organizes memory into manageable sections, reducing complexity.
3. **Access to Special Registers:**
   - SFRs and peripherals may be located in different banks, requiring switching to configure them.
4. **Program Optimization:**
   - Ensures proper resource allocation without overwriting or conflicts.

The following are some of the major reasons for writing programs in C instead of Assembly:

1. It is easier and less time consuming to write in C than in Assembly.
2. C is easier to modify and update.
3. You can use code available in function libraries.
4. C code is portable to other microcontrollers with little or no modification.

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	–32,768 to +32,767
unsigned short	16-bit	0 to 65,535
short	16-bit	–32,768 to +32,767
unsigned short long	24-bit	0 to 16,777,215
short long	24-bit	–8,388,608 to +8,388,607
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	–2,147,483,648 to +2,147,483,648

Table 7-3: Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	Y=~B

Comparison of C and Assembly Language

Aspect	C Language	Assembly Language
Ease of Use	Easy to learn and write	Difficult and time-consuming
Performance	Moderate (depends on compiler)	High (manually optimized)
Hardware Control	Limited	Precise
Portability	High (cross-platform compatibility)	Low (platform-specific)
Development Speed	Fast	Slow
Maintainability	Easy to read and modify	Difficult for large projects
Error Risk	Low (with debugging tools)	High (prone to low-level errors)
Memory Usage	Higher (due to abstraction overhead)	Lower (efficient use of resources)

Bit-wise shift operation in C

There are two bit-wise shift operators in C: (1) shift right (>>), and (2) shift left (<<).

Their format in C is as follows:

data >> number of bits to be shifted right

data << number of bits to be shifted left

(a) in hex	(b) in decimal
(FFFF - YYXX + 1) × 0.4 µs where YYXX are the TMR0H, TMR0L initial values respectively. Notice that YYXX values are in hex.	Convert YYXX values of the TMR0H, TMR0L register to decimal to get a NNNNN decimal number, then (65536 - NNNNN) × 0.4 µs

Figure 9-6. Timer Delay Calculation for XTAL = 10 MHz with No Prescaler

Assuming XTAL = 10 MHz and no prescaler we can use the following steps for finding the TMR0H and TMR0L registers' values:

1. Divide the desired time delay by 0.4 μs.
2. Perform 65,536 − *n*, where *n* is the decimal value we got in Step 1.
3. Convert the result of Step 2 to hex, where *yyxx* is the initial hex value to be loaded into the timer's registers.
4. Set TMR0L = *xx* and TMR0H = *yy*.

```
SET RCON,IPEN
SET INTCON,GIEH
SET INTCON,GIEL
```

SEMICONDUCTOR MEMORY

storage for code and data. Semiconductor memories are connected directly to the CPU and are the memory that the CPU first asks for information (code and data). For this reason, semiconductor memories are sometimes referred to as *primary memory*. The most widely used semiconductor memories are ROM and RAM.

The number of bits that a semiconductor memory chip can store is called chip *capacity*. It can be in units of Kbits (kilobits), Mbits (megabits), and so on.

Memory chips are organized into a number of locations within the IC. Each location can hold 1 bit, 4 bits, 8 bits, or even 16 bits, depending on how it is designed internally. The number of bits that each location within the memory chip can hold is always equal to the number of data pins on the chip. How many loca-

To summarize:

1. A memory chip contains 2^x locations, where x is the number of address pins.
2. Each location contains y bits, where y is the number of data pins on the chip.
3. The entire chip will contain $2^x \times y$ bits, where x is the number of address pins and y is the number of data pins on the chip.

One of the most important characteristics of a memory chip is the speed at which its data can be accessed. To access the data, the address is presented to the address pins, the READ pin is activated, and, after a certain amount of time has elapsed, the data shows up at the data pins. The shorter this elapsed time, the better, and consequently, the more expensive the memory chip. The speed of the memory chip is commonly referred to as its *access time*. The access time of memory chips varies from a few nanoseconds to hundreds of nanoseconds, depending on the IC technology used in the design and fabrication process.

ROM is a type of memory that does not lose its contents when the power is turned off. For this reason, ROM is also called *nonvolatile* memory. There are different types of read-only memory, such as PROM, EPROM, EEPROM, flash EPROM, and mask ROM. Each is explained below.

PROM refers to the kind of ROM that the user can burn information into. In other words, PROM is a user-programmable memory. For every bit of the PROM, there exists a fuse. PROM is programmed by blowing the fuses. If the information burned into PROM is wrong, that PROM must be discarded because its internal fuses are permanently blown. For this reason, PROM is also referred to as OTP (one-time programmable). Programming ROM, also called *burning* ROM, requires special equipment called a ROM burner or ROM programmer.

Comparison Between EPROM and EEPROM

Aspect	EPROM (Erasable Programmable Read-Only Memory)	EEPROM (Electrically Erasable Programmable Read-Only Memory)
Erasure Method	Erased using **UV light** through a quartz window.	Erased using **electrical signals**, byte by byte.
Programming Method	Data is written using a special **programmer device.**	Data is written electrically and can be modified in-circuit.
Ease of Use	Requires removal from the circuit for erasure/programming.	Can be erased and reprogrammed without removal (in-circuit).
Access Type	Block-wise erasure and programming.	Byte-wise erasure and programming.
Reusability	Limited; frequent erasures can degrade functionality.	Higher write/erase cycles compared to EPROM.
Speed	Slower due to UV light erasure and manual handling.	Faster as erasure and programming are done electrically.
Cost	Generally cheaper than EEPROM.	More expensive per byte.
Use Case	Ideal for long-term storage of firmware or stable data.	Suitable for applications requiring frequent updates to data.
Durability	Moderate, prone to physical damage (window exposure).	More durable, compact, and resistant to environmental factors.
Non-Volatility	Retains data without power.	Retains data without power.

Feature	Flash Memory	EPROM
Erasure Method	Electrically (block/sector)	UV light (entire chip)
Reprogramming	Easy and frequent	Limited and cumbersome
Cost	Lower due to mass production	Higher for small-scale use
Durability	Better for frequent rewriting	Less durable with fewer write cycles
Convenience	Highly convenient	Requires special UV equipment
Modern Use	Ubiquitous in consumer devices	Rare, mostly replaced by Flash

RAM memory is called *volatile* memory because cutting off the power to the IC results in the loss of data. Sometimes RAM is also referred to as RAWM (read and write memory), in contrast to ROM, which cannot be written to. There are three types of RAM: static RAM (SRAM), NV-RAM (nonvolatile RAM), and dynamic RAM (DRAM). Each is explained separately.

1. **SRAM**: Faster, expensive, and used for small, high-speed memory applications (e.g., CPU caches).
2. **DRAM**: Slower, cheaper, and used for main memory in computers.
3. **NVRAM**: Retains data without power; used for data that must persist (e.g., configuration storage, embedded systems).

Feature	SRAM	DRAM	NVRAM
Data Retention	Volatile (no power = no data)	Volatile (needs refreshing)	Non-volatile (keeps data)
Speed	Fast	Moderate	Varies (usually slower)
Cost	High	Low	Medium to high
Density	Low	High	Medium
Power Consumption	High for switching	Lower than SRAM	Low
Refreshing	Not required	Required	Not required
Applications	CPU caches, registers	Main memory in computers	Backup and persistent storage

Steps in writing to EEPROM

To write a byte of data to a location in the EEPROM memory, we go through the following steps:

1. Load the EEADR registers with the address of the EEPROM location we want to write the data byte to.
2. Load the EEDATA registers with the data byte we want to write to EEP ROM.
3. Set the EECON1 register for the EEPROM write by making (a) EEPGD = 0, (b) CFGS = 0, and (c) WREN = 1.
4. Disable all interrupts globally using "`BCF INTCON, GIE`".
5. Write 55H to the EECON2 dummy register.
6. Write AAH to the EECON2 dummy register.
7. Set WR# to 1 with the instruction "`BSF EECON1,WE`". With WE = 1, the write cycle begins.
8. Upon completion of the write cycle, the WE# bit will be cleared automatically to indicate that the write cycle is finished.
9. Re-enable the interrupts globally using "`BCF INTCON, GIE`".
10. The WREN bit should be cleared to prevent an accidental write to the EEPROM by some runaway program.

Steps in reading from EEPROM

Reading a byte from the EEPROM memory is simple and straightforward as shown in the following steps:

1. Load the EEARD register with the address of the EEPROM location we want to read from.
2. Set the EECON1 register for the EEPROM read by making (a) EEPGD = 0, (b) CFGS = 0, and (c) RD = 1.
3. Within the next instruction cycle, the PIC18 will automatically fetch the data from the EEPROM location and place it in the EEDATA register. The only thing we have to do is to move data from the EEDTAT register to a safe place before we do another read. The following shows how to read a byte from EEPROM and place it in PORTB:

```
MOVLW     0x10        ;read location 10H of EEPROM
MOVWF     EEADR       ;load the EEPROM address
BCF       EECON1,EEPGD    ;point to EEPROM memory
BCF       EECON1,CFGS     ;
BSF       EECON1,RD       ;enable read
NOP  ;data is fetched from EEPROM to EEDATA reg
MOVFF     EEDATA,PORTB    ;place the data in PORTB
```

Feature	GPR	SFR
Purpose	Stores user data and variables.	Configures and controls hardware.
Location	General data memory.	Special memory map.
Access	Fully program-controlled.	Predefined hardware functions.
Examples	Temporary variables, counters.	PORTA, TMR0, ADCON1.

Feature	GRRAM	EEPROM
Definition	Volatile memory for temporary data storage.	Non-volatile memory for long-term data storage.
Data Retention	Loses data when power is off.	Retains data even without power.
Write Cycles	Unlimited	Limited (typically ~10,000 to 1,000,000 cycles).
Speed	Faster read/write operations.	Slower due to byte-level erase/write.
Usage	Temporary program variables, stack, and buffer data.	Storing configurations, calibration data, or firmware.
Applications	Active computations and runtime operations.	Persistent settings like device IDs or logs.