The University of Texas at Arlington

Lecture 7
Arithmetic, Logic Instructions, and Programs

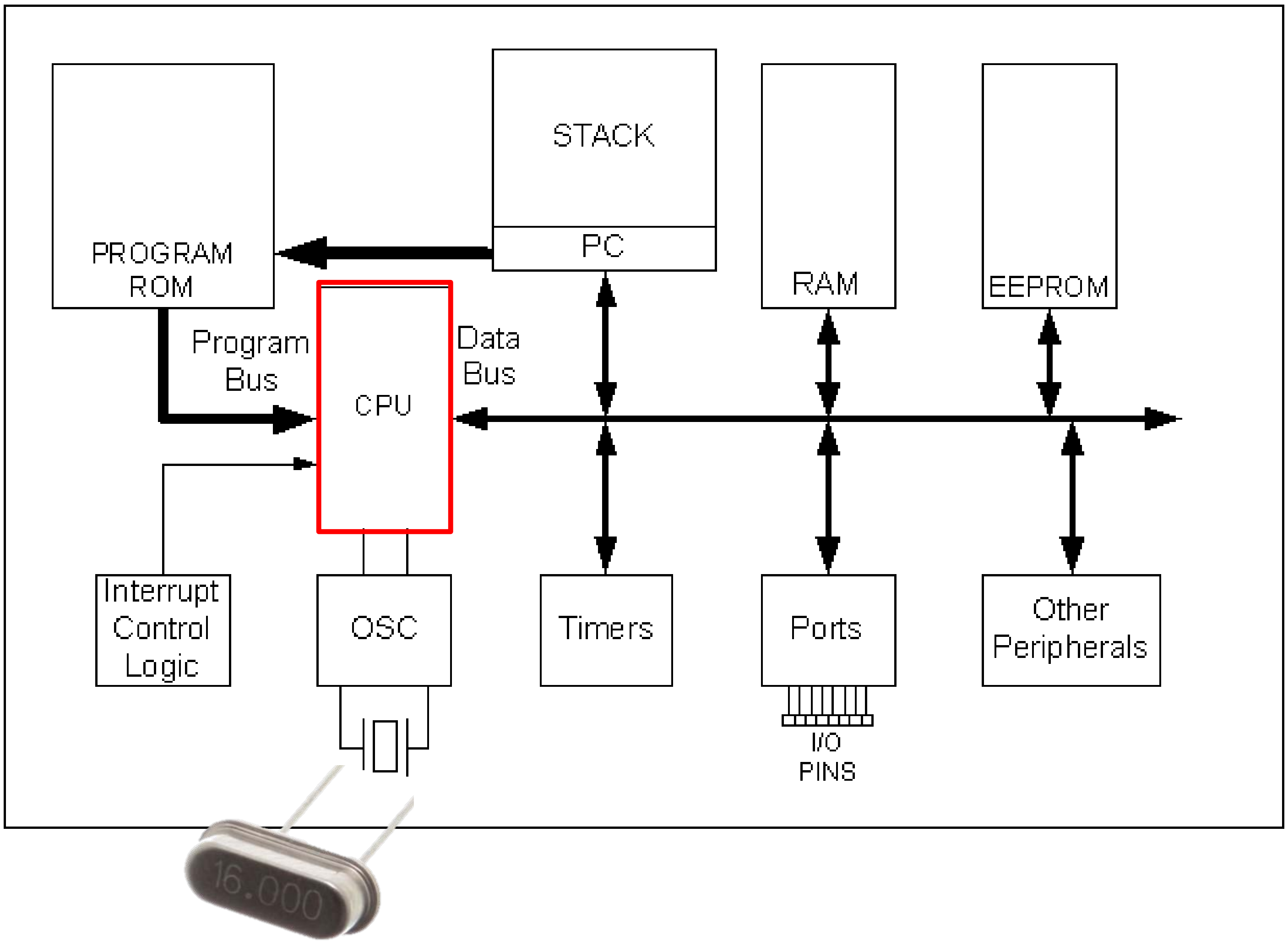
CSE@UTA

CSE 3442/5442

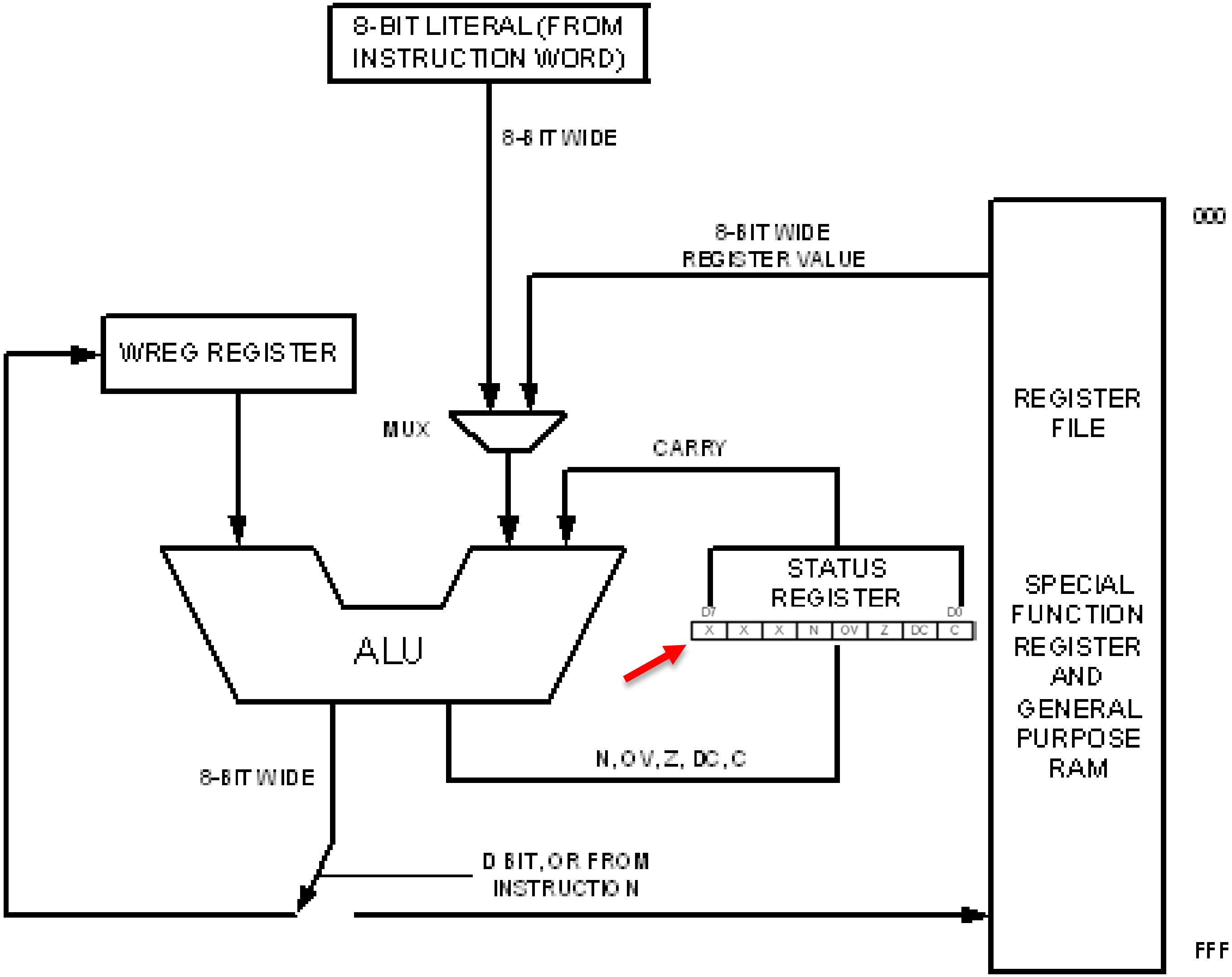
Embedded Systems I

Based heavily on slides by Dr. Gergely Záruba and Dr. Roger Walker

Simplified View of a PIC Microcontroller



Inside the CPU



Recap
Bits of Status Register

D7							D0
X	X	X	N	OV	Z	DC	C

C – Carry flag

DC – Digital Carry flag

Z – Zero flag

OV – Overflow flag

N – Negative flag

X – D5, D6, and D7 are not implemented,
and reserved for future use.

Figure 2-7

Carry Flag

- C Flag is set (1) when there is a "carry out" from the D7 bit
  - Can be set by an ADD or SUB

```
  1101 1010
+ 1010 1111
= 1 1000 1001
```

D7							D0
X	X	X	N	OV	Z	DC	C

Zero Flag

- Z Flag indicates if the the result of an arithmetic or logic operation is 0
  - Result = 0; Z = 1
  - Result ≠ 0; Z = 0

D7							D0
X	X	X	N	OV	Z	DC	C

CSE@UTA

Overflow Flag

- OV Flag is set (1) when the result of a signed number operation is too large
  - The numerical result overflows/overtakes the sign bit of the number
- Usually used to detect errors in <u>signed</u> operations

D7							D0
X	X	X	N	OV	Z	DC	C

Negative Flag

- N Flag is set (1) when the result of an arithmetic operation is less than zero
  - If D7 bit = 0, N = 0, positive result
  - If D7 bit = 1, N = 1, negative result

D7							D0
X	X	X	N	OV	Z	DC	C

Unsigned Numbers

- So far we've only used unsigned numbers
  - Only zero and positive
- All 8 bits are used to represent a number
  - Dec: 0 – 255
  - Hex: 0 – FF
  - Bin: 0 – 1111 1111
- No bits designated for + or - sign

Signed Numbers

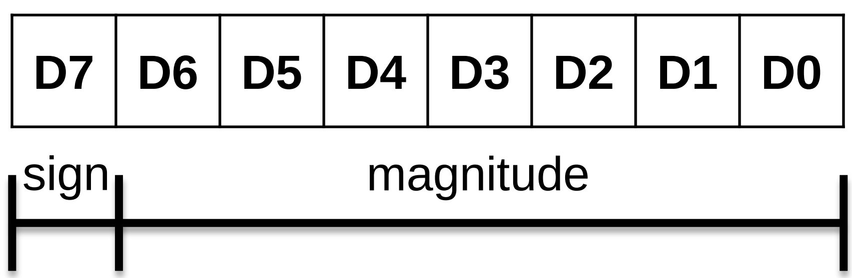
- **Decimal Range: -128 … 0 … +127**
- MSB D7 is the sign bit
  - D7 = 0 → Positive Number
  - D7 = 1 → Negative Number

0 … +127

0	D6	D5	D4	D3	D2	D1	D0

1	D6	D5	D4	D3	D2	D1	D0

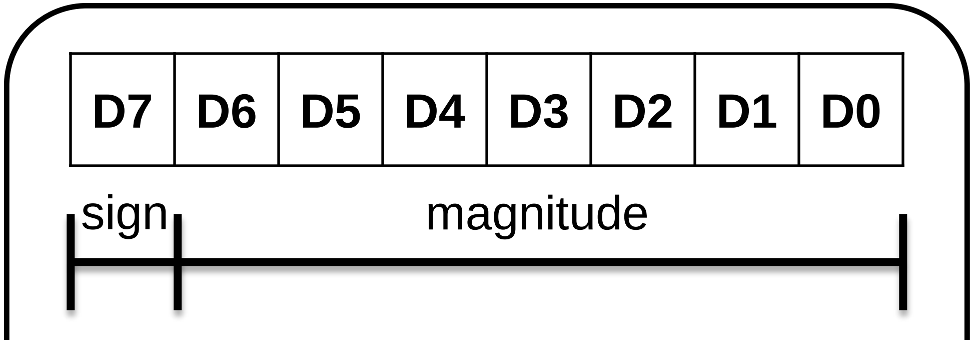
-128 … -1



8-Bit Signed Operand

Negative Numbers

- **D7 = 1** but magnitude (lower 7 bits) is represented/stored in its **2’s complement**
- Assembler does the conversion for us
  1. Write the magnitude in 8-bit binary (no sign)
  2. Invert each bit
  3. Add 1 to it



8-Bit Signed Operand

2’s Complement Example

- Represent -39 decimal in **2’s Complement**
  - Before: -0010 0111 (how we see it)

    - 39 in 8-bit binary → 0010 0111
    - Invert each bit → 1101 1000
    - Add 1 → +1
    - Final Result → 1101 1001 (0xD9)

    - Is a negative number **D7 = N = 1**

- → **0xD9** is 2’s comp. representation of **-39**

2’s Complement Example

- Represent -127 decimal in **2’s Complement**
  - Before: -0111 1111 (how we see it)
    - 127 in 8-bit binary → 0111 1111
    - Invert each bit → 1000 0000
    - Add 1 → +1
    - Final Result → 1000 0001 (0x81)
    - Is a negative number **D7 = N = 1**
- → **0x81** is 2’s comp. representation of **-127**

2’s Complement Example

- Represent -128 decimal in **2’s Complement**
  - Before: -1000 0000 (how we see it)
    - 128 in 8-bit binary → 1000 0000
    - Invert each bit → 0111 1111
    - Add 1 → +1
    - Final Result → 1000 0000 (0x80)
    - Is a negative number **D7 = N = 1**
- → **0x80** is 2’s comp. representation of **-128**

Number Ranges

Unsigned Range

Dec	Bin	Hex
0	0000 0000	00
1	0000 0001	01
2	0000 0010	02
…	…. ….	..
124	0111 1100	7C
125	0111 1101	7D
126	0111 1110	7E
127	0111 1111	7F
128	1000 0000	80
…	…. ….	..
253	1111 1101	FD
254	1111 1110	FE
255	1111 1111	FF

Signed Range

Dec	Bin	Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
…	…. ….	..
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
…	…. ….	..
+125	0111 1101	7D
+126	0111 1110	7E
+127	0111 1111	7F

Number Ranges

Unsigned Range

Dec	Bin	Hex
0	0000 0000	00
1	0000 0001	01
2	0000 0010	02
…	…. ….	..
124	0111 1100	7C
125	0111 1101	7D
126	0111 1110	7E
127	0111 1111	7F
128	1000 0000	80
…	…. ….	..
253	1111 1101	FD
254	1111 1110	FE
255	1111 1111	FF

Signed Range

Dec	Bin	Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
…	…. ….	..
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
…	…. ….	..
+125	0111 1101	7D
+126	0111 1110	7E
+127	0111 1111	7F

D7 = 1

D7 = 0

CSE@UTA

Signed Number Range

Dec	Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex
-128	10000000	80	-96	10100000	A0	-64	11000000	C0	-32	11100000	E0
-127	10000001	81	-95	10100001	A1	-63	11000001	C1	-31	11100001	E1
-126	10000010	82	-94	10100010	A2	-62	11000010	C2	-30	11100010	E2
-125	10000011	83	-93	10100011	A3	-61	11000011	C3	-29	11100011	E3
-124	10000100	84	-92	10100100	A4	-60	11000100	C4	-28	11100100	E4
-123	10000101	85	-91	10100101	A5	-59	11000101	C5	-27	11100101	E5
-122	10000110	86	-90	10100110	A6	-58	11000110	C6	-26	11100110	E6
-121	10000111	87	-89	10100111	A7	-57	11000111	C7	-25	11100111	E7
-120	10001000	88	-88	10101000	A8	-56	11001000	C8	-24	11101000	E8
-119	10001001	89	-87	10101001	A9	-55	11001001	C9	-23	11101001	E9
-118	10001010	8A	-86	10101010	AA	-54	11001010	CA	-22	11101010	EA
-117	10001011	8B	-85	10101011	AB	-53	11001011	CB	-21	11101011	EB
-116	10001100	8C	-84	10101100	AC	-52	11001100	CC	-20	11101100	EC
-115	10001101	8D	-83	10101101	AD	-51	11001101	CD	-19	11101101	ED
-114	10001110	8E	-82	10101110	AE	-50	11001110	CE	-18	11101110	EE
-113	10001111	8F	-81	10101111	AF	-49	11001111	CF	-17	11101111	EF
-112	10010000	90	-80	10110000	B0	-48	11010000	D0	-16	11110000	F0
-111	10010001	91	-79	10110001	B1	-47	11010001	D1	-15	11110001	F1
-110	10010010	92	-78	10110010	B2	-46	11010010	D2	-14	11110010	F2
-109	10010011	93	-77	10110011	B3	-45	11010011	D3	-13	11110011	F3
-108	10010100	94	-76	10110100	B4	-44	11010100	D4	-12	11110100	F4
-107	10010101	95	-75	10110101	B5	-43	11010101	D5	-11	11110101	F5
-106	10010110	96	-74	10110110	B6	-42	11010110	D6	-10	11110110	F6
-105	10010111	97	-73	10110111	B7	-41	11010111	D7	-9	11110111	F7
-104	10011000	98	-72	10111000	B8	-40	11011000	D8	-8	11111000	F8
-103	10011001	99	-71	10111001	B9	-39	11011001	D9	-7	11111001	F9
-102	10011010	9A	-70	10111010	BA	-38	11011010	DA	-6	11111010	FA
-101	10011011	9B	-69	10111011	BB	-37	11011011	DB	-5	11111011	FB
-100	10011100	9C	-68	10111100	BC	-36	11011100	DC	-4	11111100	FC
-99	10011101	9D	-67	10111101	BD	-35	11011101	DD	-3	11111101	FD
-98	10011110	9E	-66	10111110	BE	-34	11011110	DE	-2	11111110	FE
-97	10011111	9F	-65	10111111	BF	-33	11011111	DF	-1	11111111	FF

CSE@UTA

Signed Number Range

Dec	Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex	Dec	Bin	Hex
0	00000000	00	32	00100000	20	64	01000000	40	96	01100000	60
1	00000001	01	33	00100001	21	65	01000001	41	97	01100001	61
2	00000010	02	34	00100010	22	66	01000010	42	98	01100010	62
3	00000011	03	35	00100011	23	67	01000011	43	99	01100011	63
4	00000100	04	36	00100100	24	68	01000100	44	100	01100100	64
5	00000101	05	37	00100101	25	69	01000101	45	101	01100101	65
6	00000110	06	38	00100110	26	70	01000110	46	102	01100110	66
7	00000111	07	39	00100111	27	71	01000111	47	103	01100111	67
8	00001000	08	40	00101000	28	72	01001000	48	104	01101000	68
9	00001001	09	41	00101001	29	73	01001001	49	105	01101001	69
10	00001010	0A	42	00101010	2A	74	01001010	4A	106	01101010	6A
11	00001011	0B	43	00101011	2B	75	01001011	4B	107	01101011	6B
12	00001100	0C	44	00101100	2C	76	01001100	4C	108	01101100	6C
13	00001101	0D	45	00101101	2D	77	01001101	4D	109	01101101	6D
14	00001110	0E	46	00101110	2E	78	01001110	4E	110	01101110	6E
15	00001111	0F	47	00101111	2F	79	01001111	4F	111	01101111	6F
16	00010000	10	48	00110000	30	80	01010000	50	112	01110000	70
17	00010001	11	49	00110001	31	81	01010001	51	113	01110001	71
18	00010010	12	50	00110010	32	82	01010010	52	114	01110010	72
19	00010011	13	51	00110011	33	83	01010011	53	115	01110011	73
20	00010100	14	52	00110100	34	84	01010100	54	116	01110100	74
21	00010101	15	53	00110101	35	85	01010101	55	117	01110101	75
22	00010110	16	54	00110110	36	86	01010110	56	118	01110110	76
23	00010111	17	55	00110111	37	87	01010111	57	119	01110111	77
24	00011000	18	56	00111000	38	88	01011000	58	120	01111000	78
25	00011001	19	57	00111001	39	89	01011001	59	121	01111001	79
26	00011010	1A	58	00111010	3A	90	01011010	5A	122	01111010	7A
27	00011011	1B	59	00111011	3B	91	01011011	5B	123	01111011	7B
28	00011100	1C	60	00111100	3C	92	01011100	5C	124	01111100	7C
29	00011101	1D	61	00111101	3D	93	01011101	5D	125	01111101	7D
30	00011110	1E	62	00111110	3E	94	01011110	5E	126	01111110	7E
31	00011111	1F	63	00111111	3F	95	01011111	5F	127	01111111	7F

2's Complement Example
OUT OF RANGE

- Represent -129 decimal in **2's Complement**
  - Before: -1000 0001 (how we see it)
    - 129 in 8-bit binary → 1000 0001
    - Invert each bit → 0111 1110
    - Add 1 → +1
    - Final Result → 0111 1111 (0x7F)
    - **<u>NO</u>** negative bit at **D7 = N = 1**
- → **Indicates a + number (same as +127)**

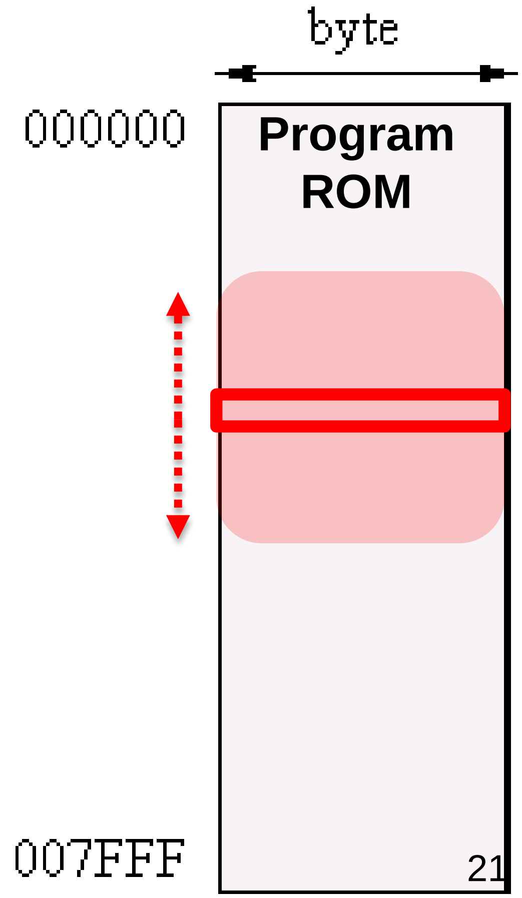
Signed Number
2's Comp. Storage

- MPLAB Example

BNZ Jump Range Explained

- Want to jump in both directions (up/down)

BNZ	**Branch if Not Zero**
Syntax:	[*label*] BNZ n
Operands:	$-128 \le n \le 127$
Operation:	if zero bit is '0' $(PC) + 2 + 2n \rightarrow PC$
Status Affected:	None
Encoding:	1110 | 0001 | nnnn | nnnn
Description:	If the Zero bit is '0', then the program will branch. The 2's complement number '2n' is added to the PC. Since the PC will have incremented to fetch the next instruction, the new address will be PC+2+2n. This instruction is then a two-cycle instruction.



Instruction Set has 5 Basic Categories

1. **Byte-Oriented Operations**
   – ADDWF, DECF, MOVWF, SUBFWB, etc.
2. **Bit-Oriented Operations**
   – BCF, BSF, BTG, BTFSC, etc.
3. **Literal Operations**
   – ADDLW, MOVLW, SUBLW, etc.
4. **Control Operations**
   – BC, BNZ, BRA, GOTO, RETURN, etc.
5. **Data Memory & Program ROM Operations**
   – TBLRD, TBLWT, etc.

Addition

- **ADDWF**
  - WREG = WREG + F or
  - F = WREG + F
- **ADDWFC**
  - WREG = WREG + F + C or
  - F = WREG + F + C
- **ADDLW**
  - WREG = WREG + L

ADDWF

ADDWF	ADD W to f
Syntax:	[*label*] ADDWF f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(W) + (f) \rightarrow dest$
Status Affected:	N, OV, C, DC, Z
Encoding:	`0010` `01da` `ffff` `ffff`
Description:	Add W to register 'f'. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the BSR is used.
Words:	1
Cycles:	1

ADDWF: W+f → dest
2 Destinations for Result

;BEFORE ;WREG = 11 ;myReg = 5 **ADDWF myReg, 0** *or* **ADDWF myReg, W** ;AFTER ;WREG = 16 ;myReg = 5	;BEFORE ;WREG = 11 ;myReg = 5 **ADDWF myReg, 1** *or* **ADDWF myReg, F** ;AFTER ;WREG = 11 ;myReg = 16

ADDWF

RAM Location		Value
0x40	=	7D
0x41	=	EB
0x42	=	C5
0x43	=	5B +
		288 hex

Store result in file registers over **two bytes**

0	0	0	0	0	0	1	0

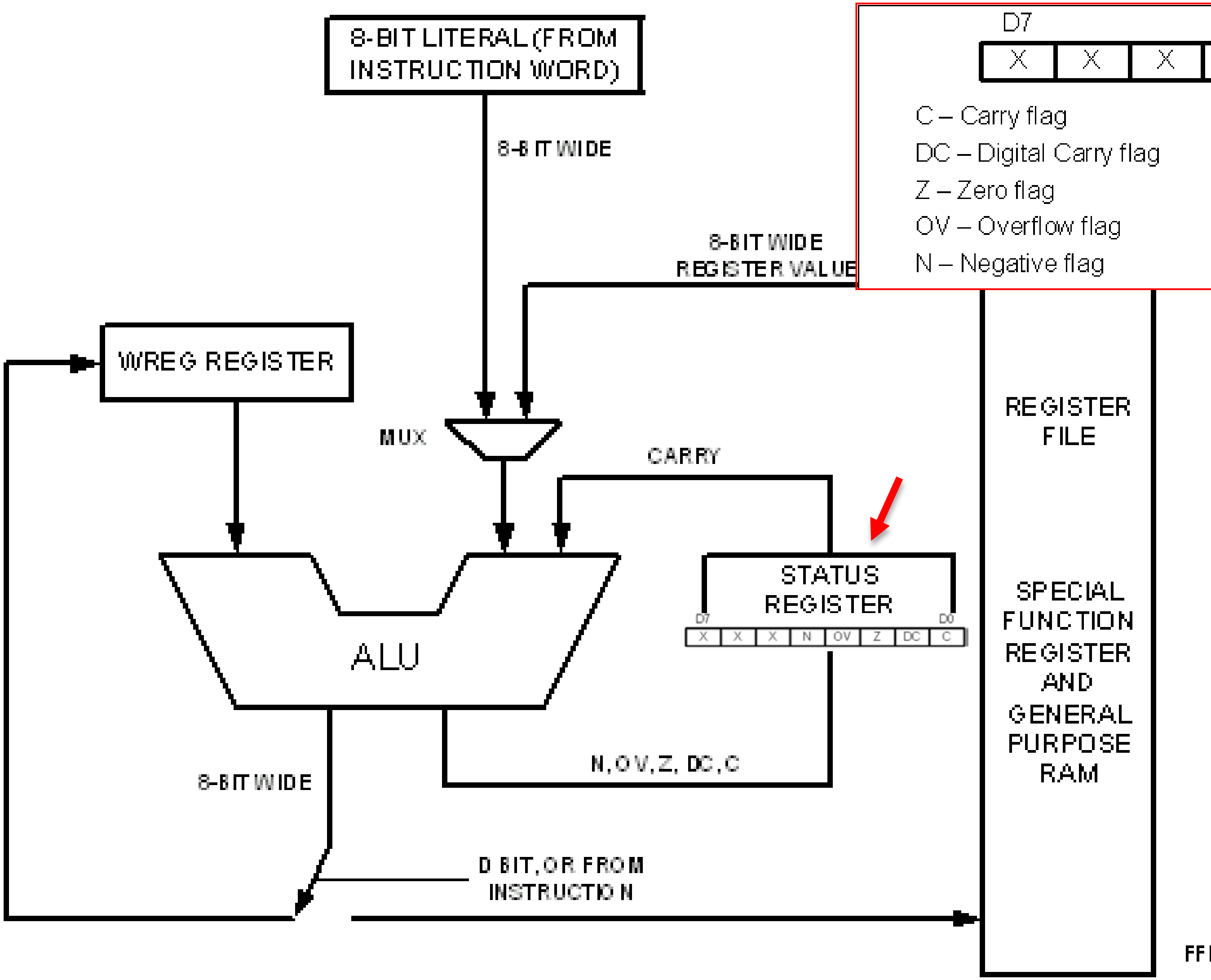
1	0	0	0	1	0	0	0

0 2 8 8

ADDWF - Carry Bit Handling

```
L_Byte EQU   0x6             ;assign RAM location 6 to L_byte of sum
H_Byte EQU   0x7             ;assign RAM location 7 to H_byte of sum

       MOVLW 0               ;clear WREG (WREG = 0)
       MOVWF H_Byte          ;H_Byte = 0
       ADDWF 0x40,W          ;WREG = 0 + 7DH = 7DH , C = 0
       BNC   N_1             ;branch if C = 0
       INCF  H_Byte,F        ;increment (now H_Byte = 0)
N_1    ADDWF 0x41,W          ;WREG = 7D + EB = 68H and C = 1
       BNC   N_2             ;
       INCF  H_Byte,F        ;C = 1, increment (now H_Byte = 1)
N_2    ADDWF 0x42,W          ;WREG = 68 + C5 = 2D and C = 1
       BNC   N_3             ;
       INCF  H_Byte          ;C = 1, increment (now H_Byte = 2)
N_3    ADDWF 0x43,W          ;WREG = 2D + 5B = 88H and C = 0
       BNC   N_4             ;
       INCF  H_Byte,F        ;(H_Byte = 2)
N_4    MOVWF L_Byte          ;now L_Byte = 88h
```

Status Register



D7							D0
X	X	X	N	OV	Z	DC	C

- C – Carry flag
- DC – Digital Carry flag
- Z – Zero flag
- OV – Overflow flag
- N – Negative flag

Carry Flag

- C Flag is set (1) when there is a "carry out" from the D7 bit
  - Can be set by an ADD or SUB

```
  1101 1010
+ 1010 1111
= 1 1000 1001
```

D7							D0
X	X	X	N	OV	Z	DC	C

ADDWFC

ADDWFC	ADD W and Carry bit to f
Syntax:	[*label*] ADDWFC f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(W) + (f) + (C) \rightarrow dest$
Status Affected:	N,OV, C, DC, Z
Encoding:	`0010` `00da` `ffff` `ffff`
Description:	Add W, the Carry Flag and data memory location 'f'. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed in data memory location 'f'. If 'a' is 0, the Access Bank will be selected. If 'a' is 1, the BSR will not be overridden.
Words:	1
Cycles:	1

ADDWFC
Ex: Adding two 16-bit numbers

```
      1
     3C  E7
+    3B  8D
     78  74
```

D7							D0
X	X	X	N	OV	Z	DC	C

```
;location 6 = (8D)
;location 7 = (3B)

MOVLW  0xE7     ;load the low byte now (WREG = E7H)
ADDWF  0x6,F    ;F = W + F = E7 + 8D = 74 and CY = 1
MOVLW  0x3C     ;load the high byte (WREG = 3CH)
ADDWFC 0x7,F    ;F = W + F + carry, adding the upper byte
                ;with Carry from lower byte
                ;F = 3C + 3B + 1 = 78H (all in hex)
```

Subtraction

- **SUBWF**
  - WREG = F – WREG or
  - F = F – WREG
- **SUBWFB**
  - WREG = F – WREG – !C or
  - F = F – WREG – !C
- **SUBFWB**
  - WREG = WREG – F – !C or
  - F = WREG – F – !C
- **SUBLW**
  - WREG = L – WREG

Subtraction

- Like ADD, there is SUB and SUB w/borrow
- SUB only converts second argument into 2’s complement and **adds** it to file register
  - 2’s complement turns addition into subtraction
  - Saves separate subtracter circuitry
- **NEG fileReg** does 2’s complement inversion

SUBLW

SUBLW	**Subtract W from literal**
Syntax:	[*label*] SUBLW k
Operands:	$0 \le k \le 255$
Operation:	$k - (W) \rightarrow W$
Status Affected:	N, OV, C, DC, Z
Encoding:	`0000` `1000` `kkkk` `kkkk`
Description:	W is subtracted from the eight-bit literal 'k'. The result is placed in W.
Words:	1
Cycles:	1

SUBLW
Result is Positive

```
MOVLW  0x23          ;load 23H into WREG (WREG = 23H)
SUBLW  0x3F          ;WREG = 3F - WREG
```

Solution:

```
   K    = 3F   0011 1111        0011 1111
 - WREG = 23   0010 0011    +   1101 1101  (2's complement)
          1C                  1 0001 1100
                          C = 1, D7 = N = 0 (result is positive)
```

The flags would be set as follows: C = 1, N = 0 (notice that D7 is the negative flag). The programmer must look at the N (or C) flag to determine if the result is positive or negative.

SUBLW
Result is Negative

Write a program to subtract 4C – 6E.

Solution:

```
MYREG EQU 0x20
      MOVLW  0x4C        ;load WREG (WREG = 4CH)
      MOVWF  MYREG       ;MYREG = 4CH
      MOVLW  0x6E        ;WREG = 6EH
      SUBWF  MYREG,W     ;WREG = MYREG - WREG. 4C - 6E = DE, N = 1
      BNN    NEXT        ;if N = 0 (C = 1), jump to NEXT target
      NEGF   WREG        ;take 2's complement of WREG
NEXT  MOVWF  MYREG       ;save the result in MYREG
```

The following are the steps after the SUBWF instruction:

```
   4C     0100 1100                  0100 1100
  -6E     0110 1110   2's comp =     1001 0010
  -22                                1101 1110
```

After SUBWF, we have N = 1 (or C = 0), and the result is negative, in 2's complement. Then it falls through and NEGF will be executed. The NEGF instruction will take the 2's complement, and we have MYREG = 22H.

SUBWFB
Need to Borrow

SUBWFB	Subtract W from f with Borrow
Syntax:	[*label*] SUBWFB f [,d [,a]
Operands:	$0 \le f \le 255$ $d \in [0,1]$ $a \in [0,1]$
Operation:	$(f) - (W) - (\overline{C}) \rightarrow dest$
Status Affected:	N, OV, C, DC, Z
Encoding:	`0101` `10da` `ffff` `ffff`
Description:	Subtract W and the carry flag (borrow) from register 'f' (2's complement method). If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f' (default). If 'a' is 0,

SUBWFB
Need to Borrow

Write a program to subtract two 16-bit numbers. The numbers are 2762H – 1296H. Assume fileReg location 6 = (62) and location 7 = (27). Place the difference in fileReg locations 6 and 7; loc 6 should have the lower byte.

0x2762	
- 0x1296	
= 0x14CC	

27	62
- 12	96
= 14	CC

Solution:

```
loc 6 = (62)
loc 7 = (27)

MOVLW  0x96          ;load the low byte (WREG = 96H)
SUBWF  0x6,F   ;F = F - W = 62 - 96 = CCH, C = borrow = 0, N = 1
MOVLW  0x12          ;load the high byte (WREG = 12H)
SUBWFB  0x7,F        ;F = F - W - b, sub byte with the borrow
                     ;F = 27 - 12 - 1 = 14H
```

After the SUBWF, loc 6 has = 62H – 96H = CCH and the carry flag is set to 0, indicating there is a borrow (notice, N = 1). Because C = 0, when SUBWFB is executed the fileReg location 7 has = 27H – 12H – 1 = 14H. Therefore, we have 2762H – 1296H = 14CCH.

Signed Number Range

Decimal	Binary	Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
. . .	. . . .	. .
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
. .	. . . .	. .
+127	0111 1111	7F

Signed Numbers

- **Overflow – OV Flag Bit**

D7							D0
X	X	X	N	OV	Z	DC	C

  - If result of an operation on **signed numbers** too large for the register
  - Not the same as carry, carry is the ninth bit
  - OV is kind of like the eighth bit (overwriting sign bit)
  - Won't throw error, programmer must handle
- More precisely OV is set if there is carry from D6 to D7 but none from D7 (positive)
  - MSB D7: 0 to 1, Positive to Negative overwriting
- Or if there is carry from D7 but none from D6 to D7
  - MSB D7: 1 to 0, Negative to Positive overwriting

Overflow Example

Example 5-13

Examine the following code and analyze the result, including the N and OV flags.

```
        MOVLW  +D'96'      ;WREG = 0110 0000
        ADDLW  +D'70'      ;WREG = (+96) + (+70) = 1010 0110
                           ;WREG = A6H = -90 decimal, INVALID!!
```

Solution:

```
      +96     0110 0000
    + +70     0100 0110
    + 166     1010 0110   N = 1 (negative) and OV = 1. Sum = -90
```

According to the CPU, the result is negative (N = 1), which is wrong. The CPU sets OV = 1 to indicate the overflow error. Remember that the N flag is the D7 bit. If N = 0, the sum is positive, but if N = 1, the sum is negative.

Overflow Example

Example 5-14

Observe the following, noting the role of the OV and N flags:

```
MOVLW -D'128'       ;WREG = 1000 0000 (WREG = 80H)
ADDLW -D'2'         ;W = (-128) + (-2)
                    ;W = 1000000 + 11111110 = 0111 1110,
                    ;N = 0, W = 7EH = +126, invalid
```

Solution:

```
  -128          1000 0000
+ - 2           1111 1110
- 130           0111 1110   N = 0 (positive) and OV = 1
```

According to the CPU, the result is +126, which is wrong, and OV = 1 indicates that.

CSE@UTA

Overflow Example

Example 5-15

Observe the following, noting the OV and N flags:

```
MOVLW -D'2'         ;WREG = 1111 1110 (WREG = FEH)
ADDLW -D'5'         ;WREG = (-2) + (-5) = -7 or F9H
                    ;correct, since OV = 0
```

Solution:

```
   -2     1111 1110
 + -5     1111 1011
 - 7      1111 1001  and OV = 0 and N = 1. Sum is negative
```

According to the CPU, the result is –7, which is correct, and the OV flag indicates that. (OV = 0).

Overflow

- **OV = 1** means result is erroneous
- **OV = 0** means result is valid
- In unsigned addition/subtraction
  - Monitor the C carry flag bit
- In signed addition/subtraction
  - Monitor the OV overflow flag bit

- How do we handle this?

C and OV Handling

- Use BC, BNC, BOV, BNOV <u>right after</u> the unsigned or signed operation
- Build "special case routines" that attempt to perform 16-bit adds/subs

```
;location 6 = (8D)
;location 7 = (3B)
```

```
       1
     3C E7
+    3B 8D
     78 74
```

```
MOVLW  0xE7       ;load the low byte now (WREG = E7H)
ADDWF  0x6,F      ;F = W + F = E7 + 8D = 74 and CY = 1
MOVLW  0x3C       ;load the high byte (WREG = 3CH)
ADDWFC 0x7,F      ;F = W + F + carry, adding the upper byte
                  ;with Carry from lower byte
                  ;F = 3C + 3B + 1 = 78H (all in hex)
```

Multiplication of Unsigned Bytes

- Bytes-only and unsigned-only operation
- **One must be WREG** & other is Literal/fileReg
- Result placed across 2 SFRs

- **MULWF f** or **MULLW k**
  - **PRODH** = high byte
  - **PRODL** = low byte

```
MOVLW 0x25          ;load 25H to WREG (WREG = 25H)
MULLW 0x65          ;25H * 65H = E99 where
                    ;PRODH = 0EH and PRODL = 99H
```

Multiplication

W x File Register

MULWF — **Multiply W with f**

Syntax:	[*label*] MULWF f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(W) x (f) → PRODH:PRODL
Status Affected:	None
Encoding:	0000 | 001a | ffff | ffff
Description:	An unsigned multiplication is carried out between the contents of W and the register file location 'f'. The 16-bit result is stored in the PRODH:PRODL register pair. PRODH contains the high byte. Both W and 'f' are unchanged.

W x Literal

MULLW — **Multiply Literal with W**

Syntax:	[*label*] MULLW k
Operands:	$0 \le k \le 255$
Operation:	(W) x k → PRODH:PRODL
Status Affected:	None
Encoding:	0000 | 1101 | kkkk | kkkk
Description:	An unsigned multiplication is carried out between the contents of W and the 8-bit literal 'k'. The 16-bit result is placed in PRODH:PRODL register pair. PRODH contains the high byte. W is unchanged.

MAX Bit-Width for Result

- For an **ADD** of two 8-bit numbers
  - MAX result is 9 bits wide

- For a **MULT** of two 8-bit numbers
  - MAX result is 16 bits wide

MAX Bit-Width for Result
ADDITION

0001 1001	(0x19)
+ 0000 1111	(0x0F)
= 0 0010 1000	(0x28)

1101 1010	(0xDA)
+ 1010 1111	(0xAF)
= 1 1000 1001	(0x189)

1111 1111	(0xFF)
+ 1111 1111	(0xFF)
= 1 1111 1110	(0x1FE)

Single CARRY bit works just fine

MAX Bit-Width for Result
MULTIPLICATION

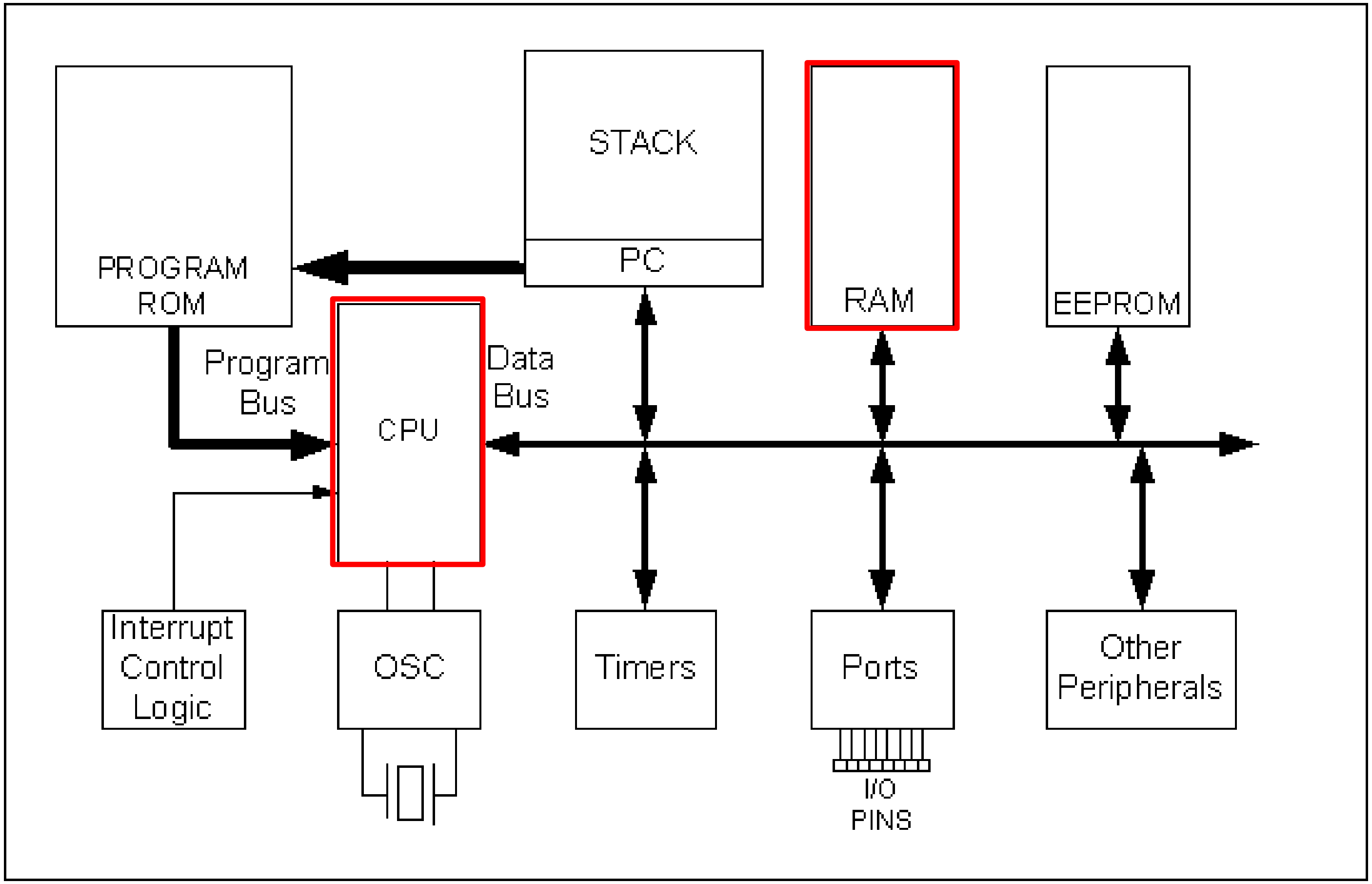
0010 1001	(0x29)
x 0000 1111	(0x0F)
= 0000 0010 0110 0111	(0x267)

1111 1111	(0xFF)
x 1111 1111	(0xFF)
= 1111 1110 0000 0001	(0xFE01)

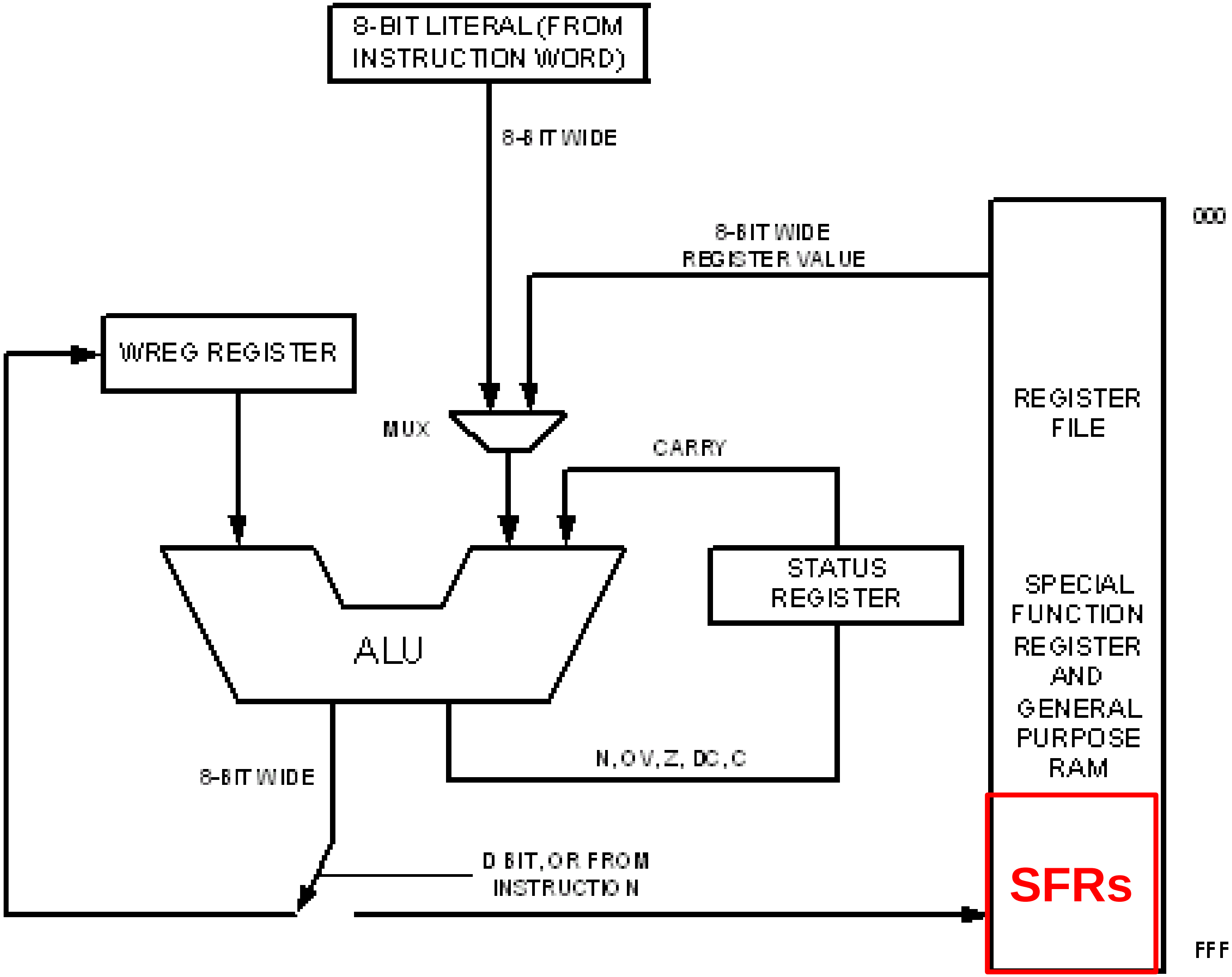
Single CARRY bit is not enough

Need two dedicated extra registers

PRODH & PRODL Stored in the File Register



PRODH & PRODL Located in the SFRs



PRODH & PRODL Stored in the File Register

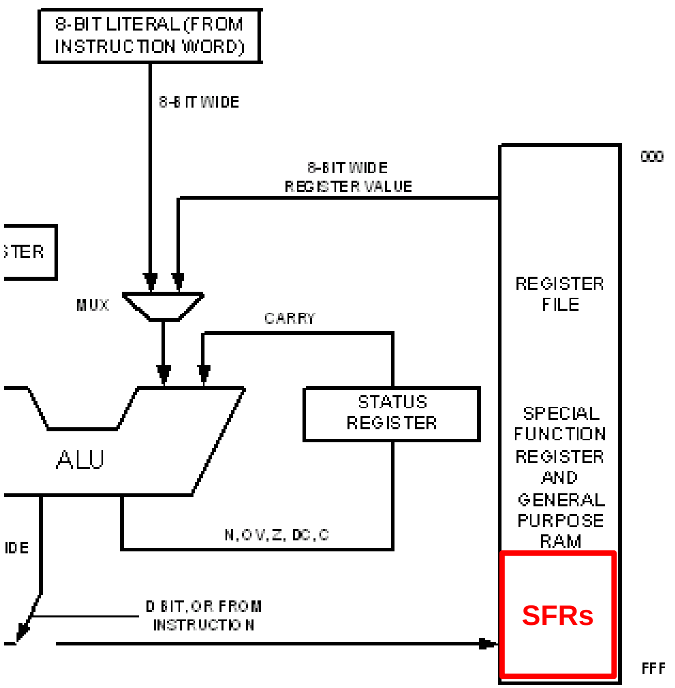


TABLE 4-1: SPECIAL FUNCTION REGISTER MAP

Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2[3]	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2[3]	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2[3]	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2[3]	FBCh	CCPR2H	F9Ch	—
FFBh	PCLATU	FDBh	PLUSW2[3]	FBBh	CCPR2L	F9Bh	—
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—
FF9h	PCL	FD9h	FSR2L	FB9h	—	F99h	—
FF8h	TBLPTRU	FD8h	STATUS	FB8h	—	F98h	—
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	—	F97h	—
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	—	F96h	TRISE[2]
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD[2]
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	LVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—
FF0h	INTCON3	FD0h	RCON	FB0h	—	F90h	—
FEFh	INDF0[3]	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—
FEEh	POSTINC0[3]	FCEh	TMR1L	FAEh	RCREG	F8Eh	—
FEDh	POSTDEC0[3]	FCDh	T1CON	FADh	TXREG	F8Dh	LATE[2]
FECh	PREINC0[3]	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD[2]
FEBh	PLUSW0[3]	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—
FE7h	INDF1[3]	FC7h	SSPSTAT	FA7h	EECON2	F87h	—
FE6h	POSTINC1[3]	FC6h	SSPCON1	FA6h	EECON1	F86h	—
FE5h	POSTDEC1[3]	FC5h	SSPCON2	FA5h	—	F85h	—
FE4h	PREINC1[3]	FC4h	ADRESH	FA4h	—	F84h	PORTE[2]
FE3h	PLUSW1[3]	FC3h	ADRESL	FA3h	—	F83h	PORTD[2]
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	—	FA0h	PIE2	F80h	PORTA

MULLW Example

$$
\begin{array}{rl}
0010\ 1001 & (0x29) \\
\underline{\times\ 0000\ 1111} & (0x0F) \\
=0000\ 0010\ 0110\ 0111 & (0x267)
\end{array}
$$

MOVLW 0x29

MULLW 0x0F

;W x k → PRODH:PRODL

FF7h	TBLPTRH	
FF6h	TBLPTRL	
FF5h	TABLAT	
FF4h	PRODH	**0x02**
FF3h	PRODL	**0x67**
FF2h	INTCON	

MULLW .ASM Example

- MPLAB Example

Division of Unsigned Numbers

- No single instruction for division of byte/byte numbers in PIC18
- Must use repeated subtraction
- Our “human algorithm” to divide numbers
  - we subtract the denominator from the numerator as many times as we can

Ex: 95 ÷ 10 = 9.5

```
NUM     EQU     0x19            ;set aside fileReg
MYQ     EQU     0x20
MYNMB   EQU     D'95'
MYDEN   EQU     D'10'
        CLRF    MYQ             ;quotient = 0
        MOVLW   MYNMB           ;WREG = 95
        MOVWF   NUM             ;numerator = 95
        MOVLW   MYDEN           ;WREG = denominator = 10
B1      INCF    MYQ,F           ;increment quotient for every 10 subtr
        SUBWF   NUM,F           ;subtract 10 (F = F - W)
        BC      B1              ;keep doing it until C = 0
        DECF    MYQ,F           ;once too many
        ADDWF   NUM,F           ;add 10 back to get remainder
```

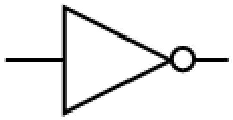
Division .ASM Example

- [MPLAB Example]

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
LITERAL OPERATIONS									
ADDLW	k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move literal (12-bit) 2nd word	2	1110	1110	00ff	kkkk	None	
		to FSRx 1st word		1111	0000	kkkk	kkkk		
MOVLB	k	Move literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW	k	Move literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000	1101	kkkk	kkkk	None	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	

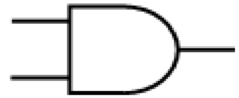
Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da0	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	0da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2

Boolean Logic



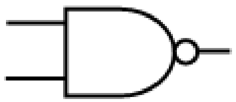
NOT

Input	Output
I	F
0	1
1	0



AND

Inputs		Output
A	B	F
0	0	0
1	0	0
0	1	0
1	1	1



NAND

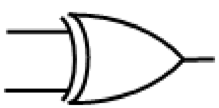
Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0

OR

Inputs		Output
A	B	F
0	0	0
1	0	1
0	1	1
1	1	1

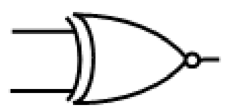
NOR

Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0



Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE OR



EXCLUSIVE NOR

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Logic Instructions

• ANDLW	k	ANDWF	f
• IORLW	k	IORWF	f
• XORLW	k	XORWF	f
• COMF	f	NEG	f

- For Masking, Querying, Toggling

ANDLW k ANDWF f
Masking bits to 0

Logical AND Function

Inputs		Output
X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1


```
MOVLW  0x35          ;WREG = 35H
ANDLW  0x0F          ;W = W AND 0FH (now W = 05)
```

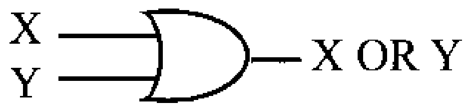
Solution:

```
35H   0 0 1 1 0 1 0 1
0FH   0 0 0 0 1 1 1 1
05H   0 0 0 0 0 1 0 1    ;35H AND 0FH = 05H, Z = 0, N = 0
```

IORLW k IORWF f
Masking bits to 1

Logical OR Function

Inputs		Output
X	**Y**	**X OR Y**
0	0	0
0	1	1
1	0	1
1	1	1




```
MOVLW  0x04                ;WREG = 04
IORLW  0x30                ;now WREG = 34H
```

```
04H       0000 0100
30H       0011 0000
34H       0011 0100       04 OR 30 = 34H, Z = 0 and N = 0
```

Masking

Masked **ON**

"OR" with 1

```
   10010101
OR 11110000
 = 11110101
```

"Forced" to 1 Unchanged

Masked **OFF**

"AND" with 0

```
    10010101
AND 00001111
  = 00000101
```

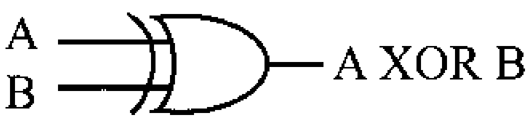
"Forced" to 0 Unchanged

XORLW k XORWF f
Toggling Bits

```
      10011101    10010101
XOR   00001111    11111111
  =   10010010    01101010
```

Logical XOR Function

Inputs		Output
A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0




```
MOVLW  0x54
XORLW  0x78
```

```
54H        0 1 0 1 0 1 0 0
78H        0 1 1 1 1 0 0 0
2CH        0 0 1 0 1 1 0 0    54H XOR 78H = 2CH, Z = 0, N = 0
```

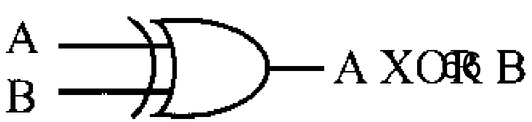
XORLW k XORWF f
Querying Bits

```
OVER   MOVF   PORTB,W     ;get a byte from PORTB into WREG
       XORLW  0x45
       BNZ    OVER        ;branch if not zero
```

```
 45H          01000101
 45H          01000101
 --------------------
 00           00000000
```

Logical XOR Function

Inputs		Output
A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0



COMF f

- Complement file register (1's complement)

```
CLRF   TRISB          ;Port B = Output
MOVLW  0x55
MOVWF  PORTB
COMF   PORTB,F        ;now PORTB = AAH
```

Logical Inverter

Input	Output
X	**NOT X**
0	1
1	0

0x55 0101 0101

0xAA 1010 1010

NEGF f

- Negate file register (2's complement)

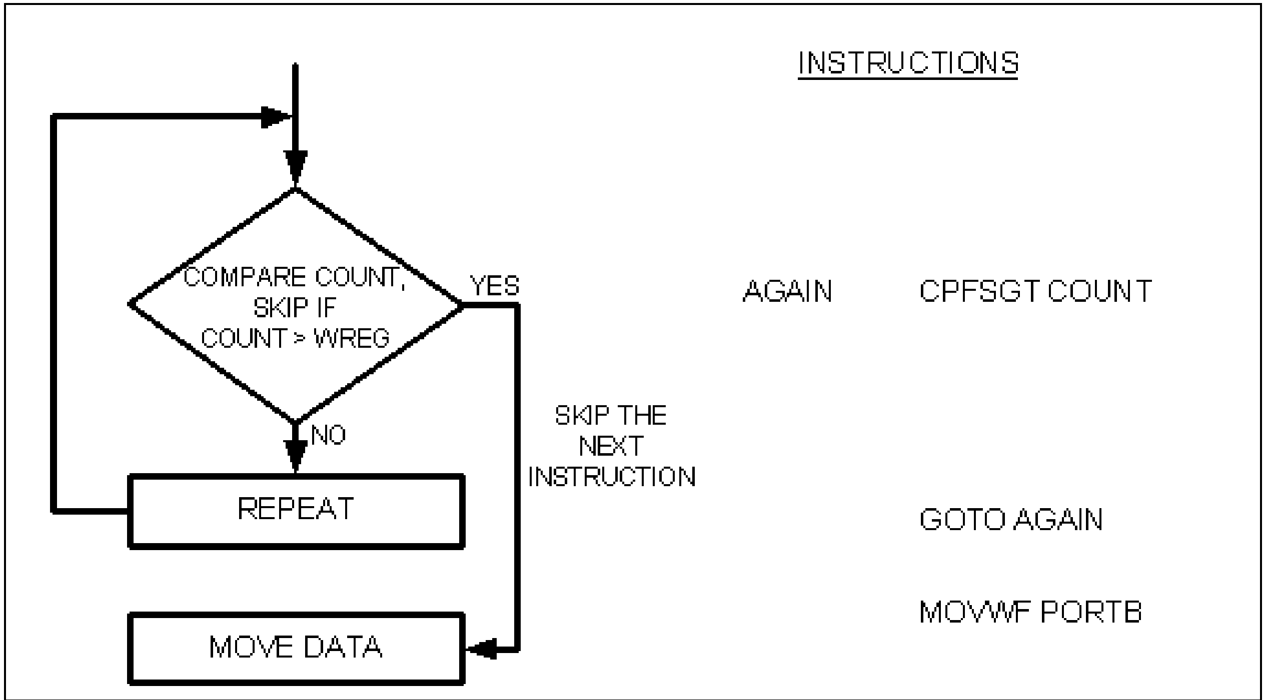
Find the 2's complement of the value 85H. Note that 85H is –123.

Solution:

```
MYREG  EQU  0x10
       MOVLW  0x85                    85H = 1000 0101
       MOVWF  MYREG                   1'S = 0111 1010
       NEGF   MYREG                              + 1
                                      -----------------
                        2's comp        0111 1011  =  7BH
```

Compare Instructions

- We have seen similar “skip-next” instructions before
- **CPFSGT fileReg** ;Skip if fileReg > WREG
- **CPFSEQ fileReg** ;Skip if fileReg = WREG
- **CPFSLT fileReg** ;Skip if fileReg < WREG



f > WREG

Write a program to find the greater of the two values 27 and 54, and place it in file register location 0x20.

Solution:

```
VAL_1  EQU   D'27'
VAL_2  EQU   D'54'
GREG   EQU   0x20

MOVLW  VAL_1        ;WREG = 27
MOVWF  GREG         ;GREG = 27
MOVLW  VAL_2        ;WREG = 54
CPFSGT GREG         ;skip if GREG > WREG
MOVWF  GREG         ;place the greater in GREG
```

f < WREG

Write a program to find the smaller of the two values 27 and 54, and place it in file register location 0x20.

Solution:

```
VAL_1 EQU   D'27'
VAL_2 EQU   D'54'
LREG  EQU   0x20   ;location for smaller of two

MOVLW   VAL_1      ;WREG = 27
MOVWF   LREG       ;LREG = 27
MOVLW   VAL_2      ;WREG = 54
CPFSLT  LREG       ;skip if LREG < WREG
MOVWF   LREG       ;place the smaller value in LREG
```

f == WREG

Write a program to monitor PORTD continuously for the value 63H. It should stop monitoring only if PORTD = 63H.

Solution:

```
        SETF    TRISD       ;PORTD = input
        MOVLW   0x63        ;WREG = 63H
BACK    CPFSEQ  PORTD       ;skip BRA instruction if PORTD = 63H
        BRA     BACK
        .....
```

Assume that Port D is an input port connected to a temperature sensor. Write a program to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following.

If T = 75	then WREG = 75
If T > 75	then GREG = T
If T < 75	then LREG = T

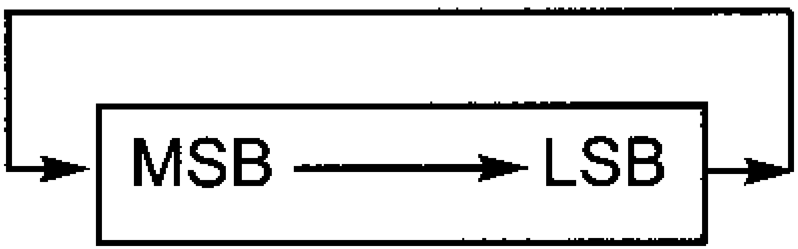
```
LREG EQU 0x20
GREG EQU 0x21
       SETF     TRISD              ;PORTD = input
       MOVLW    D'75'              ;WREG = 75 decimal
       CPFSGT   PORTD              ;skip BRA instruction if PORTD > 75
       BRA      LEQ
       MOVFF    PORTD, GREG
       BRA      OVER
LEQ    CPFSLT   PORTD              ;skip if PORTD < 75
       BRA      OVER
       MOVFF    PORTD, LREG
OVER   .....                       ;it must be equal, WREG = 75
```

Rotate and Swap Nibbles

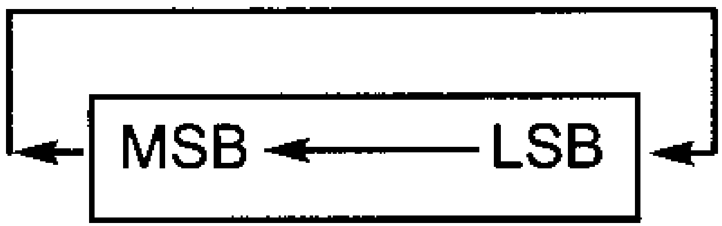
- **RRNCF f** ;rotate right one bit
- **RLNCF f** ;rotate left one bit
- **RRCF f** ;rotate right one through carry
- **RLCF f** ;rotate left one through carry
- **SWAPF f** ;swaps upper and lower nibbles (4 bits)

Visualized

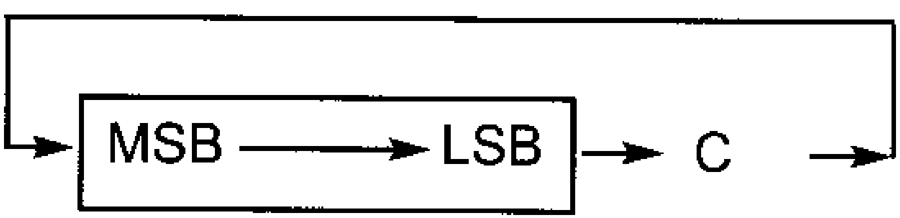
RRNCF f



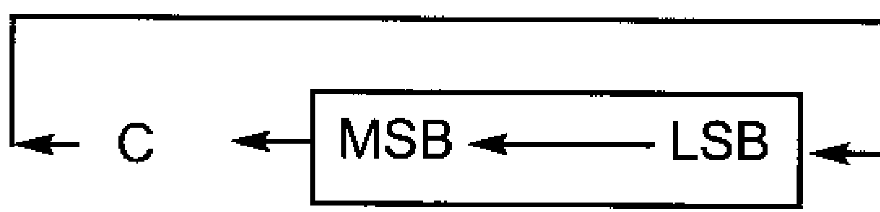
RLNCF f



RRCF f



RLCF f



Data Serialization

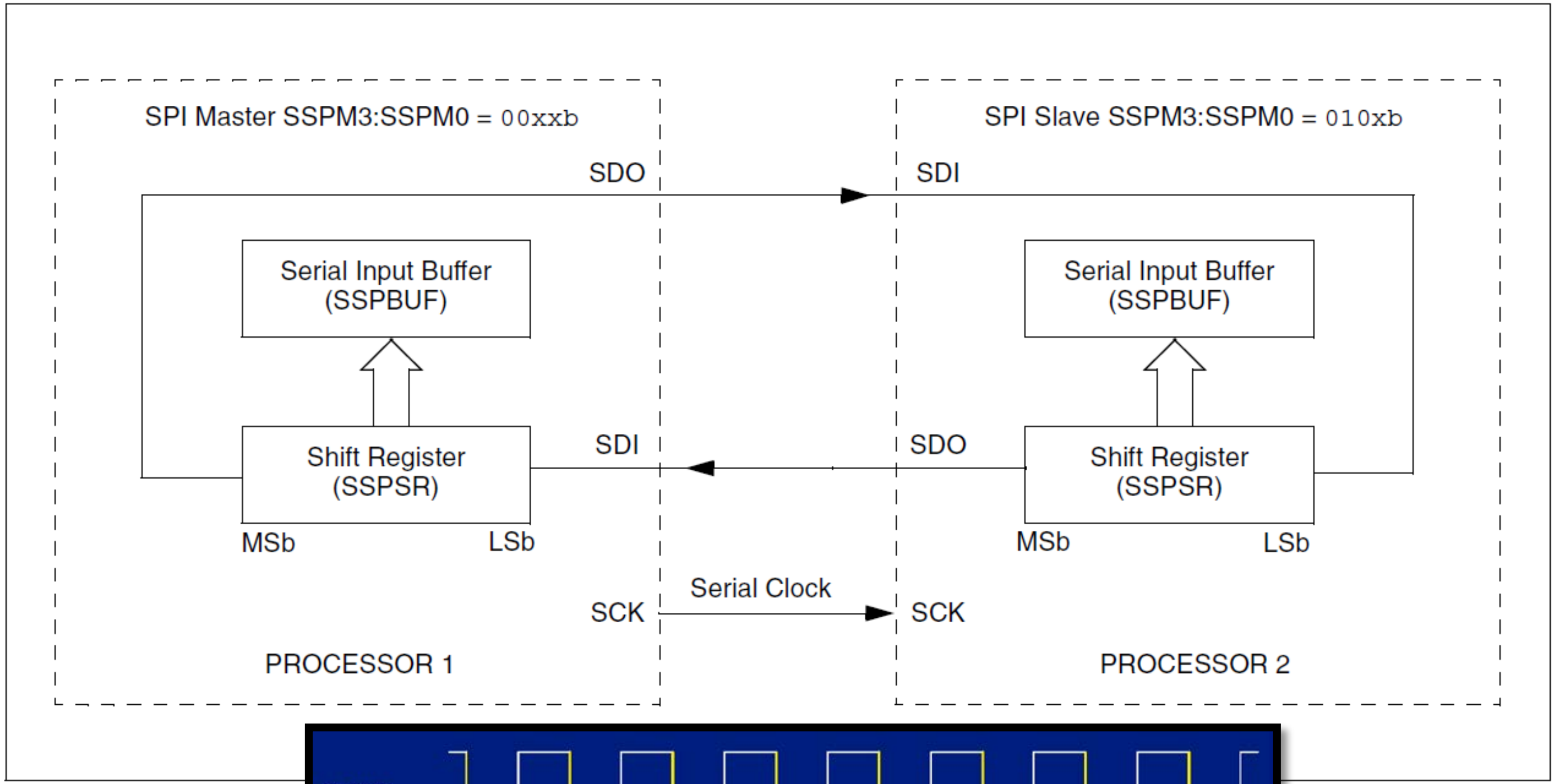
Want to send 0x41 (0100 0001) serially (1-bit at a time) via Pin RB1. Send LSB first. Start and end with a HIGH (1), to “wrap” the data.

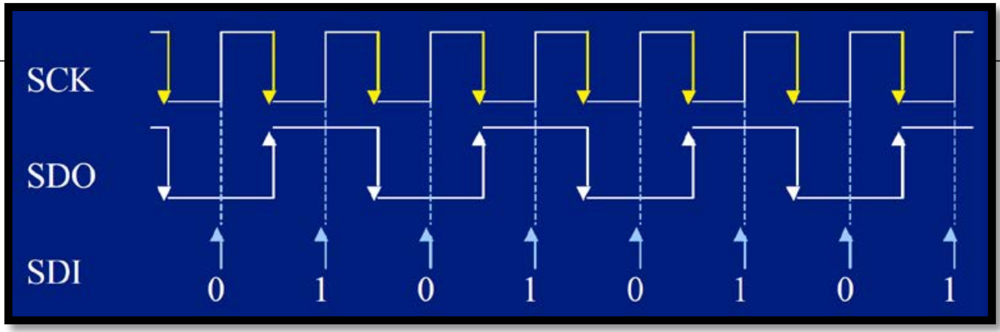
```
        Counter  EQU       0x20          ; counter variable
        Myreg    EQU       0x21          ; variable for serialization buffer

        BCF      TRISB,    1             ; RB1 is output
        MOVLW    0x41                    ; we will serialize this value
        MOVWF    Myreg                   ; load value into myreg
        BCF      STATUS,C                ; clear carry
        MOVLW    0x08                    ; counter
        MOVWF    Counter                 ; load the counter
        BSF      PORTB,    1             ; initialize serialization bit (START)
AGAIN   RRCF      Myreg,   F             ; rotate right with carry
        BNC      OVER
        BSF      PORTB,    1
        BRA      NEXT
OVER    BCF      PORTB,    1
NEXT    DECF     Counter,  F
        BNZ      AGAIN
        BSF      PORTB,    1             ; final high serialization bit (END)
```

MSSP SPI Mode

The SPI mode allows 8-bits of data to be synchronously transmitted and received, simultaneously. All four modes of SPI are supported. To accomplish communication, typically three pins are used:

FIGURE 15-2: SPI MASTER/SLAVE CONNECTION





C → ASM

- MPLAB can show the assembly code generated from your C code
- Build and Debug .c file (stop or continue)
  - Window > Debugging > Output > Disassembly Listing File

- Common C techniques (loops, ifs, elses, etc.) are implemented using many of these logic instructions

Questions?

- For more Arithmetic/Logic details
  - Textbook sections 5.1, 5.2, 5.3, and 5.4
  - 5.5 not covered
- Banks and Tables covered next
  - Textbook Ch. 6