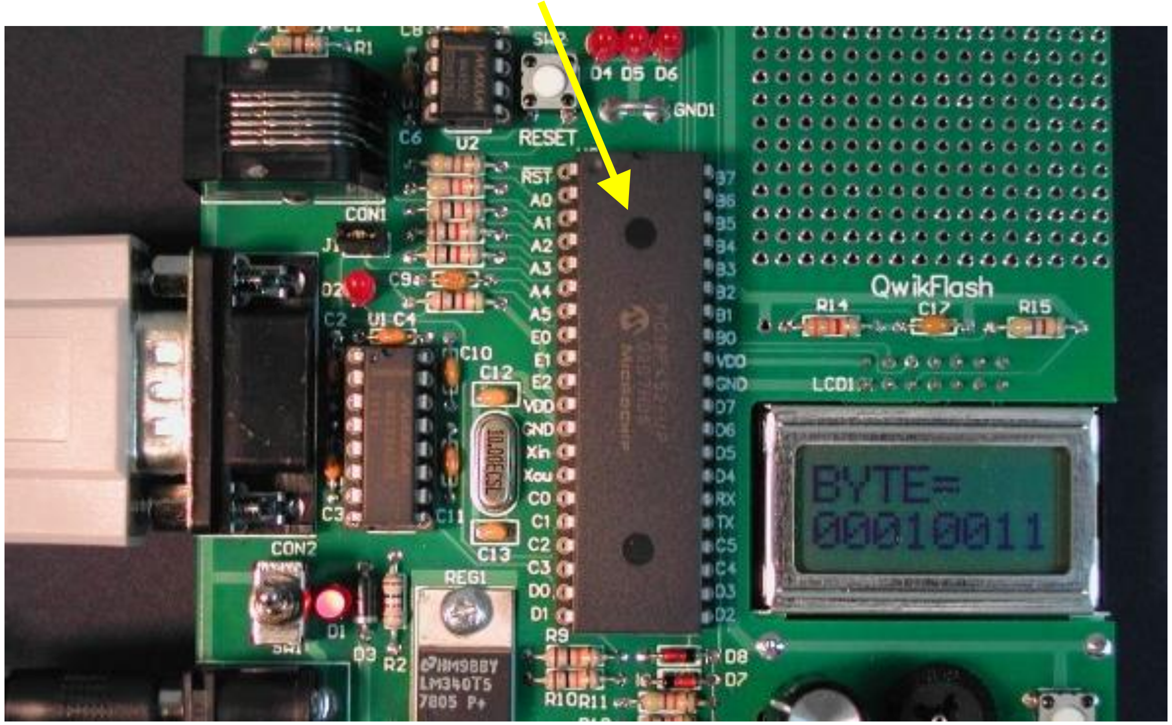
EE425 Computer Engineering Lab

Introduction

Tools

PIC18F4520



QwikFlash board

Microcontroller - Basics

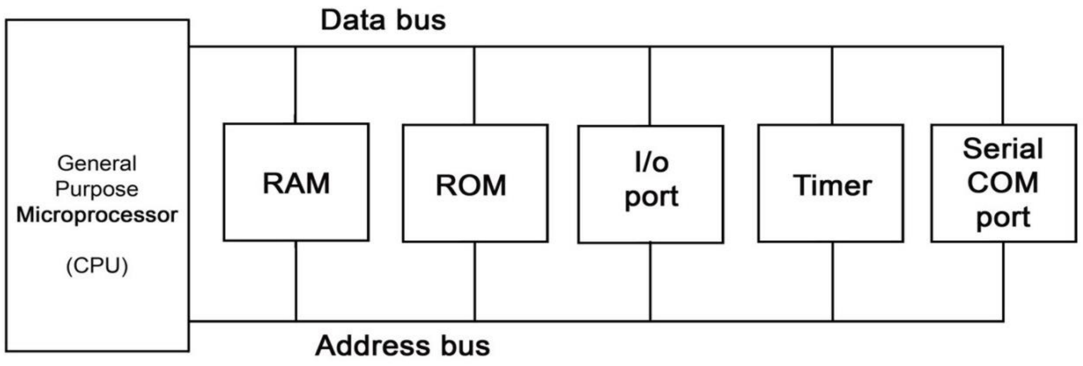
What is a Microcontroller?

Self-contained computer on a chip, consisting of:

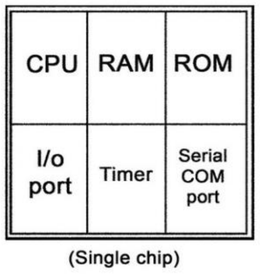
- Central Processing Unit (CPU)
- Non Volatile program memory
- Random Access Memory (RAM)
- Input Output (I/O) capabilities

MicroController - Basics

(a). GENERAL PURPOSE MICROPROCESSOR SYSTEM



(b). MICROCONTROLLER



Read Only Memory (ROM)

- Programmable ROM (PROM)
  - One Time Programmable (OTP)

- Erasable Programmable ROM (EPROM)
  - Can be erased by exposing it to strong ultraviolet light source

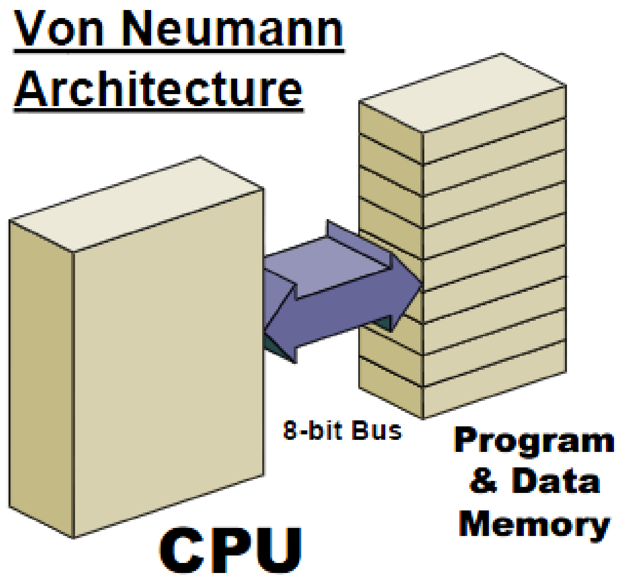
- Electrically Erasable Programmable ROM (EEPROM)

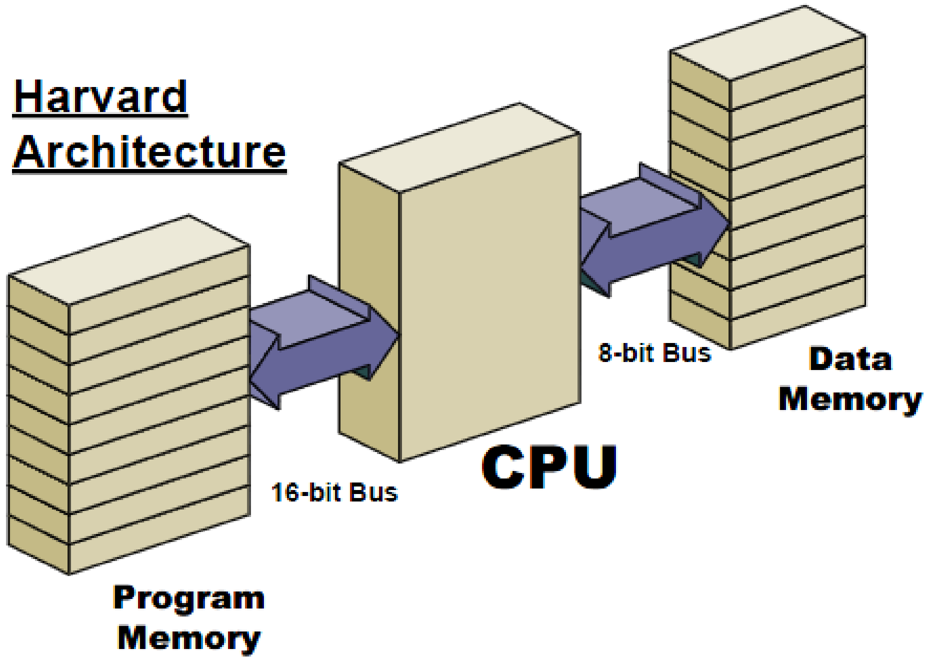
Introduction to Computer Engineering Design

CPU Architecture

CPU Architecture

- Von-Neumann architecture
  - Only one (shared) bus between CPU memory
- Harvard architecture
  - Separate data and program memory buses

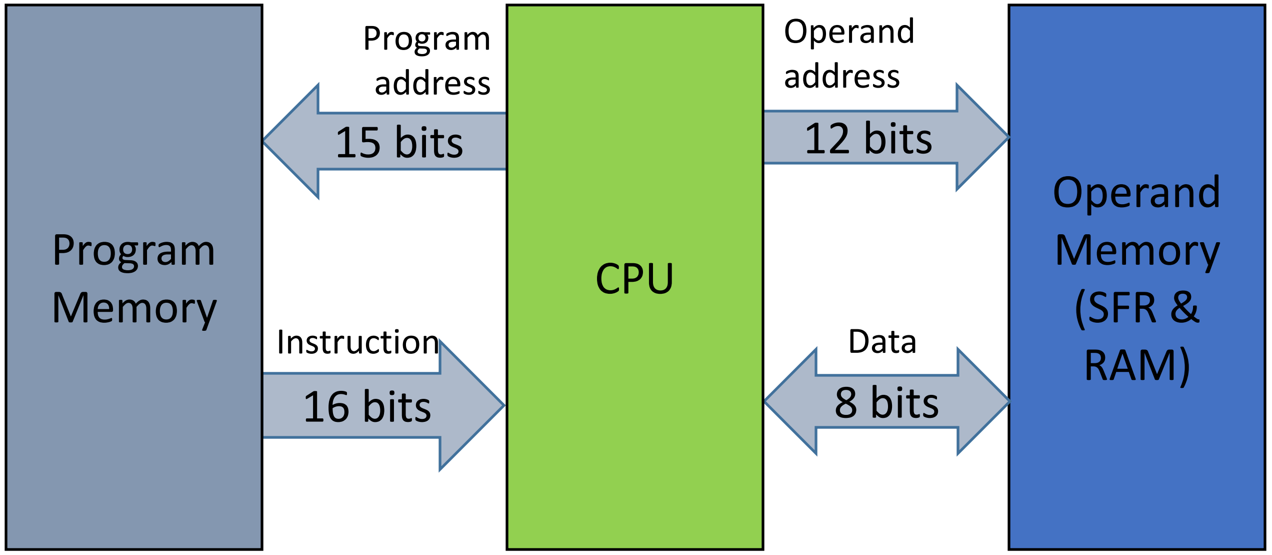




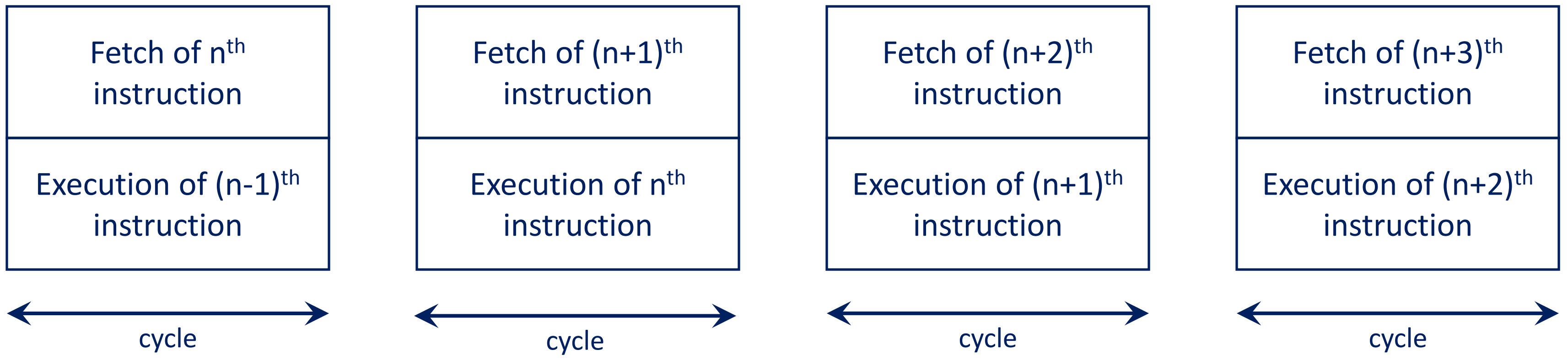
Harvard Architecture

- Separate program and data bus
- Supports pipelining
  - Fetch next instruction while executing current instruction

- Operand address bus
  - 12 bits wide
  - 2^12 = 4096
- Data bus
  - 8 bits wide
- Program address bus
  - 15 bits wide
  - 2^15 = 32768
- Instruction bus
  - 16 bits wide

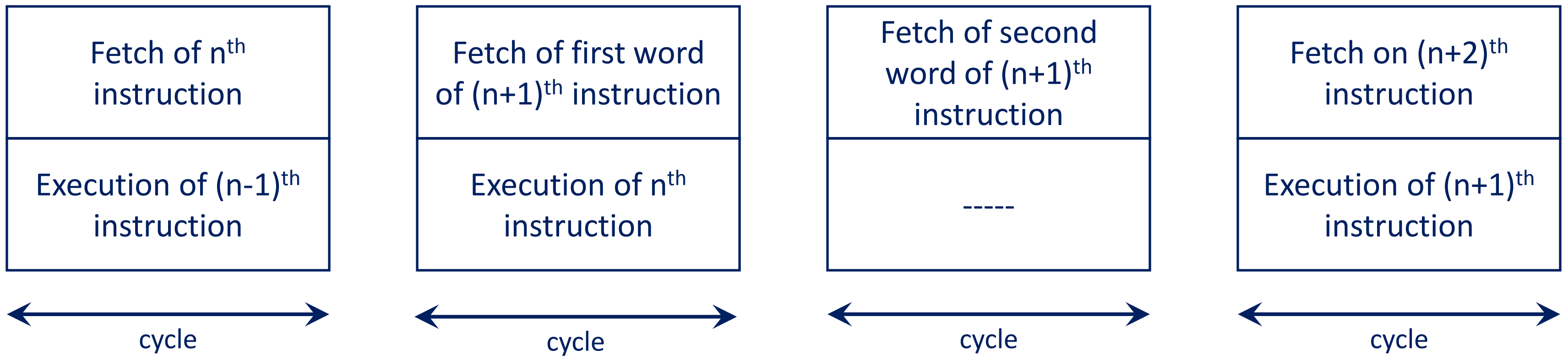


Instruction Pipelining: one-word instruction


Fetch of nth instruction
Execution of (n-1)th instruction
Fetch of (n+1)th instruction
Execution of nth instruction
Fetch of (n+2)th instruction
Execution of (n+1)th instruction
Fetch of (n+3)th instruction
Execution of (n+2)th instruction
cycle
cycle
cycle
cycle

- CPU fetches a new instruction while executing the previously fetched instruction
- An one word instruction is executed in one cycle.

Instruction Pipelining: two-word instruction

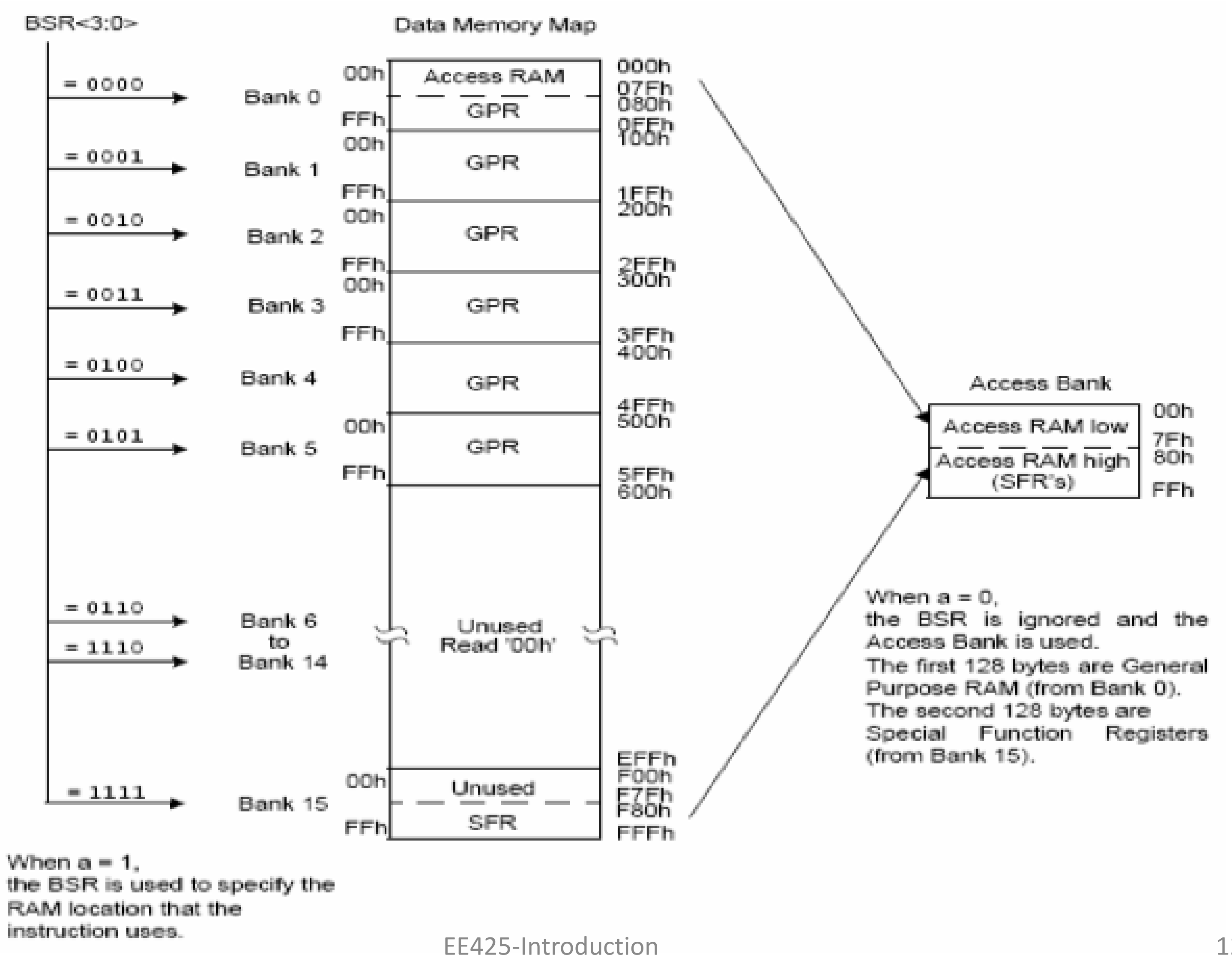

Fetch of nth instruction
Execution of (n-1)th instruction
cycle
Fetch of first word of (n+1)th instruction
Execution of nth instruction
cycle
Fetch of second word of (n+1)th instruction
-----
cycle
Fetch on (n+2)th instruction
Execution of (n+1)th instruction
cycle

- Any instruction requires two cycle for the instruction fetch then it is executed in two cycles.

Registers of interest

- **BSR**: Bank Select Register 4 bits (16 banks)
  - used in direct addressing the data memory
- **SFR**: Special Function Register
  - 16-bit registers used as memory pointers in indirect addressing data memory
  - used for accessing ports, support devices, processes of data transfer, etc.
- **GPR**: General purpose register, store data 6banks
- **FSR**: File select register
  - used as pointers for data registers
  - holds 12-bit address of data register
- **INDF**: Indirect File operand(s) (not physical)

Operand Memory Map



When a = 1,
the BSR is used to specify the RAM location that the instruction uses.

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).

Operand Addressing

The PIC18F452 employs any of the three methods for addressing an operand:

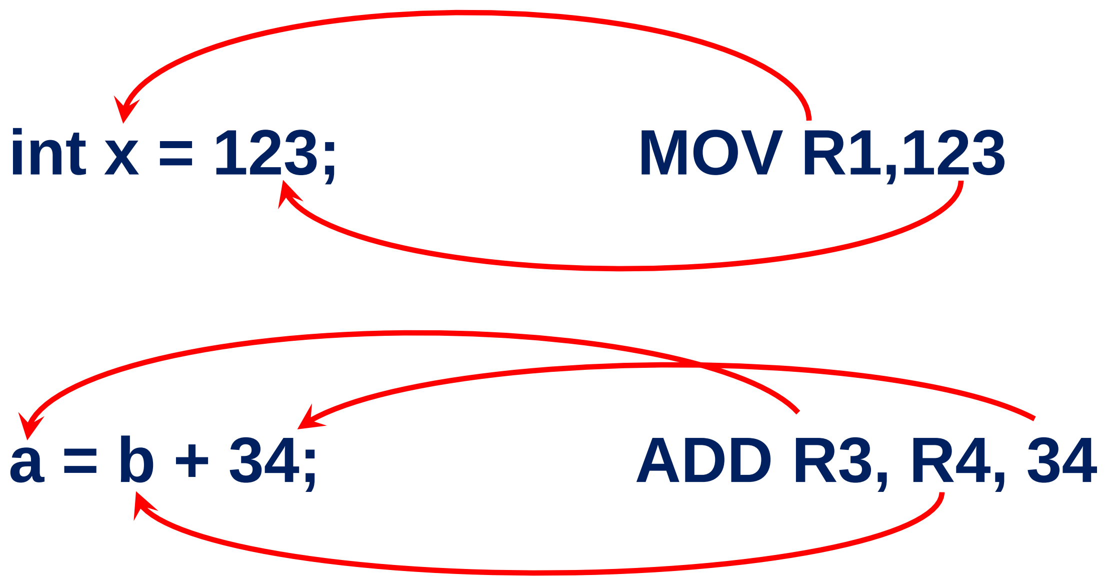
- Literal addressing
- Direct addressing
- Indirect addressing

Literal Addressing

Both operand value and operand contained in the instruction.

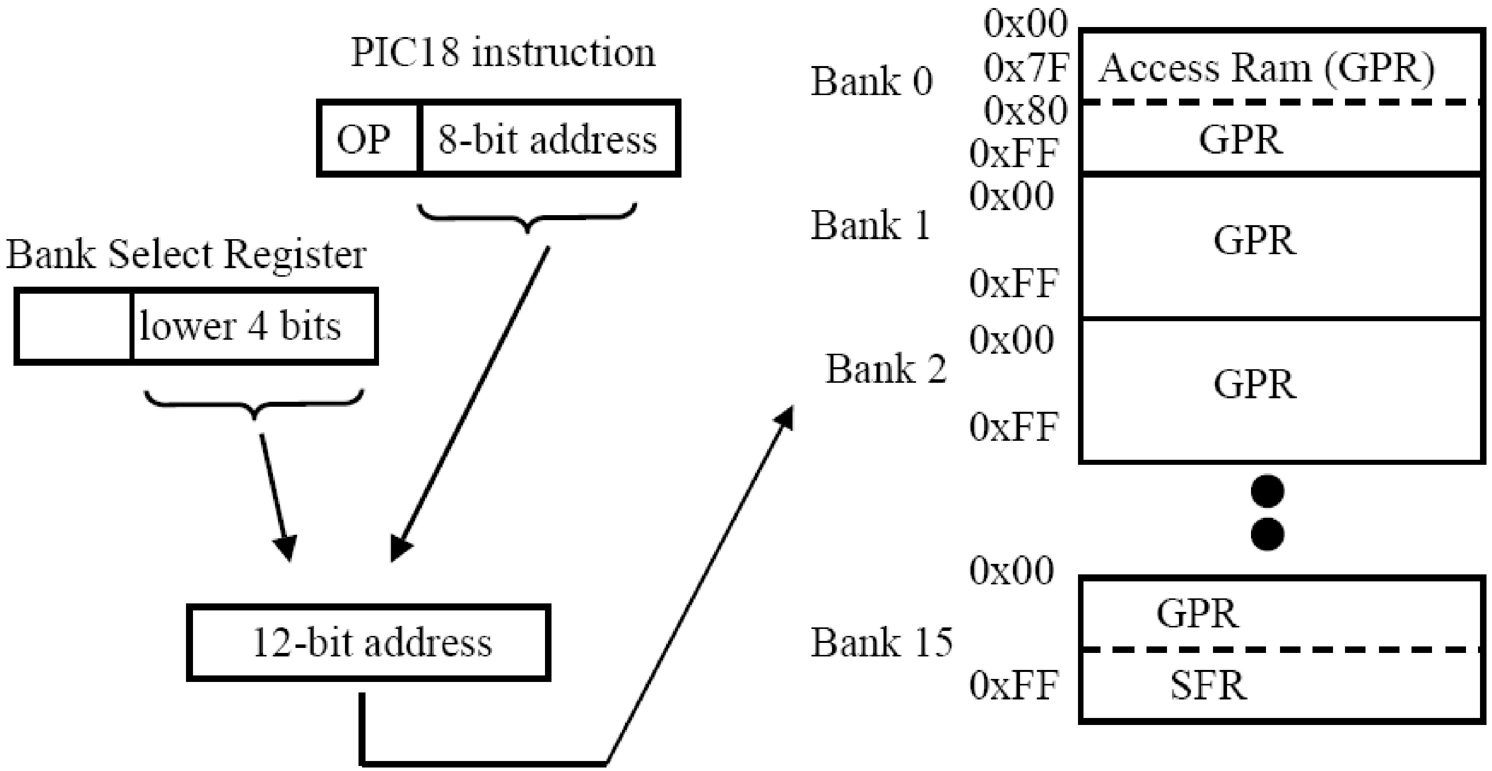
For example:

High level code	Assembly language statements
int x = 123;	MOV R1,123
a = b + 34;	ADD R3, R4, 34



Direct Addressing

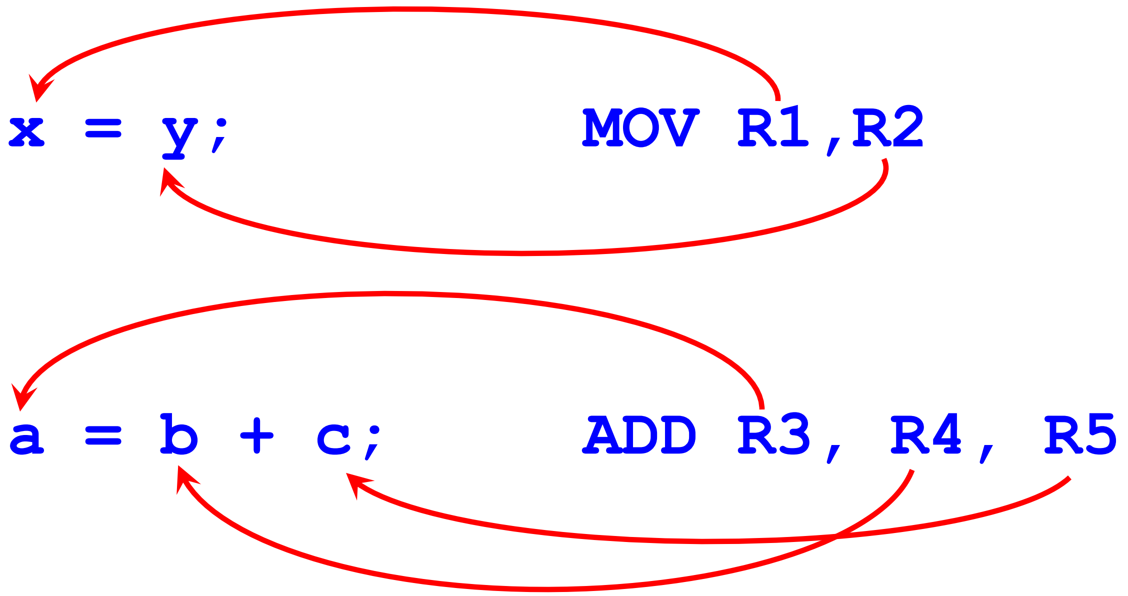
- 8 bits of the 16-bit instruction specify any one of 256 locations as the address of the operand
- The 9th bit specifies either the *Access Bank (=0)* or *one of the banks (=1)*
- The 12-bit operand address is composed by the 8 bits address contained in the instruction and the 4-bits Bank Select Register (BSR).



Direct Addressing

For example:

High level code   Assembly language statements

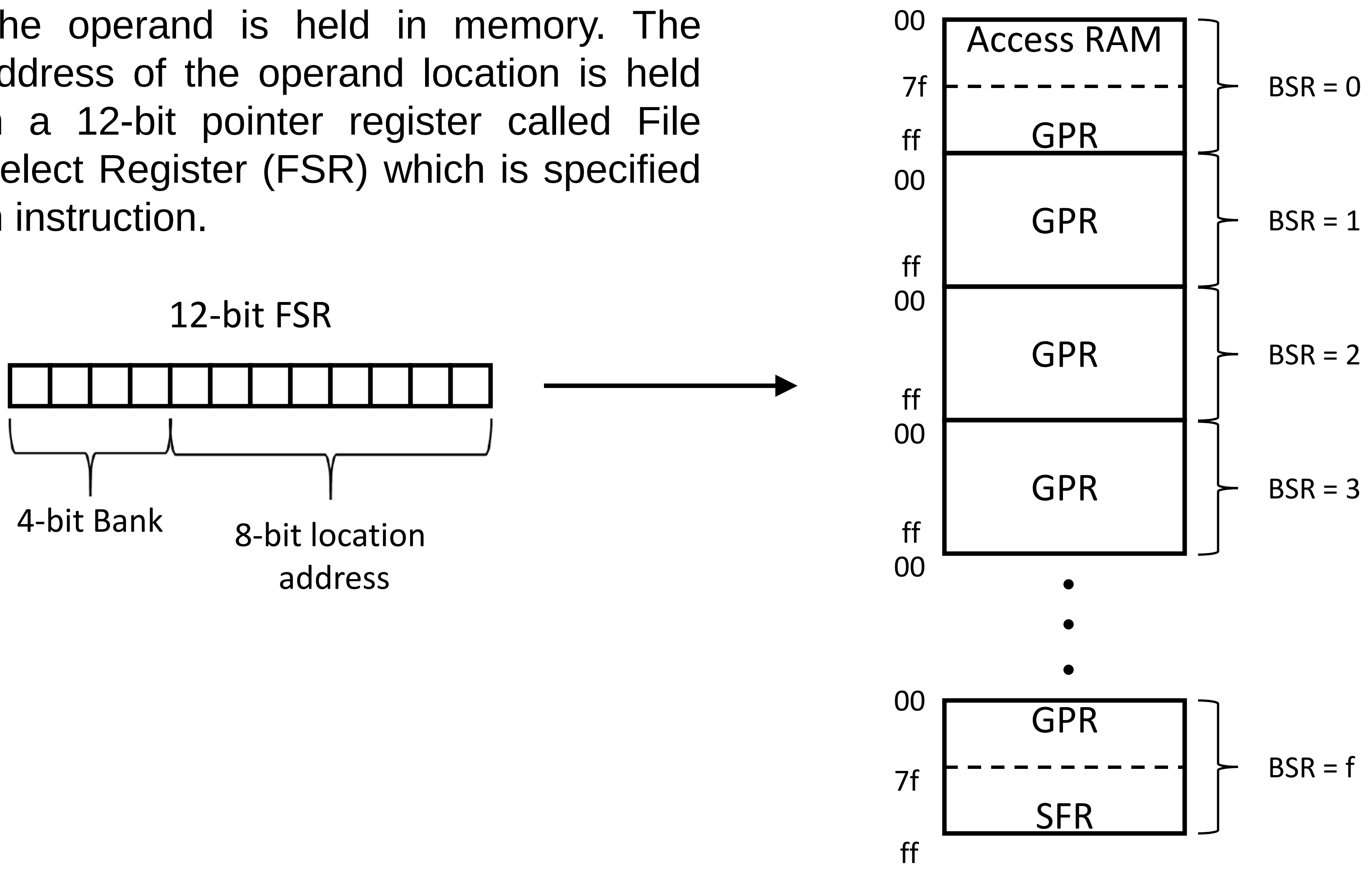


```
x = y;              MOV R1,R2

a = b + c;          ADD R3, R4, R5
```

Indirect Addressing

The operand is held in memory. The address of the operand location is held in a 12-bit pointer register called File Select Register (FSR) which is specified in instruction.



Indirect Addressing

Example of Register indirect addressing used in compiled code:

In C, for pointers.

High level code

Assembly language statements

```
int *b;    // ptr, a pointer
int a;     // a, integer say in memory loc. 120
b = &a;    // b = address of a                    MOV R1,120
*b = 123;  // location pointed by b = 123         LD [R1],123
```

Reading operands from program memory

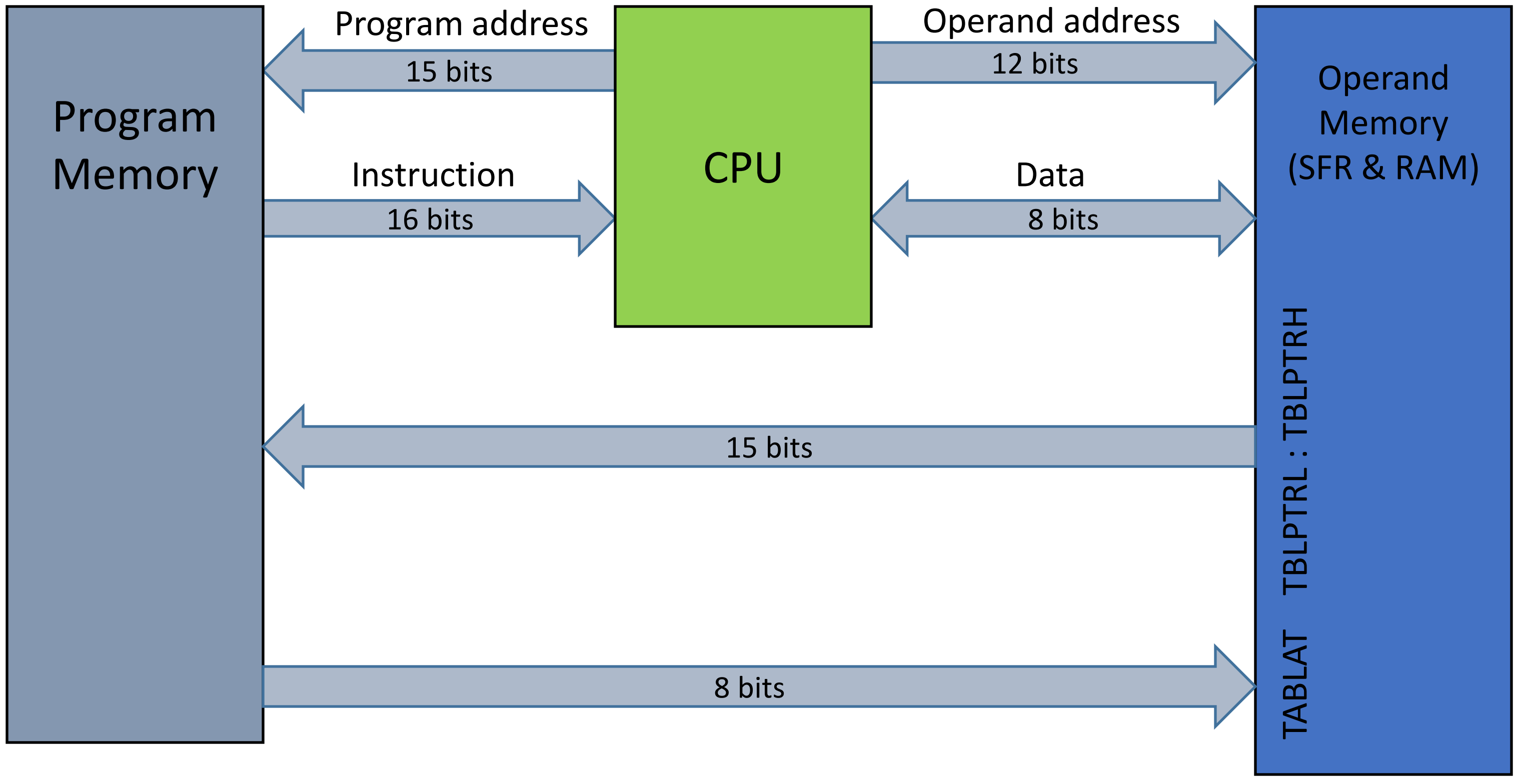
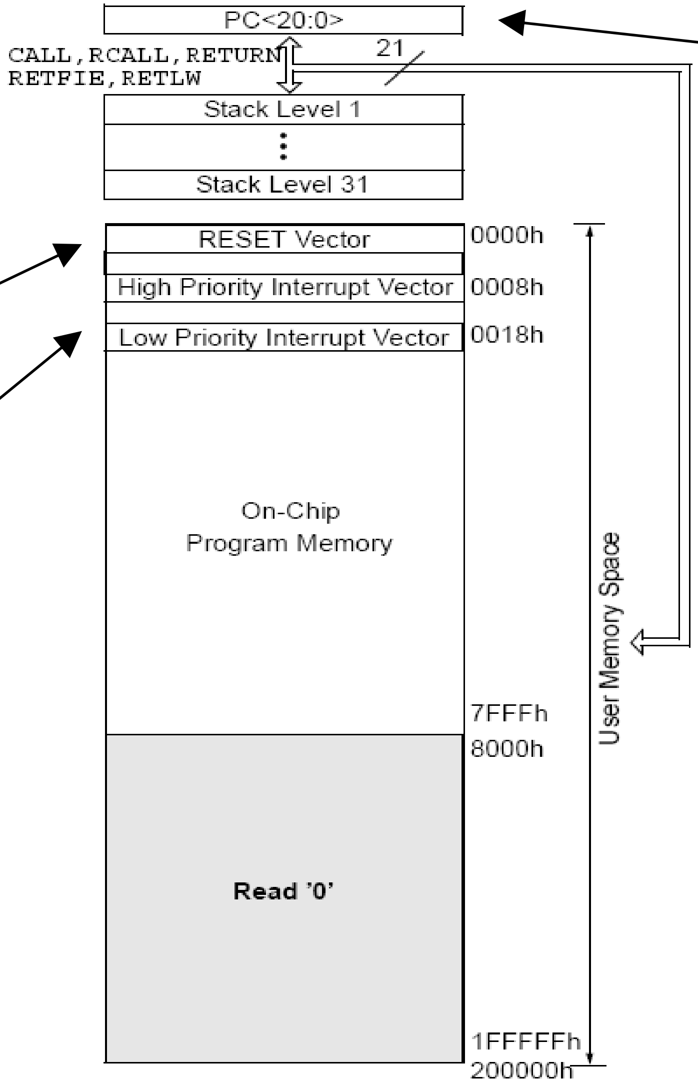


Table pointer access mode

There are 4 access mode:

- tblrd*
  - Move the 8 bit value at the location TBLPTRH:TBLPTR to TABLAT. TBLPTR remains unchanged
- tblrd*+
  - Move the 8 bit value at the location TBLPTRH:TBLPTR to TABLAT. TBLPTR is incremented
- tblrd*-
  - Move the 8 bit value at the location TBLPTRH:TBLPTR to TABLAT. TBLPTR is decremented
- tblrd+*
  - TBLPTR is first incremented then the 8 bit value at the location TBLPTRH:TBLPTR is moved to TABLAT.

Program Memory



Instructions that invoke the stack

Program Counter
It points the instruction's address of memory

The RESET vector address is at 0000h

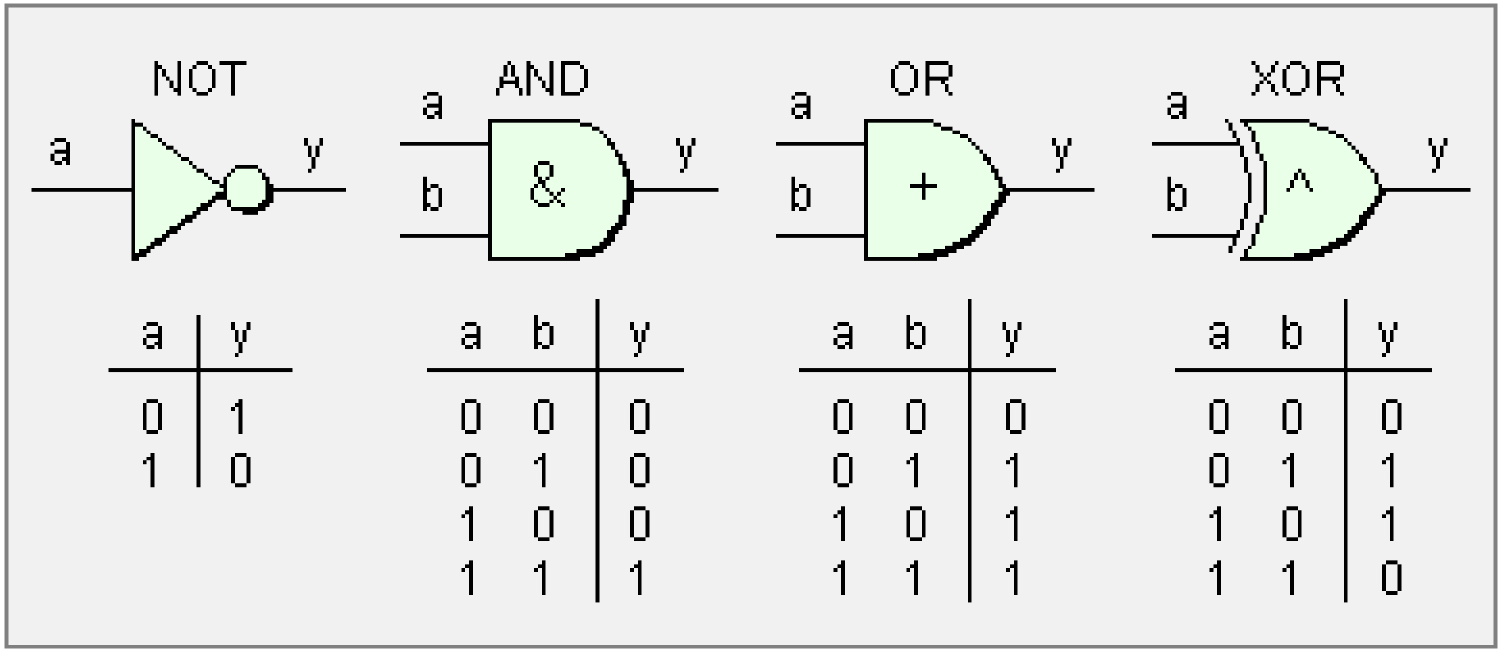
.

The interrupt vector addresses are at 0008h and 0018h.

Introduction to Computer Engineering Design

Instruction Set

Logic operator review



a	y
0	1
1	0

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

STATUS register

The STATUS register contains bits (Flags) that are set or cleared in response to execution of various instructions:

- C (Carry/Borrow Flag):
  - set when an addition of two 8-bit unsigned binary numbers generates a carry and a subtraction generates a borrow, cleared otherwise
- DC (Digit Carry Flag):
  - also called Half Carry flag; set when carry generated from bit3 to bit4 in an arithmetic operation of packed BCD (Binary-Coded Decimal) numbers, cleared otherwise
- Z (Zero Flag):
  - set when result of an operation is zero, cleared otherwise
- OV (Overflow Flag):
  - set when result of an operation of signed numbers goes beyond seven bits, cleared otherwise
- N (Negative Flag):
  - set when the most significant bit of a 2's complement signed number becomes 1 after a arithmetic/logical operation. Cleared otherwise

F/W distinction

- W, WREG (working register) is used as accumulator by moving an operand from RAM to WREG and then from WREG to a new operand in the RAM (accumulator –centered microcontroller)

```
        movf    NUM1, W           ; move NUM1 to WREG
        addwf   NUM2, W           ; WREG = NUM1 + NUM2
        movwf   NUM1              ; NUM1 = WREG
```

- F, for faster two-operand instructions. One operand is moved from RAM into the WREG and the result of a two operand instruction can be directly stored in the new operand

```
        movf    NUM2, W           ; move NUM2 to WREG
        addwf   NUM1, F           ; NUM1 = NUM1+NUM2
```

MOV instructions

- movlw (1-word)

  move a 1-byte *literal* value (constant) into WREG (working register)

  ex:

  movlw 7f $\longrightarrow$ WREG = 7f

- movwf (1-word)

  copy the contents of WREG into another 1-byte register (variable)

  ex:

  movlw B'11100000' $\longrightarrow$ WREG = 11100000

  movwf TRISB $\longrightarrow$ TRISB = 11100000

- movff (2-word)

  copy a 1-byte source operand to a destination operand

  ex:

  movff PORTB, PORTB_COPY

MOV instructions

- movf (1-word)
  - Move an operand into WREG or into the same register.
  - Usually used to move one operand of a two-operand instruction into WREG
  - It is the only one that affects the STATUS register (flags Z and N)

ex1:

movf NUM2,W $\longrightarrow$ WREG = NUM2

addwf NUM1,F $\longrightarrow$ NUM1 = WREG + NUM1 = NUM2 + NUM1

ex2:

movf COUNTL, F $\longrightarrow$ $\begin{cases} Z = 1 & \text{if COUNTL} = 0 \\ Z = 0 & \text{if COUNTL} \neq 0 \\ N = 1 & \text{if COUNTL} < 0 \\ N = 0 & \text{if COUNTL} > 0 \end{cases}$

MOV instructions

- clrf (1-word)

  Initialize a 1-byte operand with zero (clear), It affects the Z flag.

  ex:

  clrf TEMP $\longrightarrow$ TEMP = 0x00

- setf (1-word)

  Initialize an operand with its maximum value (set)

  ex:

  setf TEMP $\longrightarrow$ TEMP = 0xff

- movlb (1-word)

  move a literal value into the BSR (for “banked” addressing)

  ex:

  movlb 2 $\longrightarrow$ BSR = 0x02

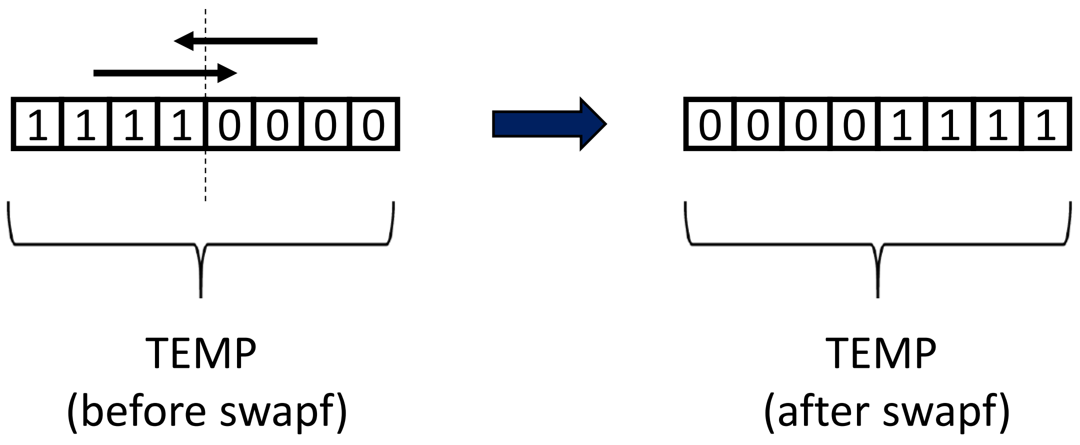
MOV instructions

- swapf (1-word)

  Swap the 4 least significant bits of a 1-bite operand with its 4 most significant bits.

  ex:

  swapf TEMP, F



Single operand instructions

- bsf (1-word)

  set the specified bit of the operand while the others are unchanged

  ex:

  bsf PORTB, 0 $\longrightarrow$ PORTB = _ _ _ _ _ _ _ _1 (_ = prev. value)

- bcf (1-word)

  clear the specified bit of the operand while the others are unchanged

  ex:

  bcf PORTB, 1 $\longrightarrow$ PORTB = _ _ _ _ _ _ _ 0_

- btg (1-word)

  toggle the specified bit of the operand while the others are unchanged

  ex:

  btg PORTB, 2 $\longrightarrow$ $\{$

  PORTB = _ _ _ _ _ 1 _ _ (before instruction)

  PORTB = _ _ _ _ _ 0 _ _ (after instruction)

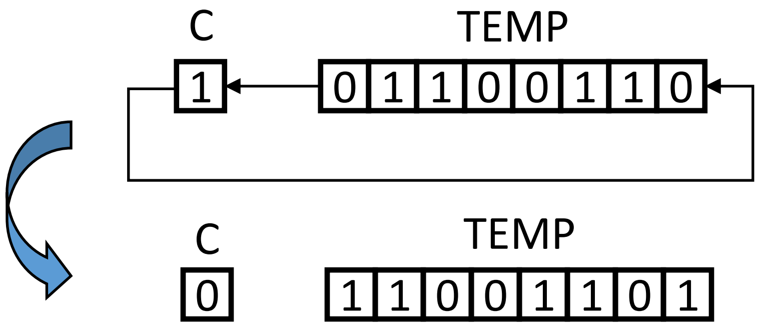
Single operand instructions

- rlcf (1-word)

  implement a 9-bit rotation to the left of an operand through the carry bit

  ex:

  rlcf TEMP, F

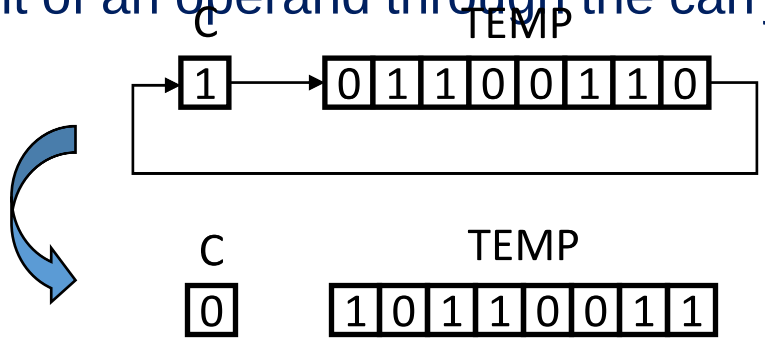


- rrcf (1-word)

  implement a 9-bit rotation to the right of an operand through the carry bit

  ex:

  rrcf TEMP, F



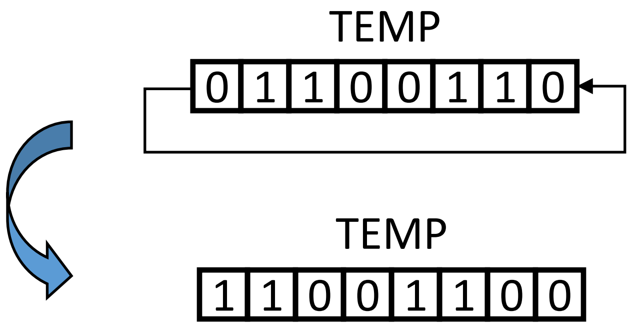
Single operand instructions

- rlncf (1-word)

  implement a 8-bit rotation to the left of an operand

  ex:

  rlncf TEMP, F

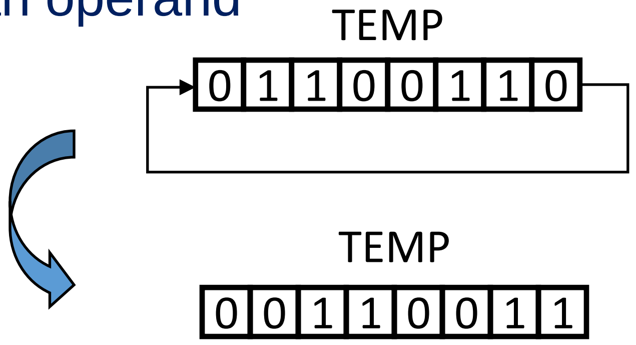


- rrncf (1-word)

  implement a 8-bit rotation to the right of an operand

  ex:

  rrncf TEMP, F



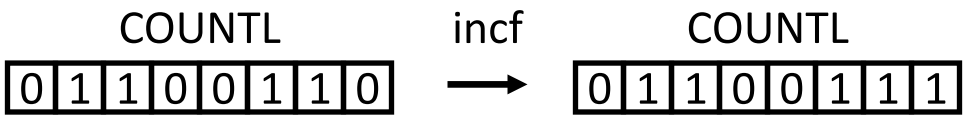
Single operand instructions

- incf (1-word)

  increment a 1-byte operand (it affects any bit of the STATUS register)

  ex:

  incf COUNTL, F

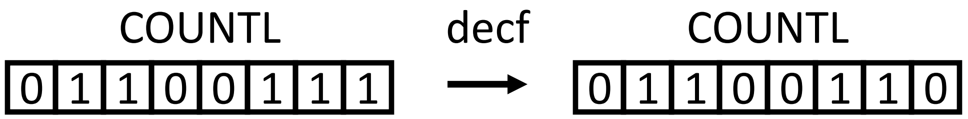


- decf (1-word)

  decrement a 1-byte operand (it affects any bit of the STATUS register)

  ex:

  decf COUNTL, F



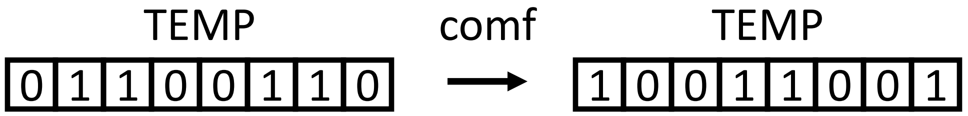
Single operand instructions

- comf (1-word)

  invert the bits of the operand (it affects Z,N bits of the STATUS register)

  ex:

  comf TEMP, F

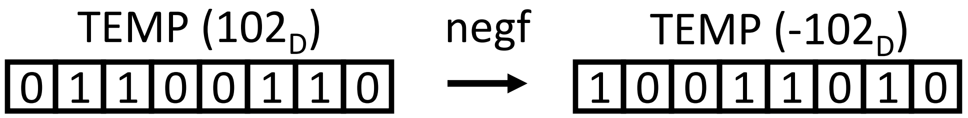


- negf (1-word)

  change the sign of a two's-complement-coded number (it affects any bit of the STATUS register)

  ex:

  negf TEMP



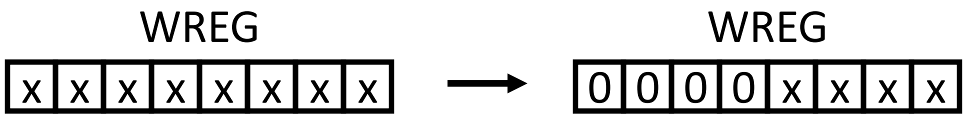
Logical instructions

- andlw (1-word)

  it performs logical AND between literal value and WREG. The result of the operation is kept WREG (it affects Z,N bits of the STATUS register)

  ex:

  andlw B '00001111'



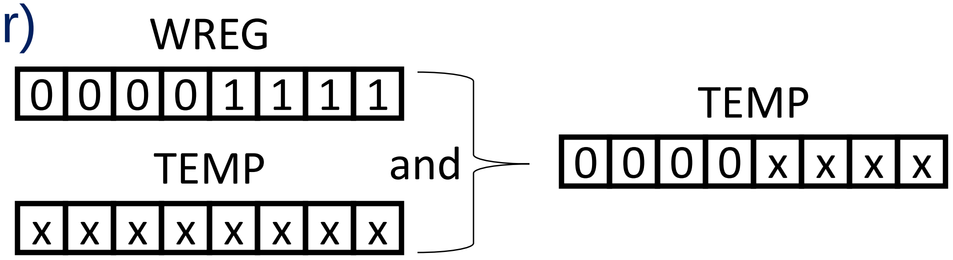
- andwf (1-word)

  it performs logical AND between WREG and the current register (it affects Z, N bits of the STATUS register)

  ex:

  movlw B '00001111'

  andwf TEMP, F



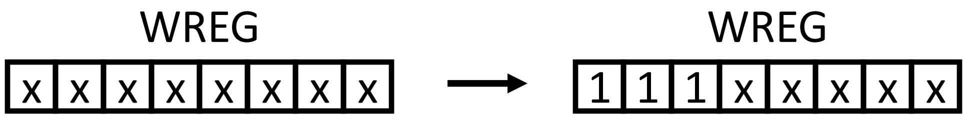
Logical instructions

- iorlw (1-word)

  it performs logical OR between literal value and WREG. The result of the operation is kept WREG (it affects Z,N bits of the STATUS register)

  ex:

  iorlw B '11100000'



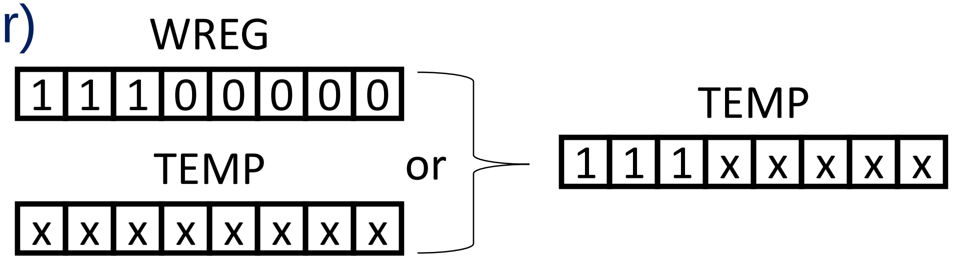
- iorwf (1-word)

  it performs logical OR between WREG and the current register (it affects Z, N bits of the STATUS register)

  ex:

  movlw B '11100000'

  iorwf TEMP, F



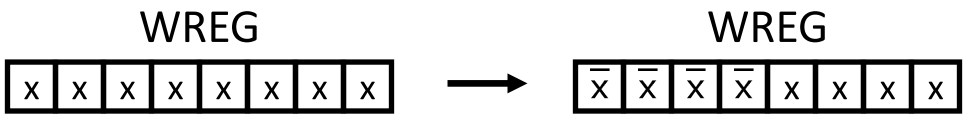
Logical instructions

- xorlw (1-word)

  it performs logical Exclusive-OR between literal value and WREG. The result of the operation is kept WREG (it affects Z,N bits of STATUS reg.)

  ex:

  xorlw B ’11110000’



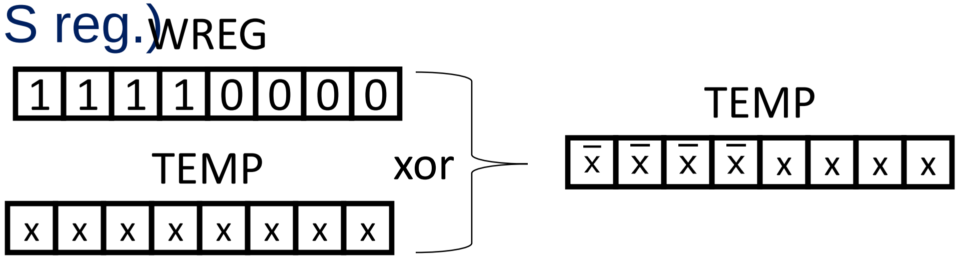
- xorwf (1-word)

  it performs logical Exclusive-OR between WREG and the current register (it affects Z, N bits of the STATUS reg.)

  ex:

  movlw B ’11110000’

  xorwf TEMP, F



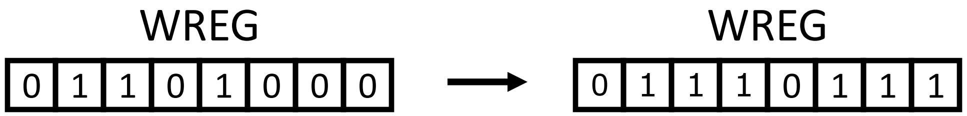
Arithmetic instructions

- addlw (1-word)

  add the 1-byte binary literal value to the 1-byte binary value contained in WREG. The result of the operation is kept WREG (it affects any bit of STATUS reg.)

  ex:

  addlw B ’00001111’



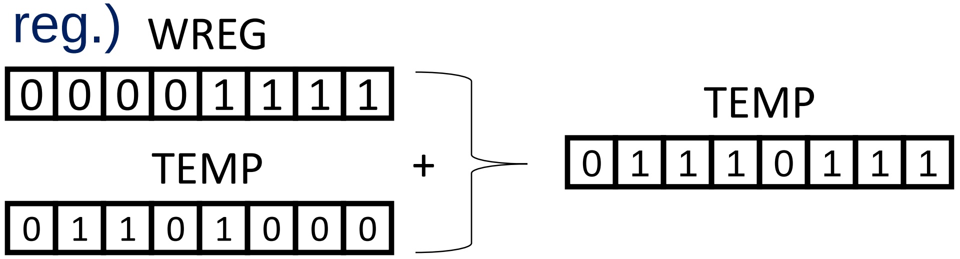
- addwf (1-word)

  add the 1-byte binary value contained in WREG to the current register (it affects any bit of the STATUS reg.)

  ex:

  movlw B ’ 00001111’

  addwf TEMP, F



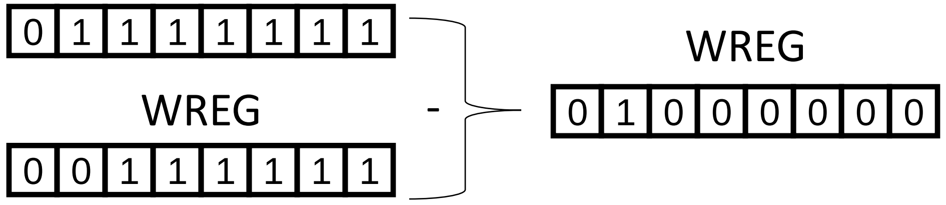
Arithmetic instructions

- sublw (1-word)

  subtract WREG from the literal value. The result of the operation is kept WREG (it affects any bit of STATUS reg.)

  ex:

  sublw B '01111111'



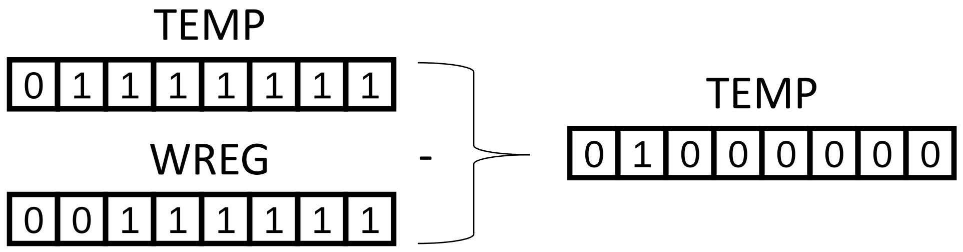
- subwf (1-word)

  subtract WREG from to the current register (it affects any bit of the STATUS reg.)

  ex:

  movlw B ' 01111111'

  subwf TEMP, F



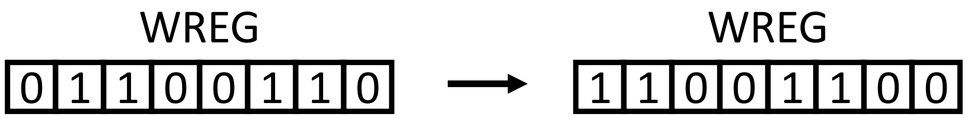
Arithmetic instructions

- mullw (1-word)

  multiply WREG with literal value, puts the result in PRODH:PRODL (WREG is unchanged)

  ex:

  mullw 2



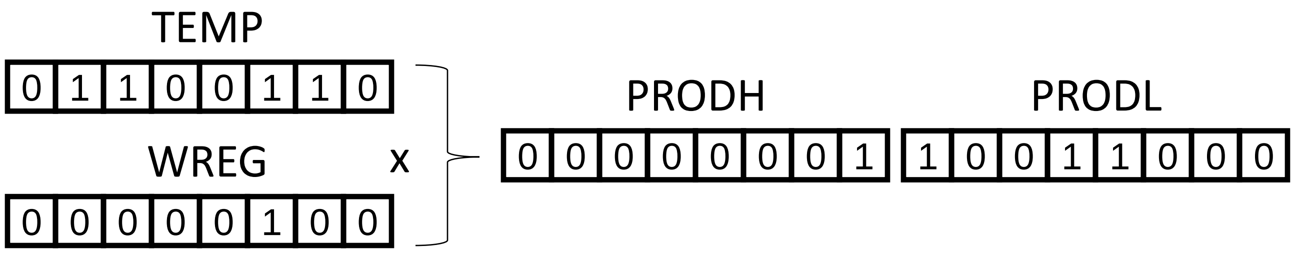
- mulwf (1-word)

  multiply WREG with the current register puts the result in PRODH:PRODL. The original operands remain unchanged

  ex:

  movlw 2

  mulwf TEMP



Unconditional branches

Instructions that force the program to resume execution at the indicated label

- bra (1-word)

  branch to labeled instruction

  within □ 64 one-word instructions

  ex:

```
    mowlw 0f
    movwf TEMP, F
    mowlw 2
    mulwf   TEMP
    bra Here
    .
    .
    .
Here
    .
```

$\}$ Less than 65 one-word instructions

- goto (2-word)

  go to any labeled instruction

  ex:

```
    mowlw 0f
    movwf TEMP, F
    mowlw 2
    mulwf   TEMP
    bra Here
    .
    .
    .
Here
    .
    .
    .
```

$\}$ Any number of instructions

Conditional branches

1-word instructions that force the program to resume execution at the indicated label (within 64 1-word instructions) if a particular condition occurs (based on the STATUS register bits C, Z, N, OV)

- bc

  branch to labeled instruction if carry (C=1)
- bnc

  branch to labeled instruction if not carry (C=0)
- bz

  branch to labeled instruction if zero (Z=1)
- bnz

  branch to labeled instruction if not zero (Z=0)
- bn

  branch to labeled instruction if negative (N=1)
- bnn

  branch to labeled instruction if not negative (N=0)
- bov

  branch to labeled instruction if overflow (OV=1)

Subroutine call and return

Subroutines are a set of instructions that can be placed anywhere in the program and called by different instructions based on their distance from the mainline.
After the return instruction the program resume executing after the call instruction

- rcall (1-word)

  call a subroutine within □ 512

  one-word instructions

  ex:

```
        mowlw 0f
        movwf  TEMP, F
        mowlw 2
        mulwf   TEMP
        rcall MyFunc
        .                  Less than 513 one-word
                           instructions
MyFunc
        movlw 5
        return
```

P2_original.asm

- call (2-word)

  call a subroutine anywhere

  in the program

  ex:

```
         mowlw 0f
         movwf  TEMP, F
         mowlw 2
         mulwf   TEMP
         rcall MyFunc
         .
         .             Any number of
         .             instructions
MyFunc
         .
         movlw 5
         return
```

Assignment