The University of Texas at Arlington

Exam #1 Review

Exam Date: February 27th, 2018

CSE 3442/5442

Embedded Systems 1

Materials

- No calculator
- No scantron
- No scratch paper
- No cheat sheet

- Only need pen and pencil/eraser (preferred)

Exam Format

- 50-60 questions (100 points total)
  - True/False
  - Multiple Choice
    - Some multiple response
  - Fill-in-the-blank
  - Write/Read Assembly Code
  - Text Free Response
  - Math Free Response
- 2 Bonus Questions

Topics Covered

- Lectures 1 – 8
- PIC18F452 Datasheet
  - *PIC18FXX2 Data Sheet.pdf*
- Textbook
  - *PIC Microcontroller and Embedded Systems - Mazidi, Mckinlay, and Causey*
- PIC18Fxxx Instruction Set
  - http://technology.niagarac.on.ca/staff/mboldin/18F_Instruction_Set/

Topics Covered

Topic	Lecture #	Textbook Ch.Sec	Datasheet Sec
PIC18 Overview	1, 2	1.2	1.0
Architecture	2	2	1.0
Assembly	3, 4, 5, 7, 8	2	20.0
Branching	4	3, 4.2	-
Instruction Cycle Time & Delay	4, 7.0	3.3, 7.1	-
Digital I/O	5	4	1.0, 9.0
Arithmetic & Logic	7	5 (no BCD)	4.13
C Programming	6	7.1, 7.2, 7.3, 7.6, 7.7	-
Addressing	8	6.1, 6.2	4.10, 4.11, 4.12
Banks	8	6.5	4.10, 4.11
Tables	8	6.3	4.8
Memory	8	7.6, 7.7	4.0, 4.1, 4.4, 4.7, 4.9
Macros & Modules	8	6.7	-
Also Know: 2's Complement & Converting between hex, binary, and decimal			
Not Covered: BCD, LCD, MPLAB, QwikFlash, or MCLR wiring/connections			

Base/Radix
"Has X Unique Symbols"

2 Binary
0, 1

10 Decimal
0 – 9

16 Hexadecimal
0 – 9, A – F

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Binary to Hex
"Split into 4s"

0b110101011011

110101011011

1101	0101	1011	bin
13	5	11	dec
D	5	B	hex

→ 0xD5B or D5BH

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Binary to Hex
"Split into 4s"

0b1110011111

1110011111

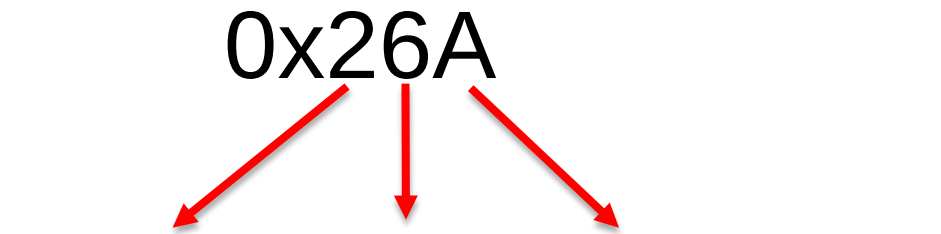
001110011111

0011	1001	1111	bin
3	9	15	dec
3	9	F	hex

→ 0x39F or 39FH

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Hex to Binary
"Expand to 4s"



2	6	A	hex
10	110	1010	bin
0010	0110	1010	bin

→ 0b1001101010

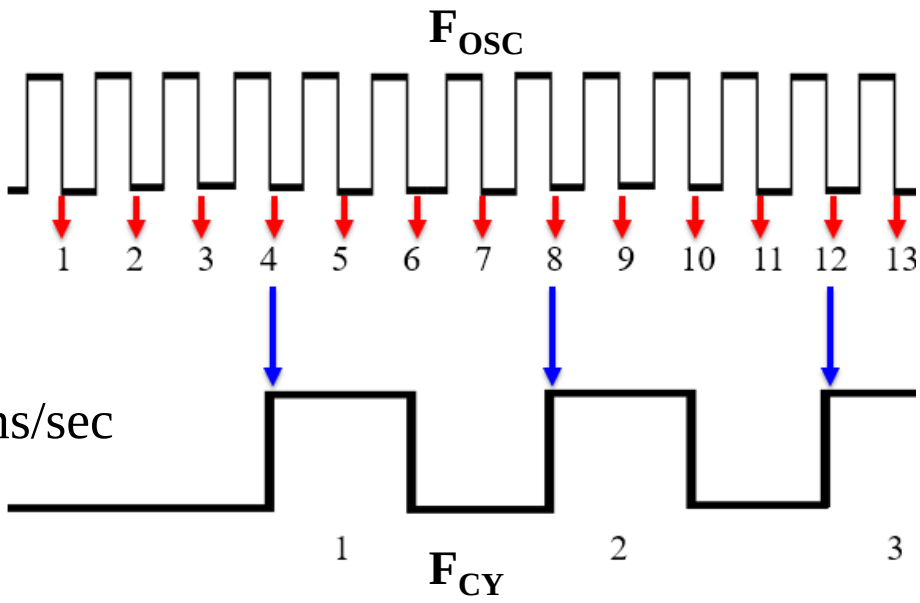
Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Instruction Cycle

F_{OSC} = Oscillator Frequency

= 10 MHz = 10,000,000 cycles/sec

Each instruction takes 4 clock cycles (ticks)



F_{CY} = Instruction Cycle Frequency

$$= \frac{F_{OSC}}{4} = \frac{10MHZ}{4} = 2.5\text{MHz} = 2{,}500{,}000 \text{ Ins/sec}$$

T_{CY} = Instruction Cycle Time

$$= \frac{1}{F_{CY}} = \frac{1}{2.5\text{MHz}} = 0.0000004 \text{ sec per Ins}$$

$$= 0.0004 \text{ ms} = 0.4 \text{ µs}$$

How many IC (instructions) fit into 1ms?

1ms / 0.0004ms = 2,500

→ 2,500 Instruction Cycles take place in 1ms

→ 2,500 Instructions can complete in 1ms (generalizing since most instructions only take 1 Ins. Cycle)

What is a register?

- **Register**
  - A place inside the PIC that can be written to, read from, or both (8-bit numbers)

TABLE 9-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on All Other RESETS
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
LATB	LATB Data Output Register								xxxx xxxx	uuuu uuuu
TRISB	PORTB Data Direction Register								1111 1111	1111 1111
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
INTCON2	RBPU	INTEDG0	INTEDG1	INTEDG2	—	TMR0IP	—	RBIP	1111 -1-1	1111 -1-1
INTCON3	INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF	11-0 0-00	11-0 0-00

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

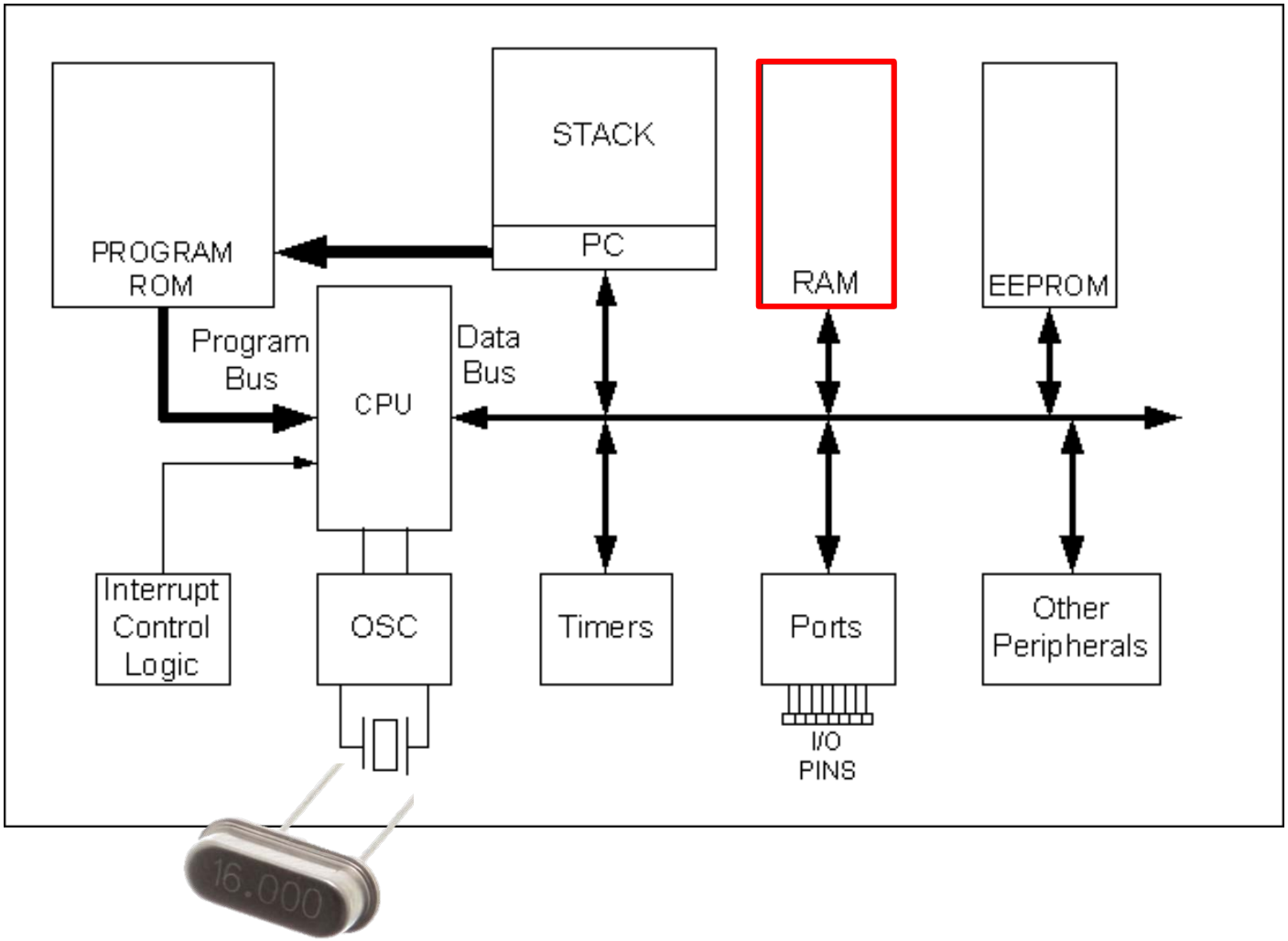
WREG

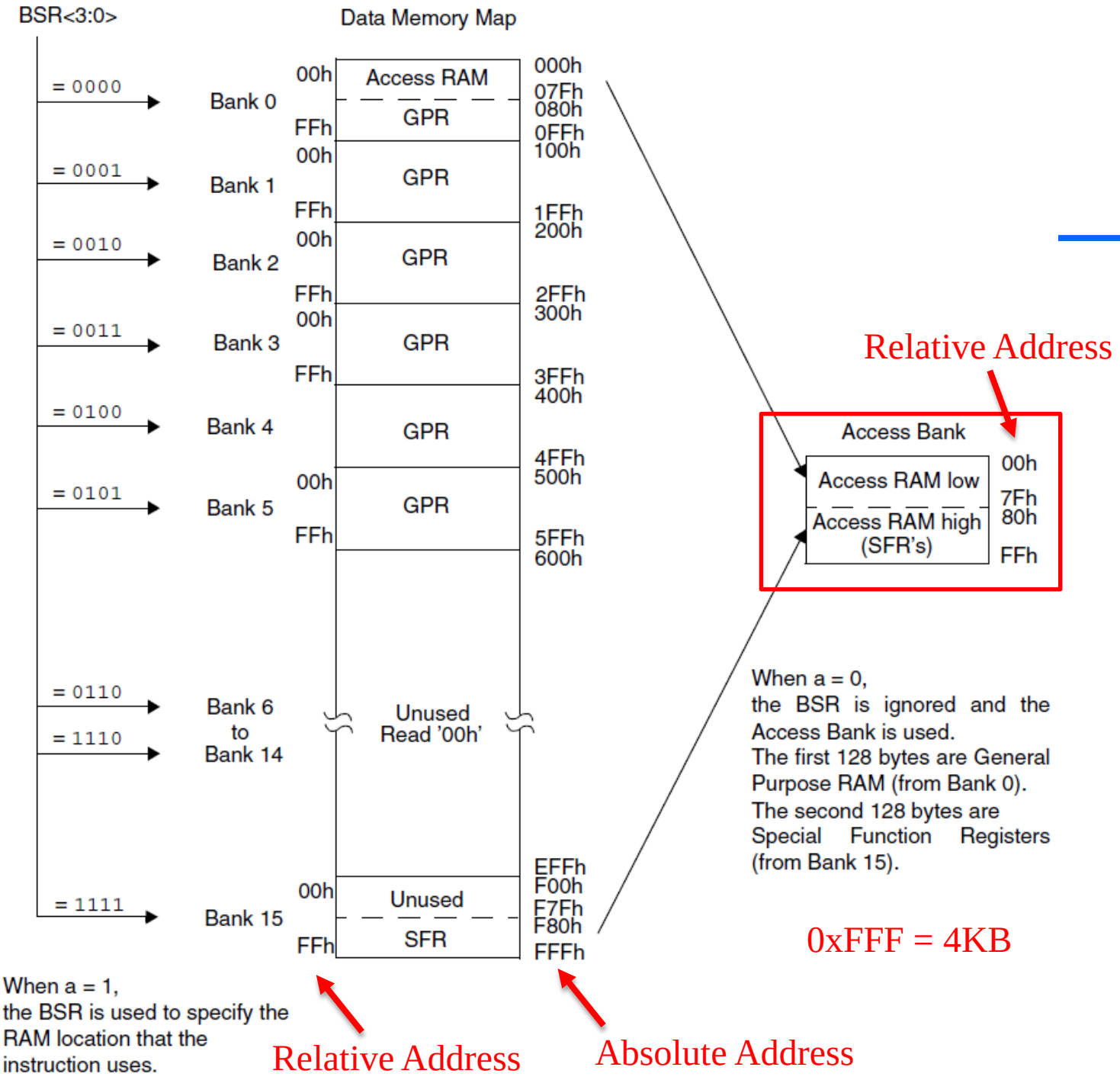
- **W**orking **Reg**ister is the same as the **accumulator** in other microprocessors
- Used for all arithmetic/logic instructions
  - Avoids use of main memory
  - Close as possible to the ALU within the CPU
- **Can only hold 0-255dec (0-FFh)**
  - Truncates larger values and cause warning
    - 1001 1101 1100 = 2,524dec = 9DCh
    - 0000 1101 1100 = 220dec = DCh

FILE REGISTER

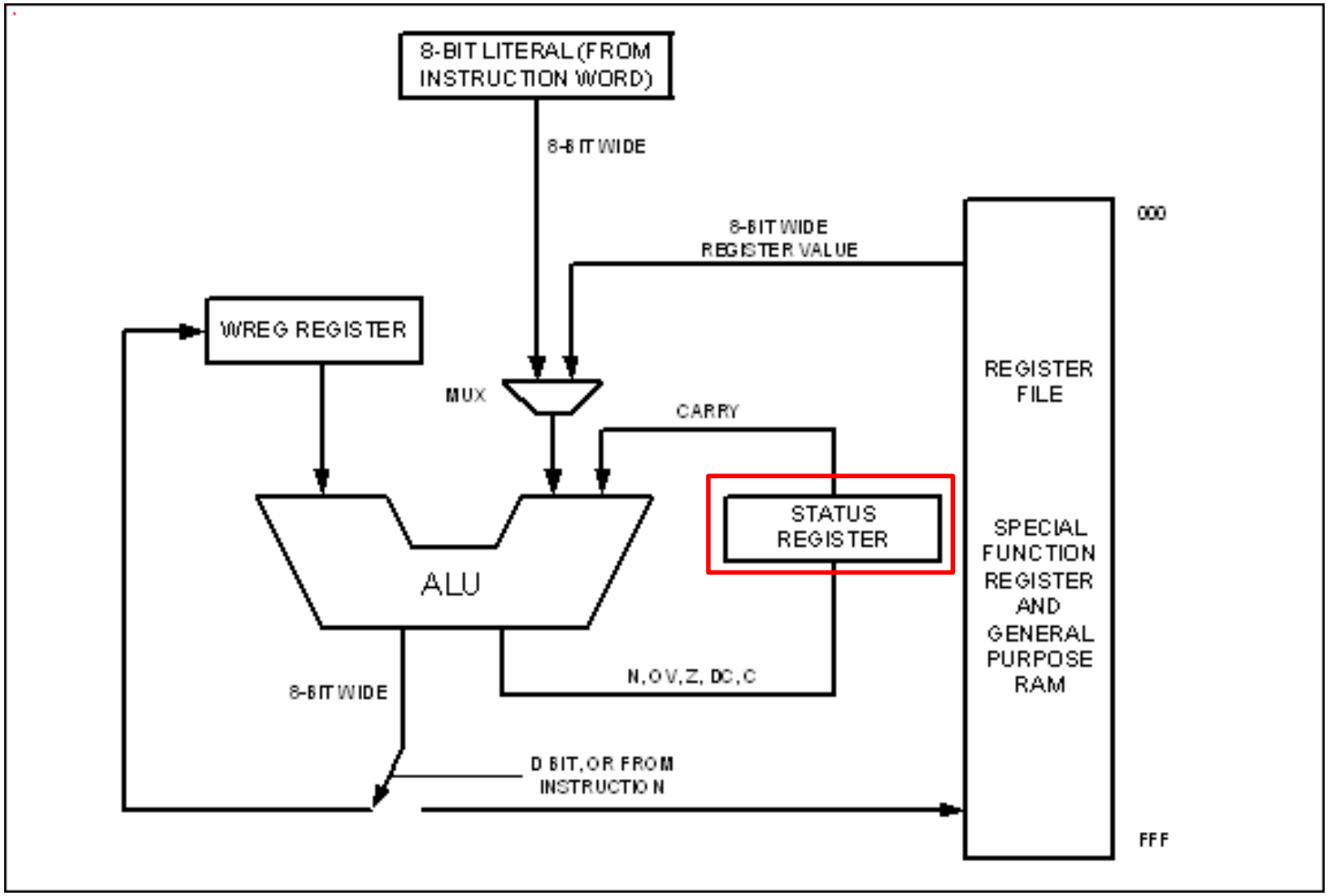
- **File Register = Data Memory (RAM)**
  - Read/write memory used by CPU for data storage
  - Varies from 32 bytes to thousands depending on chip size (family)
  - Can perform arithmetic/logic operations on many locations of File Register data
- Divided into two sections:
  - Special Function Registers (**SFR**)
  - General Purpose Registers (**GPR**) or (GP RAM)

File Registers = Data RAM

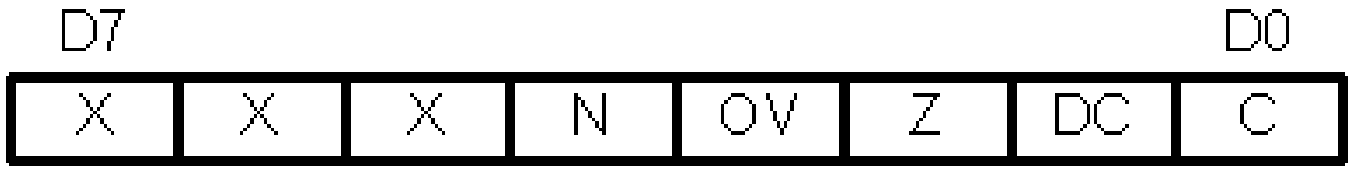



BSR<3:0>
Data Memory Map
= 0000
Bank 0
00h
Access RAM
GPR
FFh
000h
07Fh
080h
0FFh
= 0001
Bank 1
00h
GPR
FFh
100h
1FFh
= 0010
Bank 2
00h
GPR
FFh
200h
2FFh
= 0011
Bank 3
00h
GPR
FFh
300h
3FFh
= 0100
Bank 4
GPR
400h
4FFh
= 0101
Bank 5
00h
GPR
FFh
500h
5FFh
600h
= 0110
= 1110
Bank 6
to
Bank 14
Unused
Read '00h'
= 1111
Bank 15
00h
Unused
SFR
FFh
EFFh
F00h
F7Fh
F80h
FFFh
Relative Address
Access Bank
Access RAM low
Access RAM high
(SFR's)
00h
7Fh
80h
FFh
When a = 0,
the BSR is ignored and the Access Bank is used.
The first 128 bytes are General Purpose RAM (from Bank 0).
The second 128 bytes are Special Function Registers (from Bank 15).
0xFFF = 4KB
When a = 1,
the BSR is used to specify the RAM location that the instruction uses.
Relative Address
Absolute Address

Status Register in the CPU



Status Register
Only the 5 LSBs are Used



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

Flag Bits used in Branching

- BC — Branch if C = 1 (carry, positive)
- BNC — Branch if C ≠ 1
- BZ — Branch if Z = 1 (zero)
- BNZ — Branch if Z ≠ 1
- BN — Branch if N = 1 (negative)
- BNN — Branch if N ≠ 1
- BOV — Branch if OV = 1 (overflow, 2s cmp)
- BNOV — Branch if OV ≠ 1

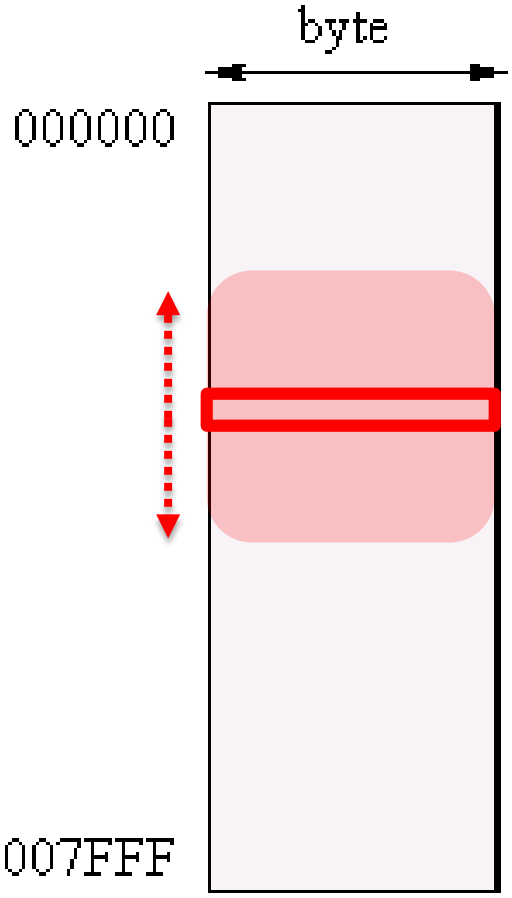
Other Conditional Branch Instructions

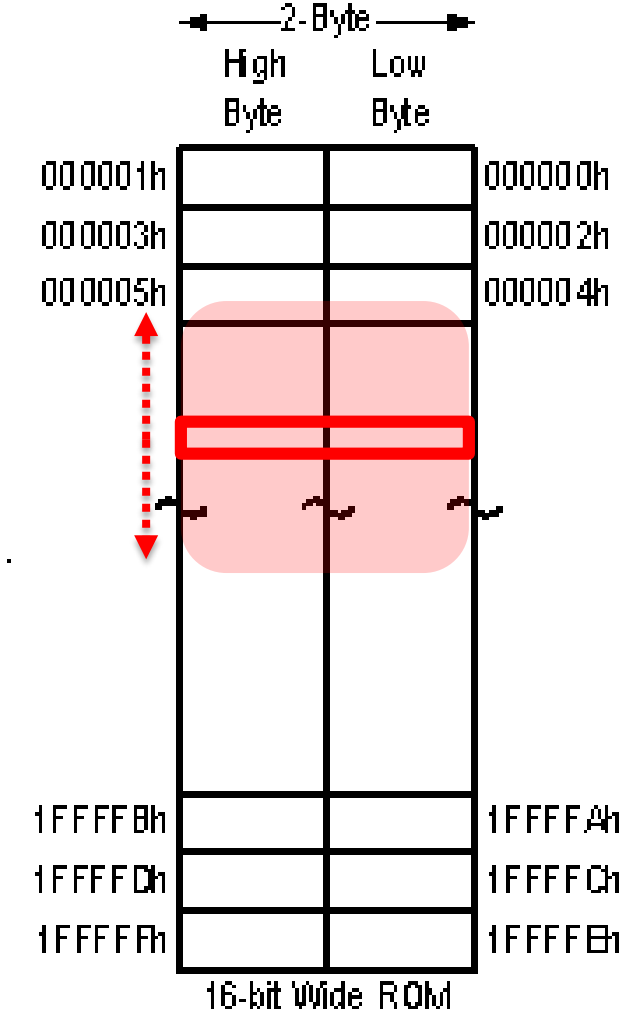
BC	**(C=1)**
BNC	
BZ	**(Z=1)**
BNZ	
BN	**(N=1)**
BNN	
BOV	**(OV=1)**
BNOV	

BC	**Branch if Carry**
Syntax:	[*label*] BC n
Operands:	-128 ≤ n ≤ 127
Operation:	if carry bit is '1' (PC) + 2 + 2n → PC
Status Affected:	None
Encoding:	1110 | 0010 | nnnn | nnnn
Description:	If the Carry bit is '1', 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.
Words:	1
Cycles:	1(2)

Note: (All conditional branches are 2 bytes thus represent short jumps, within ~ +/-128 bits to PC)

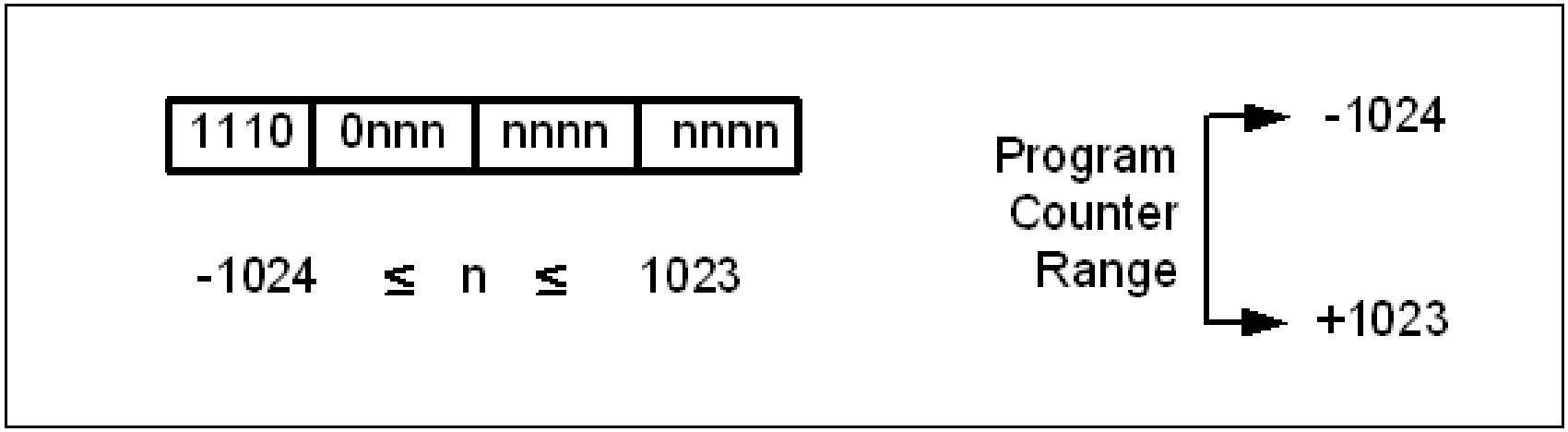
Jumping Range in ROM for Conditional Branches





BRA (Branch Unconditionally) Instruction Address Range

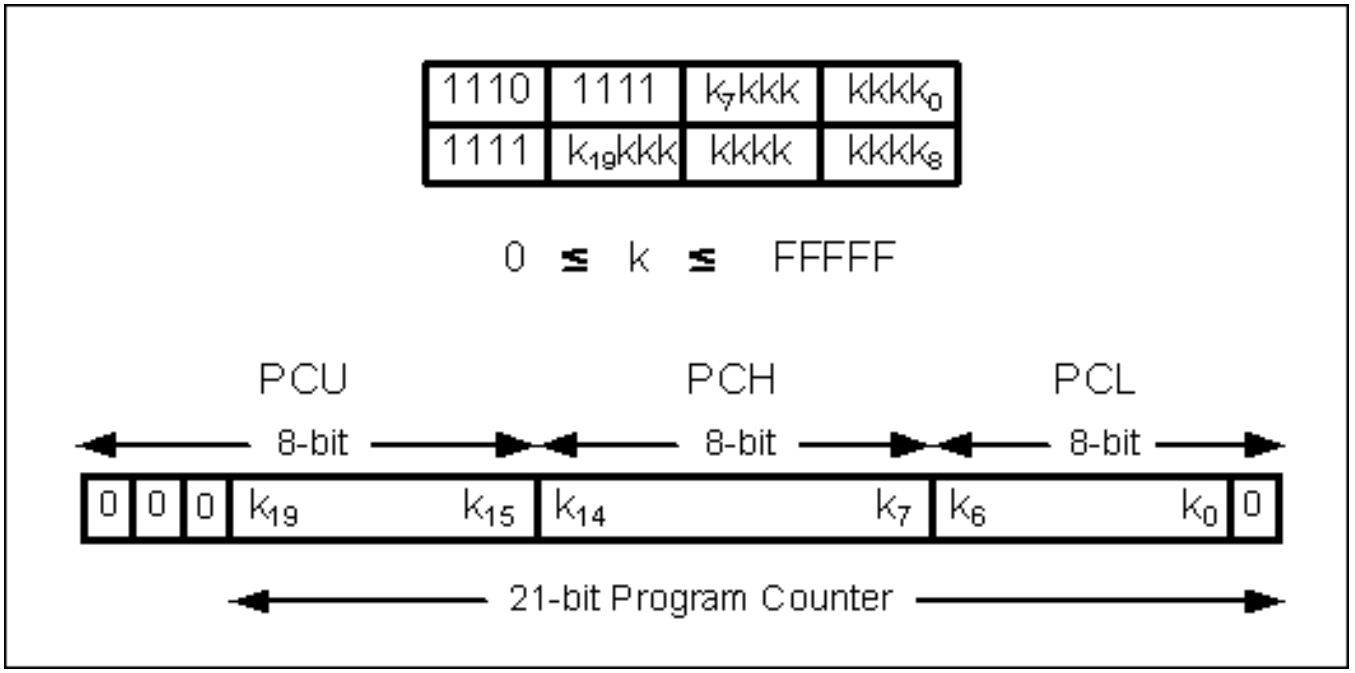
BRA n



GOTO Instruction

GOTO k

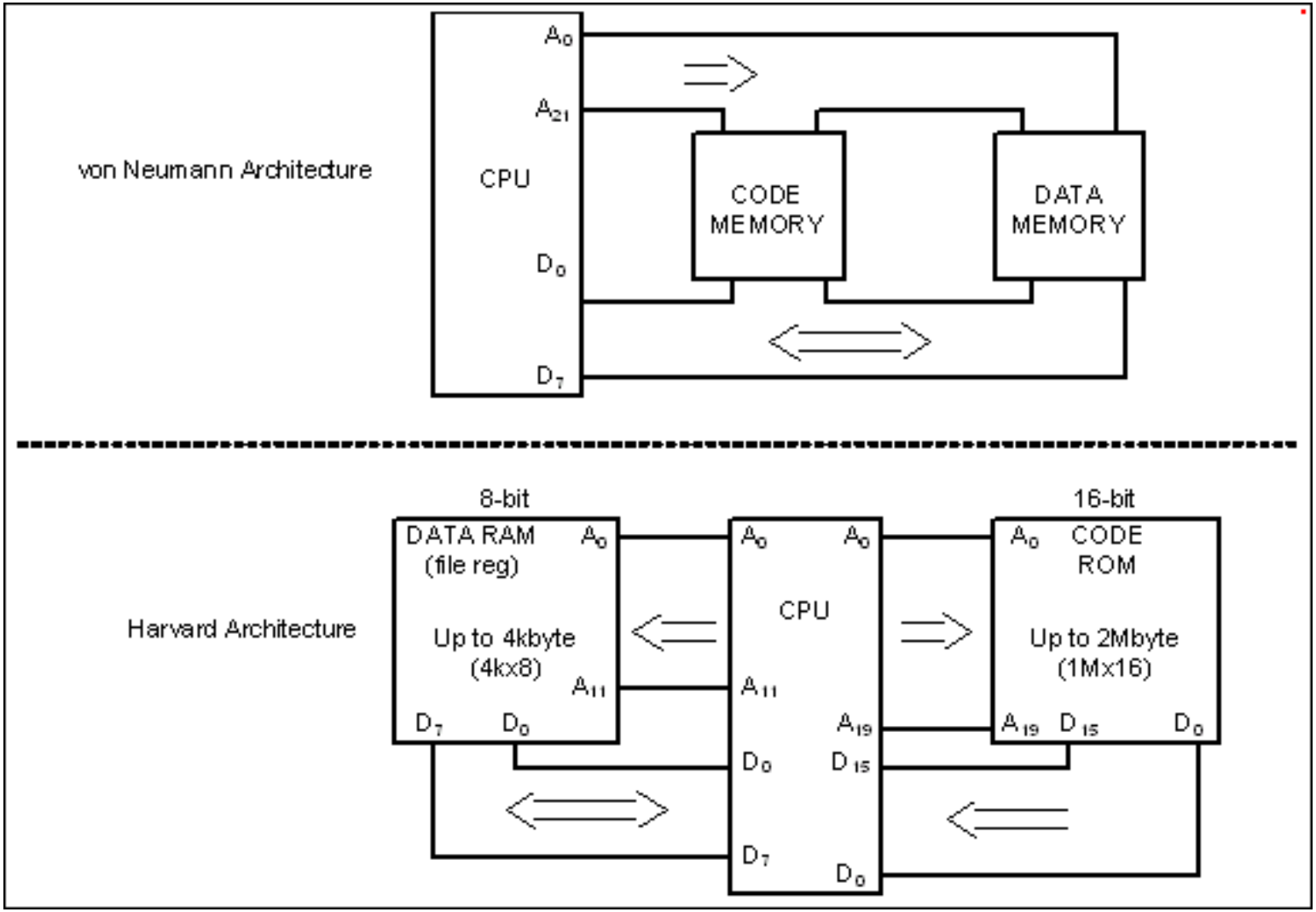
4 Byte Instruction



RISC: Reduced Instruction Set Computer

1. Fixed instruction size (2 and 4 bytes in PIC ; ADD, GOTO)
2. Many registers (no need for large stack)
3. Small instruction set – longer code
4. Small clock cycle/instruction
5. Usually Harvard architecture
6. No microcoding; instructions are internally hardwired – can result in 50% reduction in the number of transistors
7. No cross operations between GFR registers

PIC uses Harvard Architecture



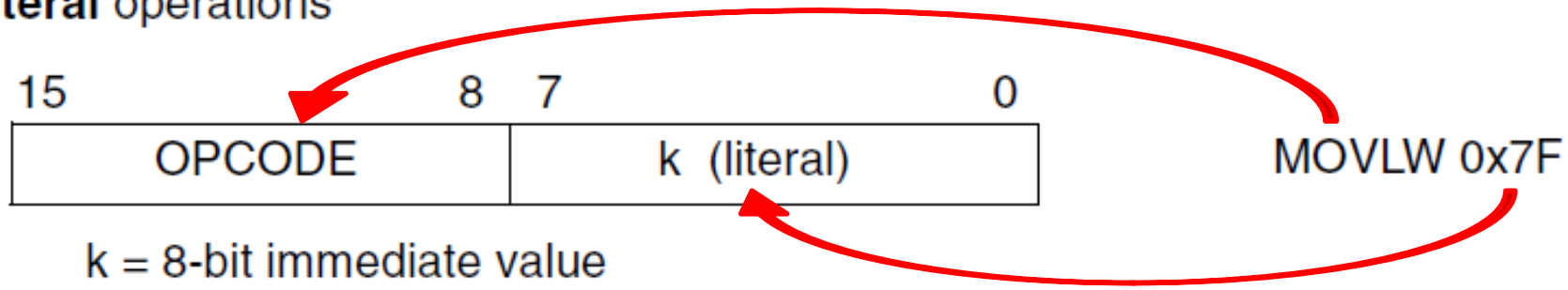
Assembler/Compiler Directives

- **Instructions** (MOVLW, ADDLW, etc.) tell CPU what to do
- **Directives** give directions to the Assembler/Compiler
  - "pseudo-instructions"

- Assembler directives:
  - **EQU** (defining constants), (**SET** is similar but can be reset)
  - **ORG** (origin - explicit address offset operand must be hex)
  - **END** (tells assembler that this is end of code)
  - **LIST** (indicates specific controller, e.g., LIST P=18F452)
  - **#include** (to include libraries associated)
  - **_config directives** – tell assembler what the configuration (stored at 300000H) bits of the target PIC should be
  - **radix** (e.g., radix dec will change to decimal notation; default is hex)

Instruction Format

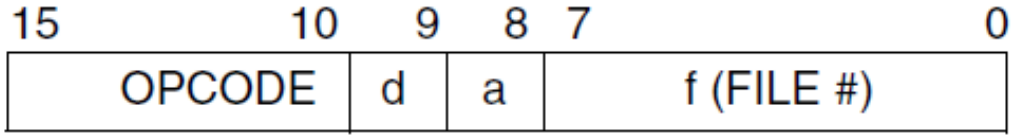
Literal operations



k = 8-bit immediate value

Byte-oriented file register operations

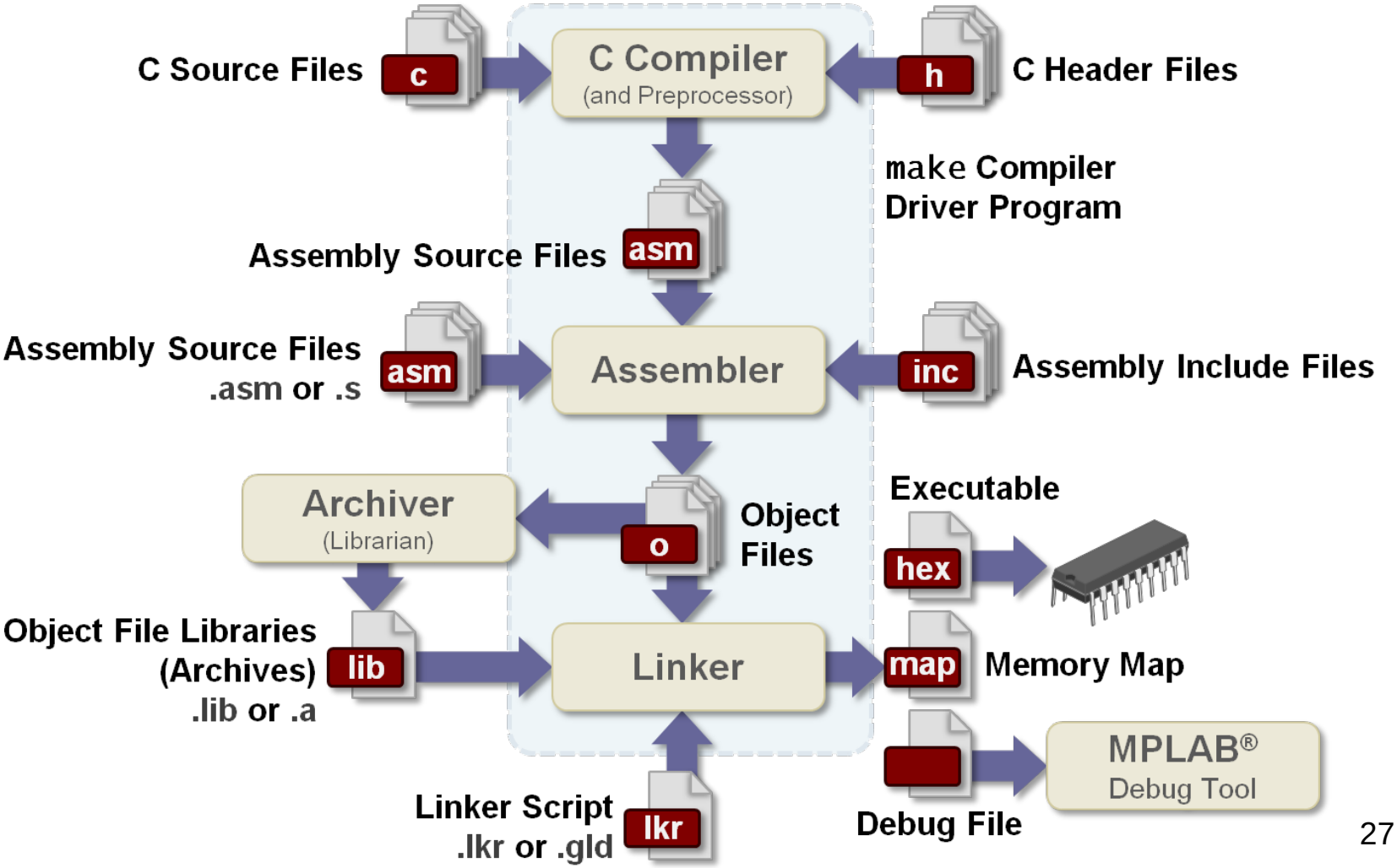
Example Instruction



ADDWF MYREG, W, B

d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Software Program Flow



Software Program Flow

CSE@UTA

```
67
68      //Main routine, forever in an infin
69      while(1)
70      {
71          //Your main code goes here
72          Print_To_LCD(Str_1);
73          Print_To_LCD(Str_2);
74
75          Toggle_LEDs();
76
77          Str_2[8]++;
78
79          if(Str_2[8] > '9')
80              Str_2[8] = '0';
81      }
```

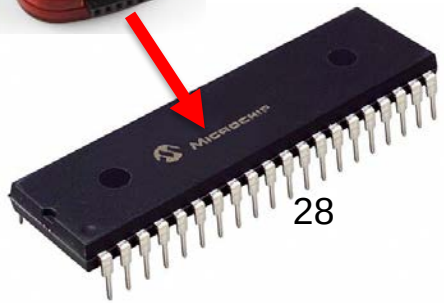
.c

```
 movlw   244
 movwf   ??_Initialize_LCD& (0+255),c
 movlw   168
u57:
 dw  65535   ; errata NOP
 decfsz  wreg,f,c
 bra u57
 decfsz  ??_Initialize_LCD& (0+255),f,c
 bra u57
 decfsz  (??_Initialize_LCD+1)& (0+255),
 bra u57
 nop2
```

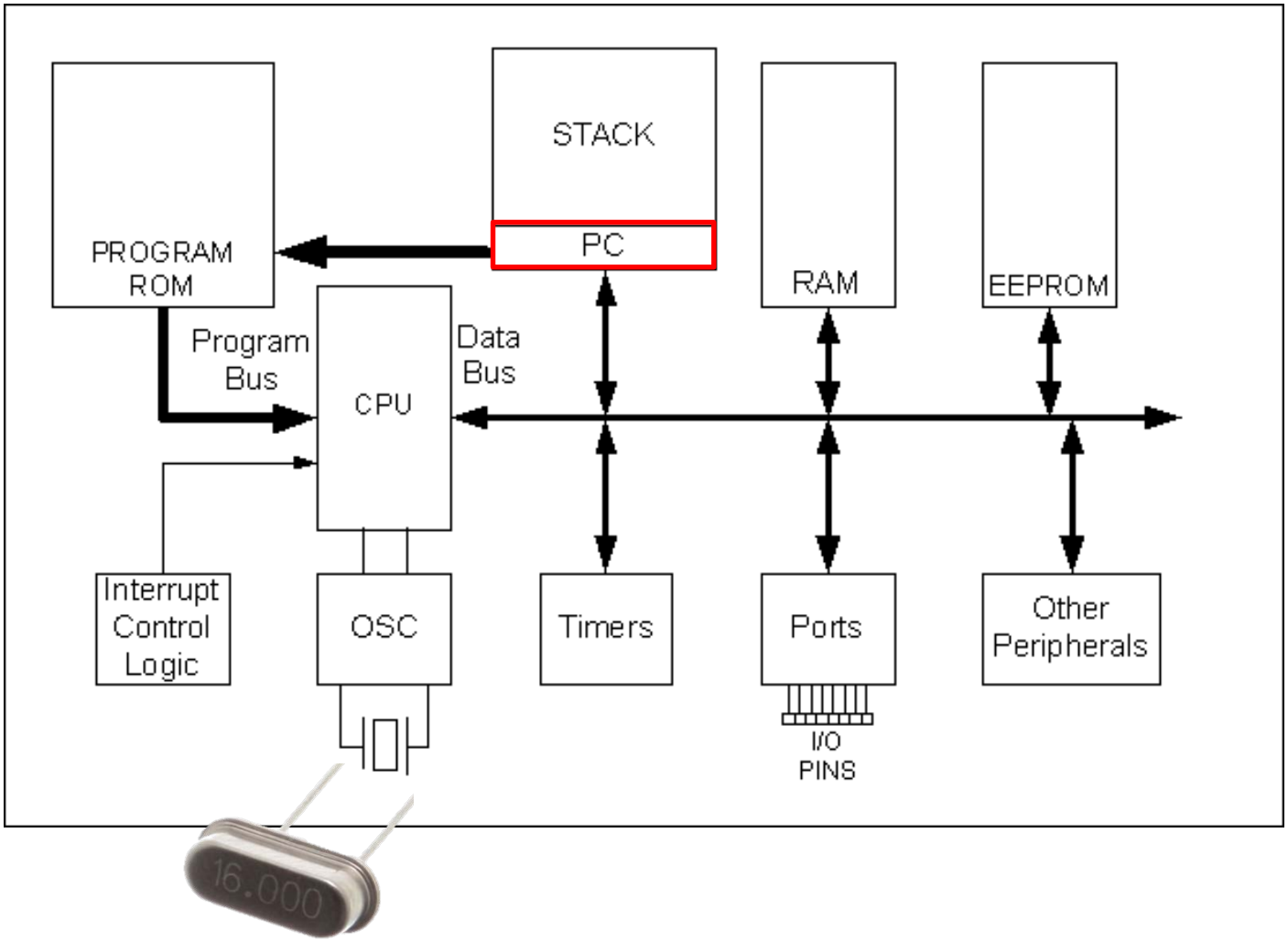
.asm

```
:100760000FEC03F01200FFFFFFFF9EEC03F00B0EF7
:10077000156E060E166E57EC03F0010E156E060E82
:10078000166E57EC03F0FFFFFED7C0202020207C
:1007900020202000802020202020202000FFFF7B
:020000040020DA
:08000000FFFFFFFFFFFFFF00
:020000040030CA
:0E000000FF220D0EFF0181FF0FC00FE00F4029
:00000001FF
```

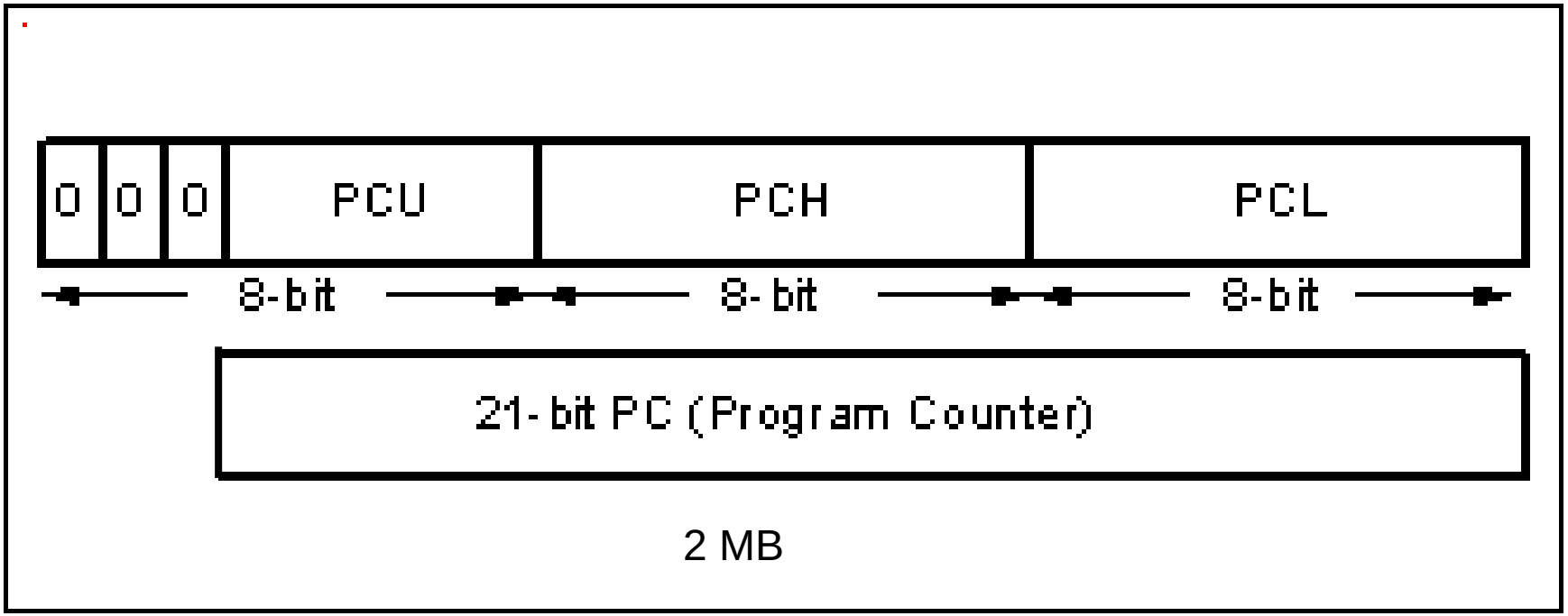
.hex



PC
Program Counter



PIC18 Program Counter



21-bit → 000000 to 1FFFFF addresses

Figure 2-9.

DECFSZ Instruction

DECFSZ fileReg, d ; Decrement fileReg and Skip next instruction if new value is 0
; if d==0 or d==w put new decremented value in WREG
; if d==1 or d==f put …. in fileReg

DECFSZ	**Decrement f, skip if 0**
Syntax:	[*label*] DECFSZ f [,d [,a]]
Operands:	0 ≤ f ≤ 255 d ∈ [0,1] a ∈ [0,1]
Operation:	(f) – 1 → dest, skip if result = 0
Status Affected:	None
Encoding:	0010 11da ffff ffff

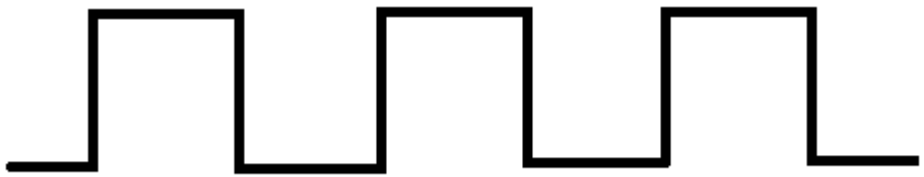
The contents of register 'f' are decremented. If 'd' is 0, the result is placed in W. If 'd' is 1, the result is placed back in register 'f' (default). If the result is 0, the next instruction, which is already fetched, is discarded, and a NOP is executed instead, making it a two-cycle instruction. If 'a' is 0, the Access Bank will be selected, overriding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

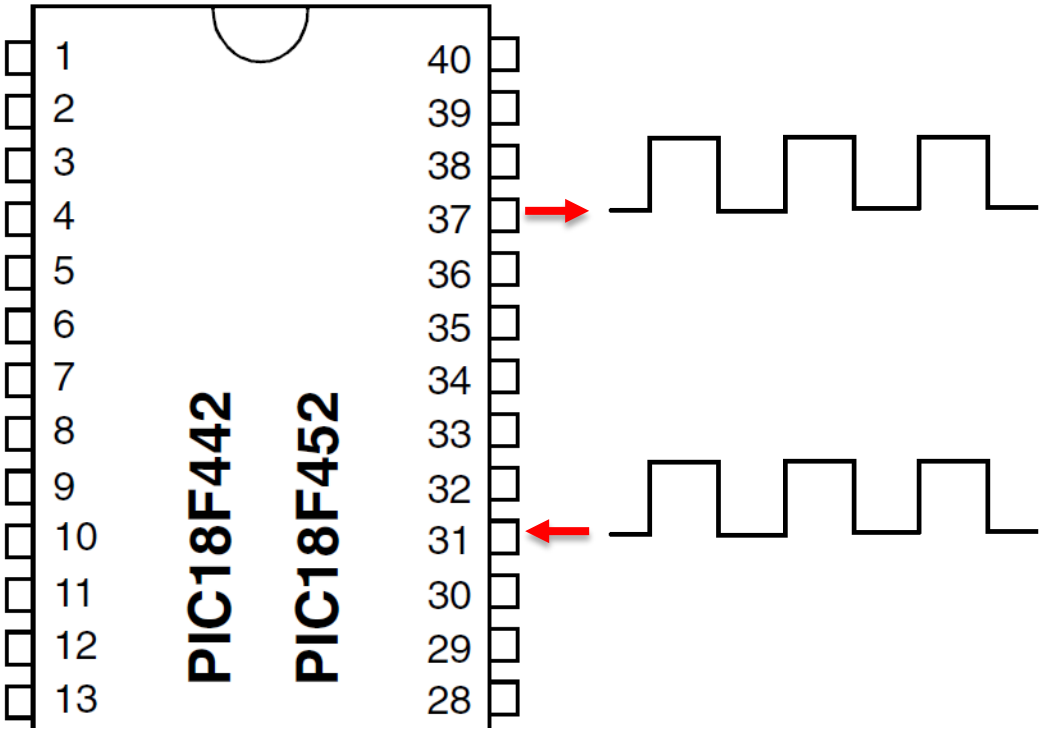
Stack

- Subroutines require stacks
- **CALL** and **RCALL** instructions can create subroutines (RETURN)
  - They are jumps but put the current PC onto the stack
- **Program Counter** needs to be stored so microcontroller knows where to return
- Stack thus has 21-bit words
  - Needs to be longer than one unit as there may be nested subroutines
  - Stack is separate RAM close to the CPU
- Separate 5-bit register (SP) for keeping track of stack (relative address)
  - SP is incremented from 0!
- User has to “stack” (store) other registers.

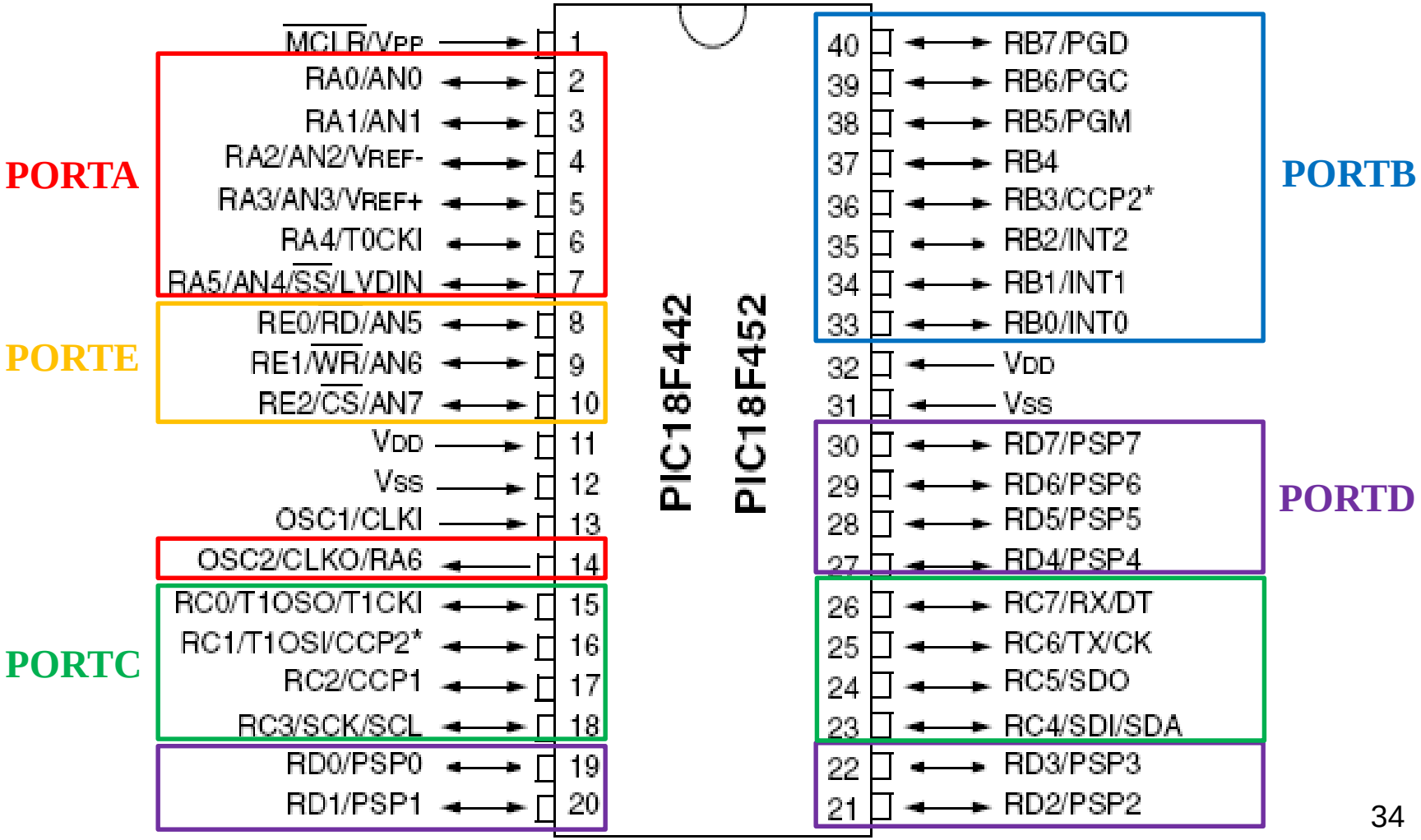
Digital Input/Output

- **Only two states**
  - ON / OFF
  - HIGH / LOW
  - 1 / 0





PIC18F452 Pin Diagram
5 Ports



Number of Individual Port Pins

- For example, the PIC18F452
  - Port A has 7 pins
  - Ports B, C, and D each have 8 pins
  - Port E has only 3 pins

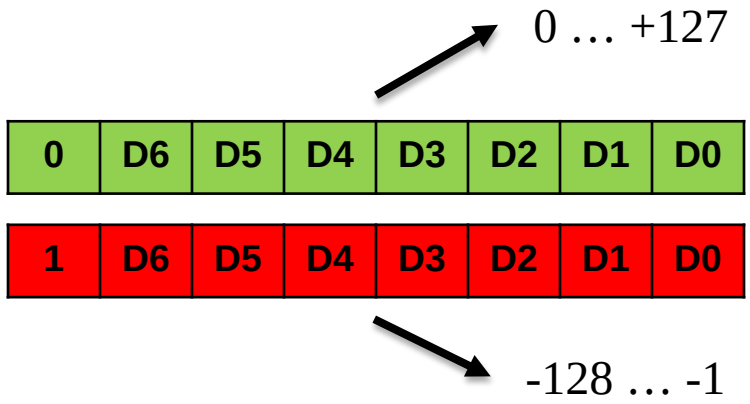
    →34 total digital IO pins
- Each port has three SFRs associated
  - **PORTx**
  - **TRISx (TRIState)**
  - ~~LATx (LATch)~~

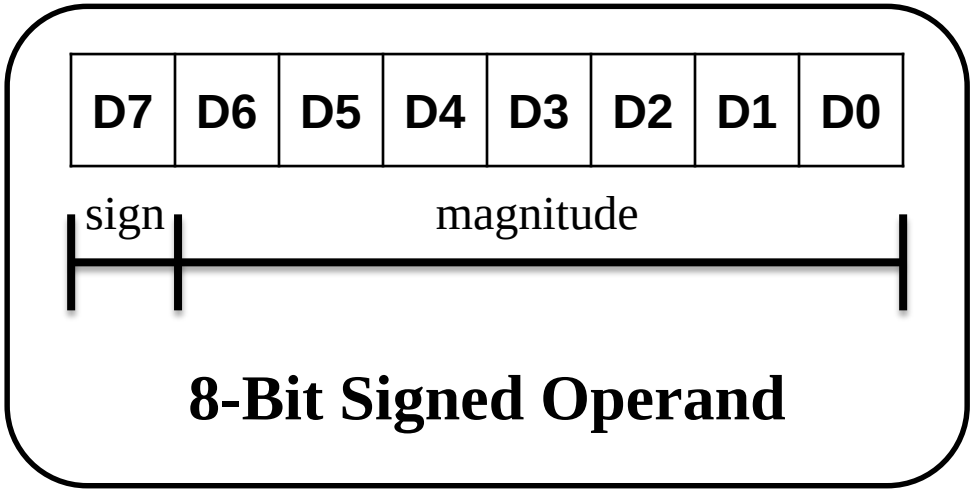
Unsigned Numbers

- 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



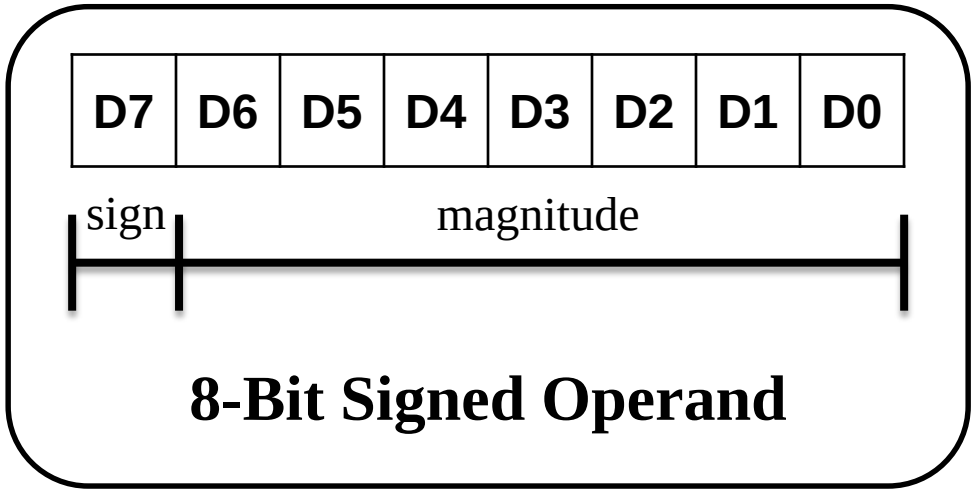


Signed Number Range

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

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



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**

Subtraction

- Like ADD, there is SUB and SUB w/borrow
- SUB only coverts 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

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
```

PRODH & PRODL Stored in the File Register

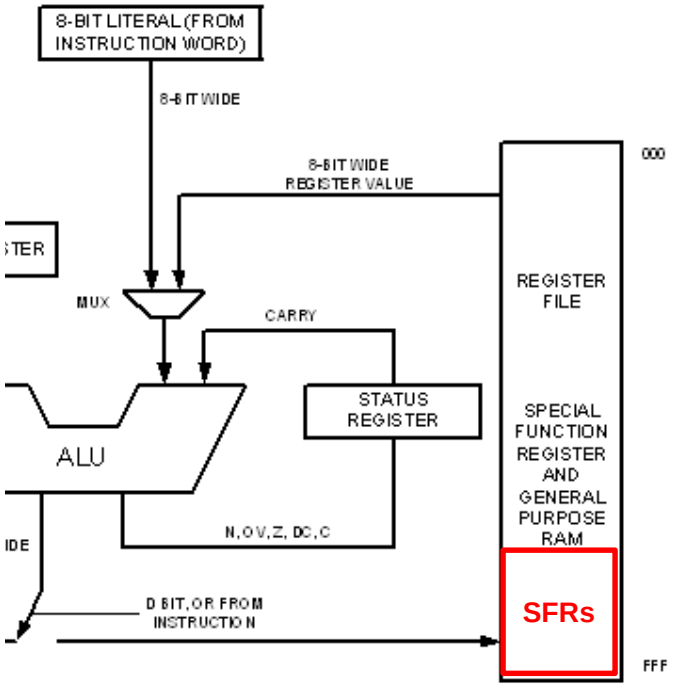


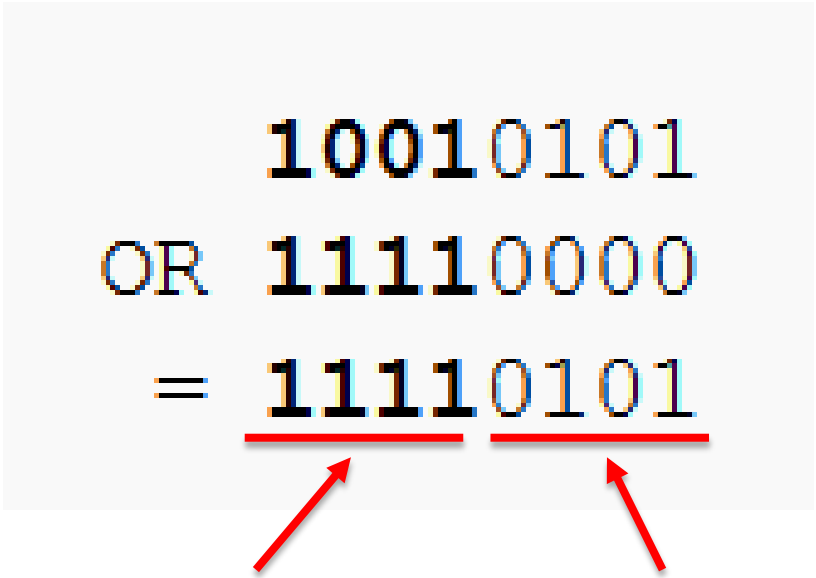
TABLE 4-1: SPECIAL FUNCTION REGISTER MAP

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

Masking

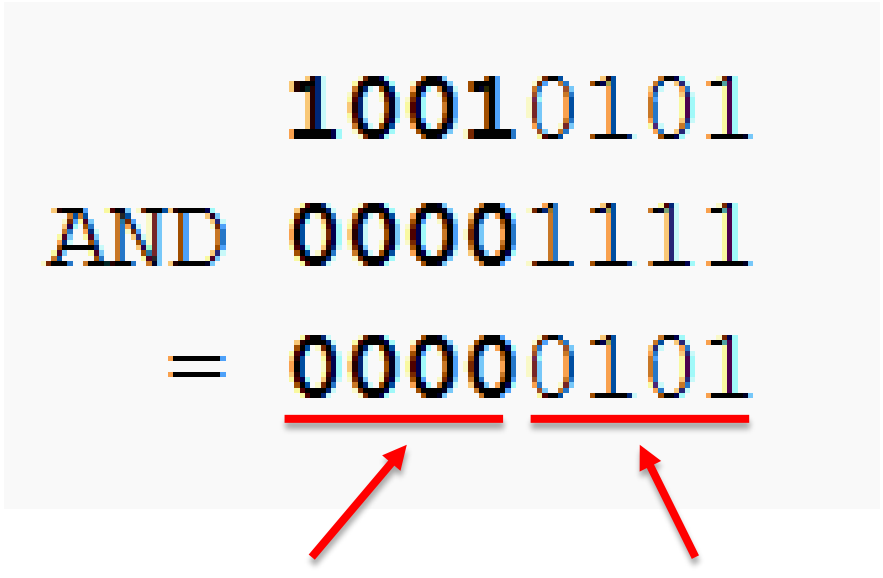
Masked **ON**

"OR" with 1

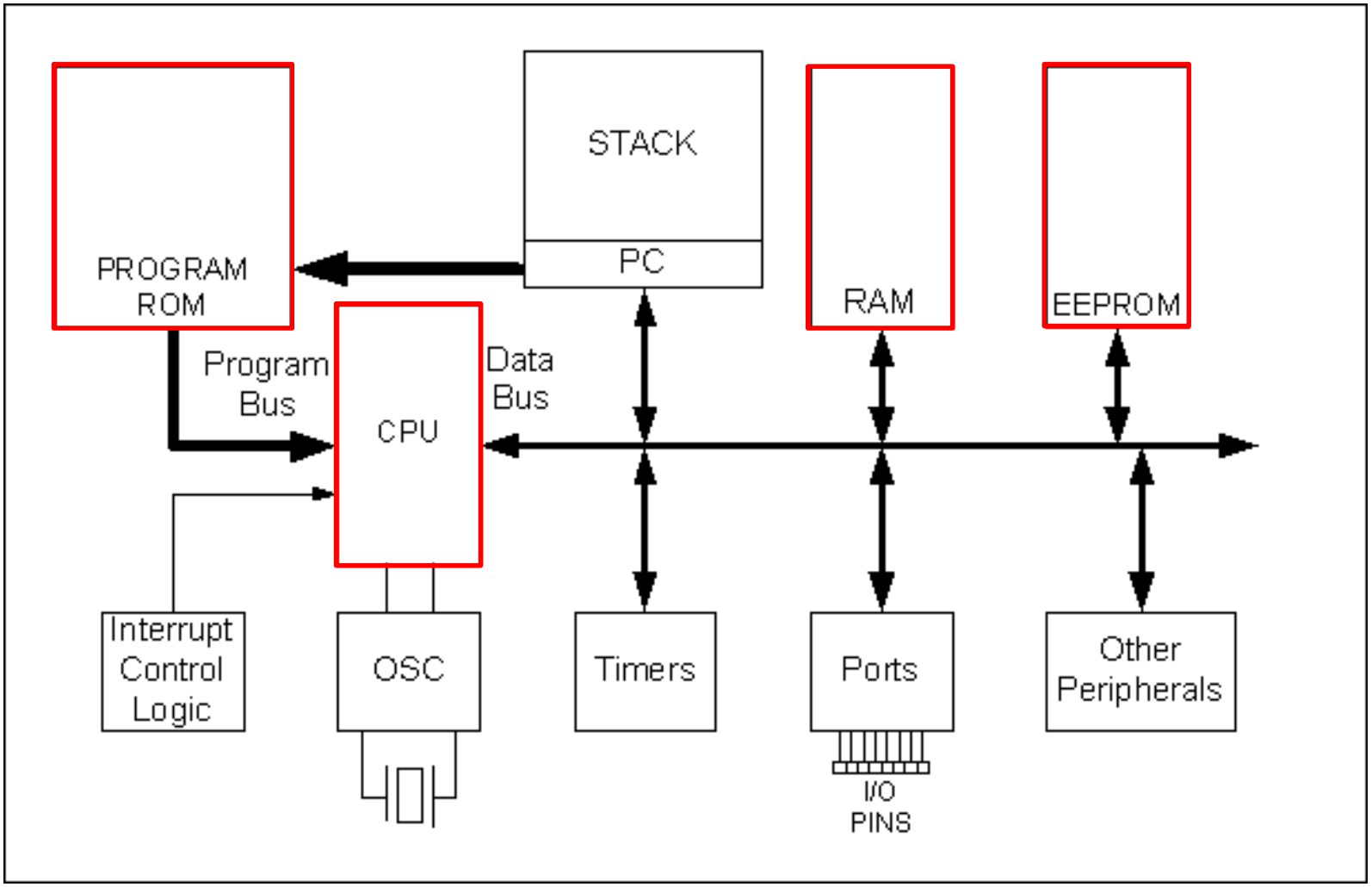


Masked **OFF**

"AND" with 0

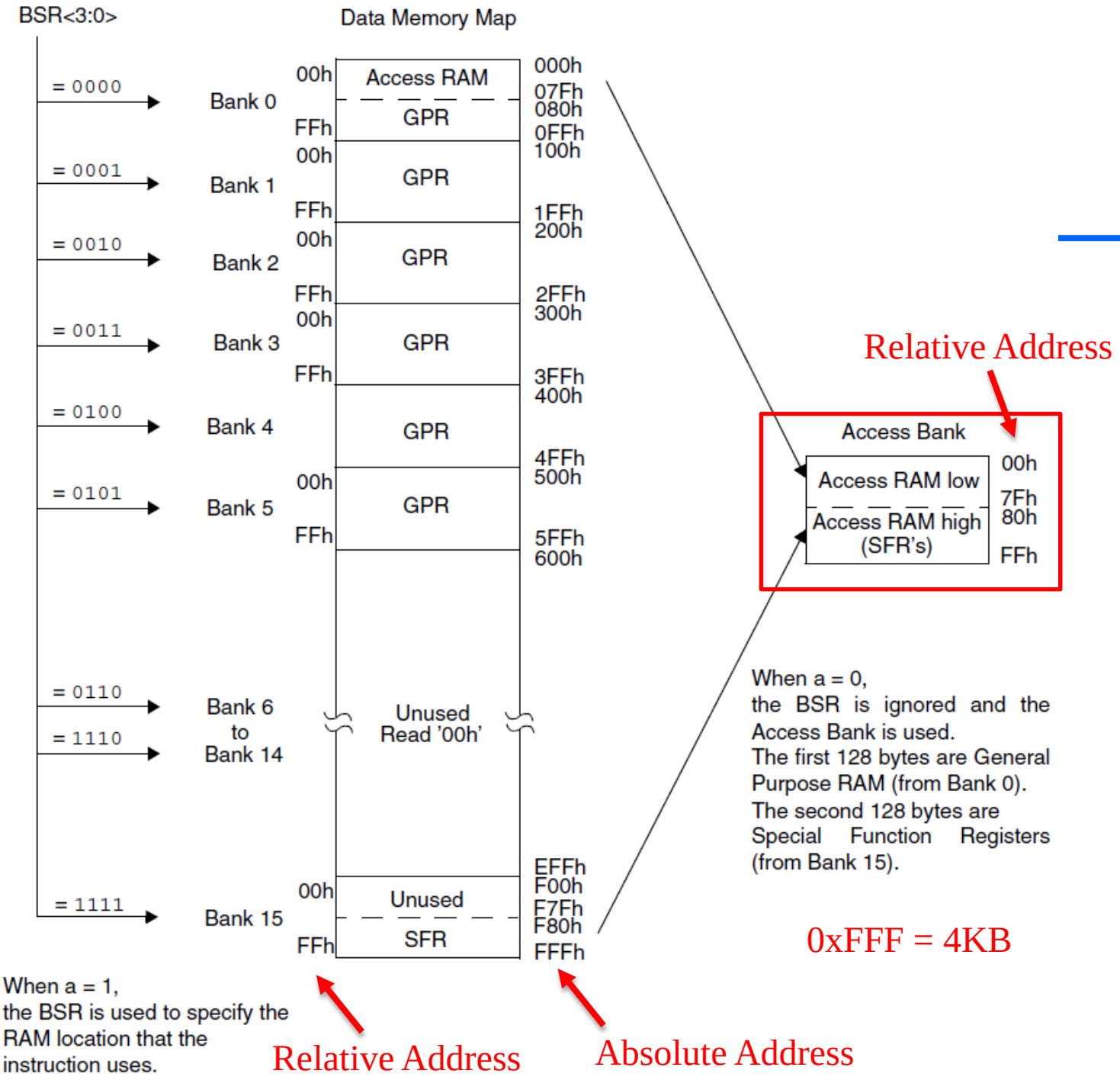


How Can the CPU Access Data?



Addressing Modes

1. **Immediate**
   - Operand part of the instruction (constant K)
2. **Direct**
   - Instruction has the operand of a RAM address and thus can be directly addressed
3. **Register Indirect**
   - Kind of like using pointers to address registers. There are specific SFRs set aside for this.
4. **Indexed-ROM**
   - Constant fixed data stored alongside the program code


BSR<3:0>
Data Memory Map
= 0000
Bank 0
00h
Access RAM
GPR
FFh
000h
07Fh
080h
0FFh
= 0001
Bank 1
00h
GPR
FFh
100h
1FFh
= 0010
Bank 2
00h
GPR
FFh
200h
2FFh
= 0011
Bank 3
00h
GPR
FFh
300h
3FFh
= 0100
Bank 4
GPR
400h
4FFh
= 0101
Bank 5
00h
GPR
FFh
500h
5FFh
600h
= 0110
= 1110
Bank 6
to
Bank 14
Unused
Read '00h'
EFFh
F00h
= 1111
Bank 15
00h
Unused
SFR
FFh
F7Fh
F80h
FFFh
Relative Address
Access Bank
Access RAM low
Access RAM high
(SFR's)
00h
7Fh
80h
FFh
When a = 0,
the BSR is ignored and the Access Bank is used.
The first 128 bytes are General Purpose RAM (from Bank 0).
The second 128 bytes are Special Function Registers (from Bank 15).
0xFFF = 4KB
When a = 1,
the BSR is used to specify the RAM location that the instruction uses.
Relative Address
Absolute Address

Macros

- Macro is used for referencing the same group of instructions repeatedly
  - Macro == sequence of instructions
- Thus do not have to repeat/write the instructions each time instruction group are used
  - For useful non-standard operations
- Place/define “above” your main code (ORG 0)
- Macros can call other macros or itself recursively
  - Max 16 nested macro calls

Macro vs. Subroutine

- **Macros**
  - Increase overall code size
    - 10-instruction macro called 10 times = 100 total instructions
  - Allows in-line arguments in macro call
  - No return values
- **Subroutines**
  - Fixed code size
  - No in-line arguments when calling a subroutine
  - Return value is “possible”
    - *retlw (return with literal in WREG)*
  - Uses stack space
    - Too many nested calls can cause stack issues

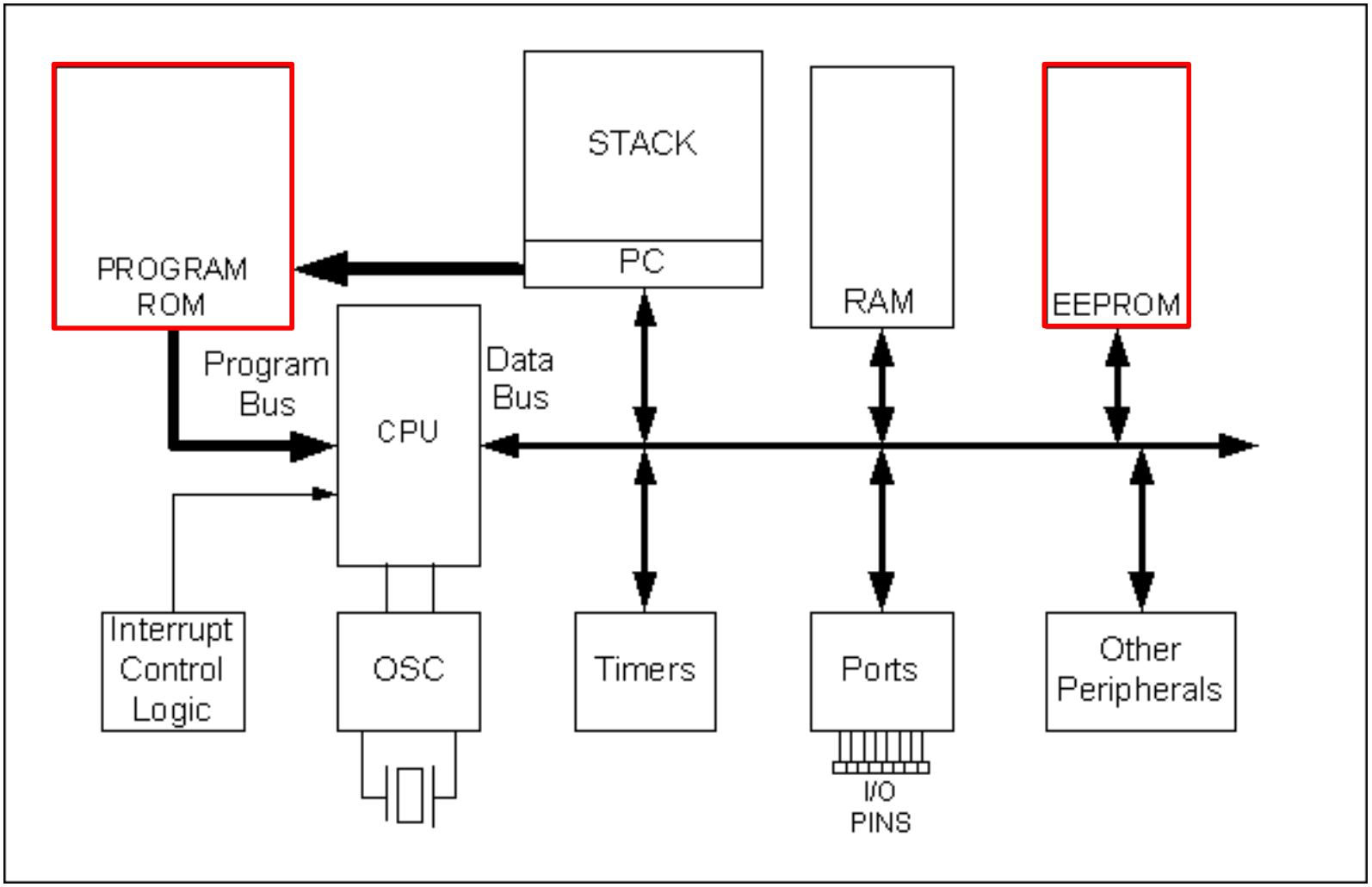
Modules

- With having the main procedure and subroutines in the same file…
  - If one subroutine fails, all must be rewritten
- Treat each subroutine as its own program
  - Known as “modules”
  - Each a separate file (.o or .asm file)
    - Assembled and tested independently
  - All brought together (linked) to form a single program

Modules Directives

- **EXTERN**
  - Notifies assembler/linker that certain names and variables are not defined in the present module but in another (externally)
- **GLOBAL**
  - Notifies assembler/linker that certain names and variables may be used by other outside (external) modules
- GLOBAL (public) allows the assembler and linker to match it with its EXTERN counterpart(s)

EEPROM and Program ROM Non-Volatile Memory in PICs



Instructions to Know

- MOVLW
- MOVFF
- MOVWF
- GOTO
- ADD instructions
- SUB instructions
- Multiplication instructions
- DECFSZ/DCFSNZ
- CALL
- RCALL
- AND instructions
- IOR instructions
- Rotate instructions
- Branch instructions
- CLRF/SETF

Directives to Know

- ORG
- LOCAL
- DB
- DATA
- EQU
- GLOBAL
- EXTERN
- #pragma
- #include

Questions?