11097 DP2

DSP (Part 2)

Using the DSP Features of the dsPIC® DSC Architecture

DSP Hands-on Class Series

- DSP: Part 1
  - Introduction to Digital Signal Processing using the dsPIC® DSC
- DSP: Part 2
  - Using the DSP Features of the dsPIC DSC Architecture
- DSP: Part 3
  - Using Software Design Tools to Design a DSP Application on the dsPIC DSC

Class Objective

When you finish this class you will:

- Understand and be able to use the DSP-oriented features in the dsPIC® DSC architecture
- Be able to develop simple signal processing algorithms
- Be able to integrate DSP functionality in sensor processing and other applications

Agenda

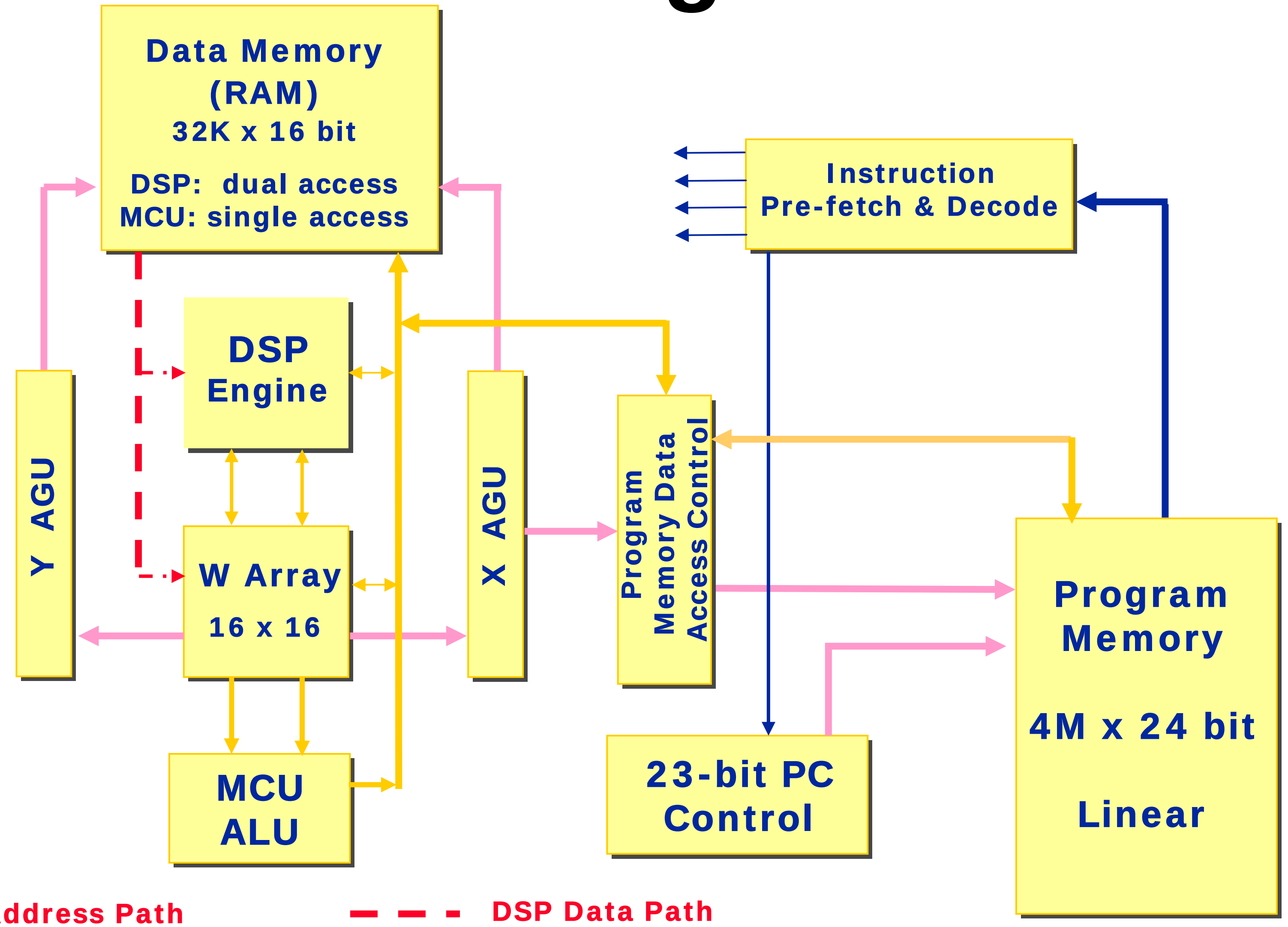
- **Key Features of the dsPIC® DSC Architecture**
- **DSP Accumulator Operations**
  - Hands-on Exercise 1
- **DSP Multiplier and MAC Instructions**
  - Hands-on Exercise 2
- **Other DSP Features**
  - Hands-on Exercise 3

Key Features of the dsPIC® DSC Architecture

dsPIC® DSC Architecture Highlights

- Single CPU integrating MCU & DSP functions
- Modified Harvard Bus Architecture:
- 3 operand instructions: A = B + C
- Extensive Addressing Modes
- 16 x 16-bit general purpose registers
- Fast, deterministic interrupt response
- Flexible software Stack with overflow detection

dsPIC® DSC CPU Block Diagram



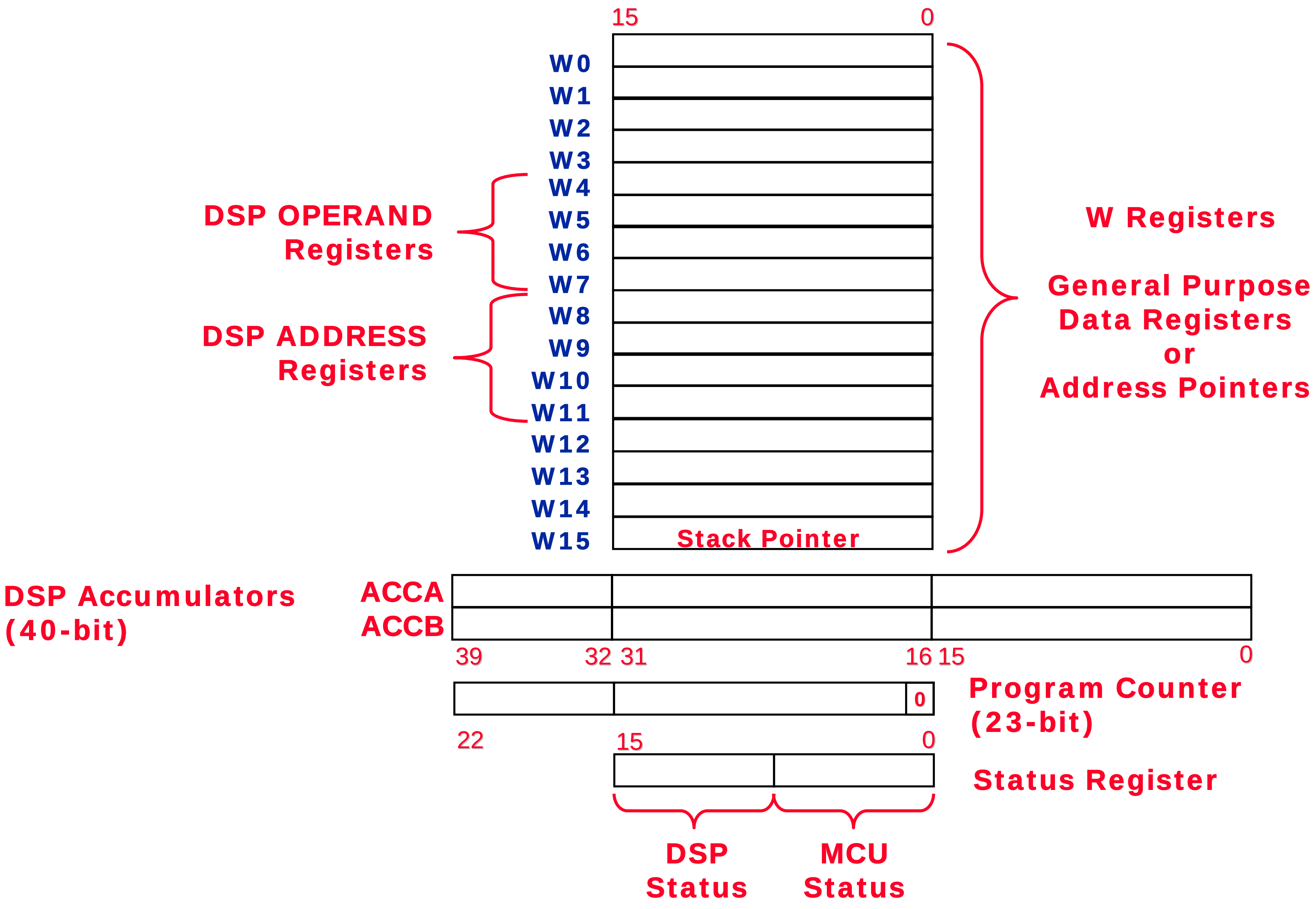
Address Path

MCU / DSP Data Path

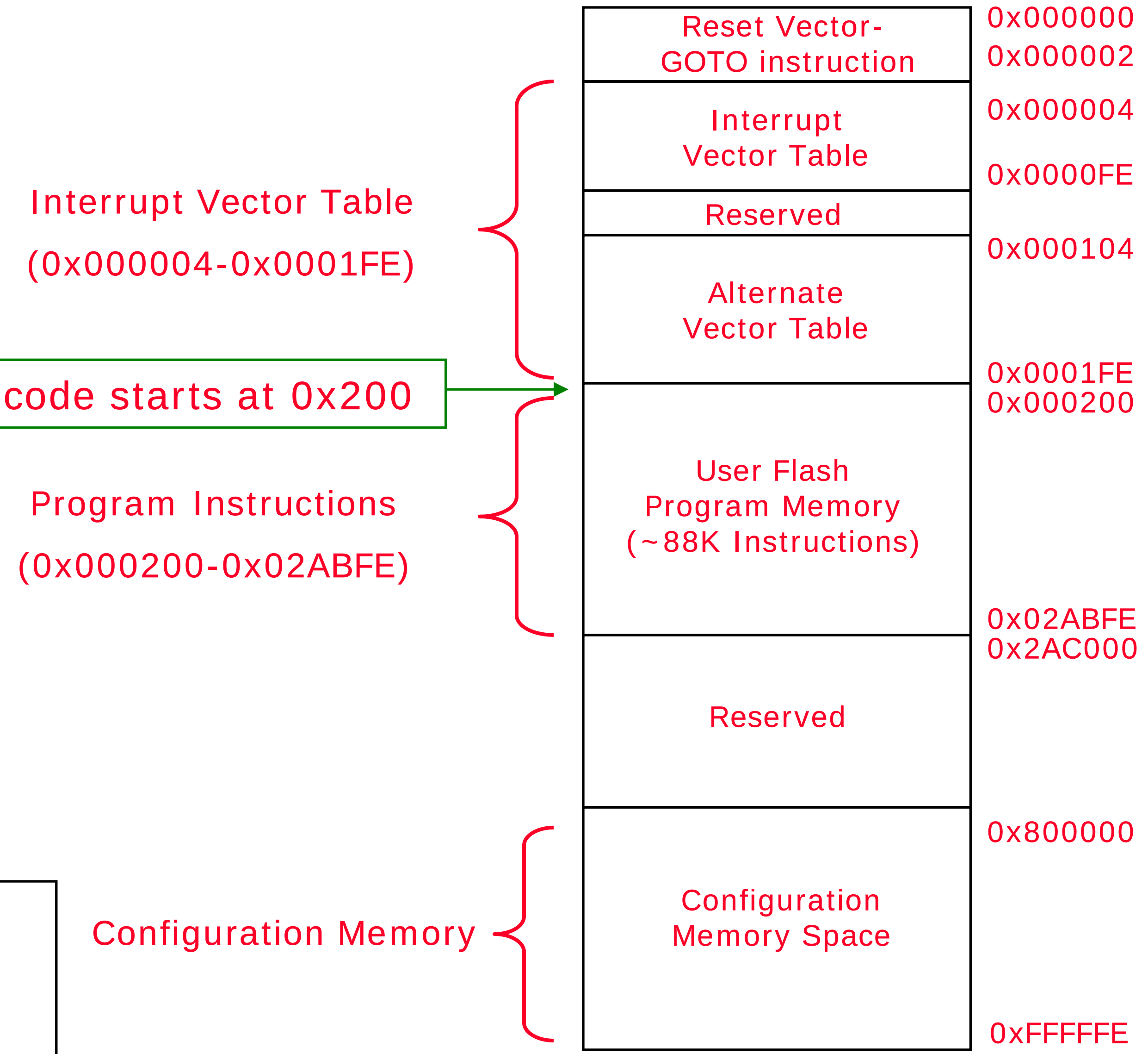
DSP Data Path

Program Data / Control Path

Programmer's Model



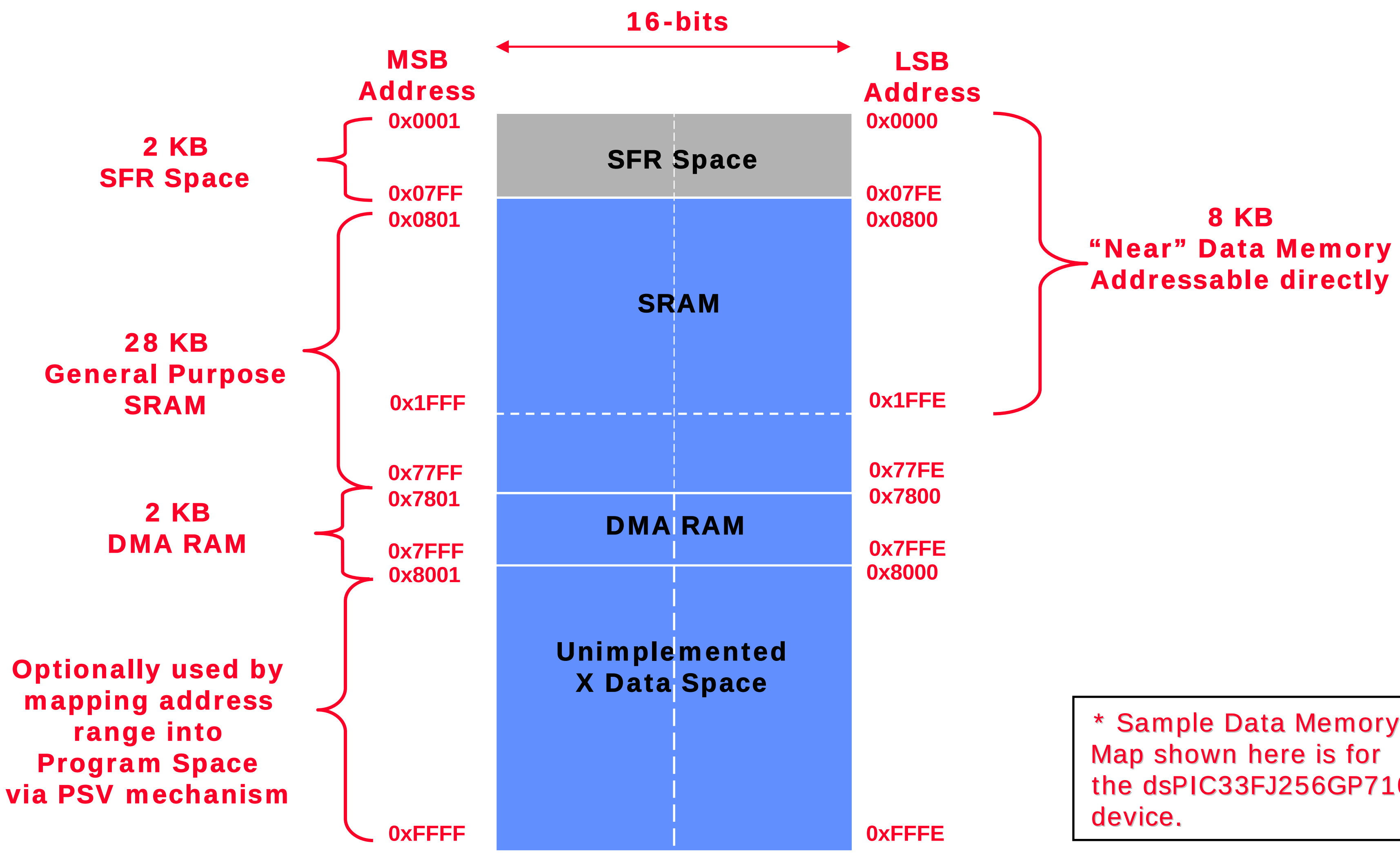
Program Memory Organization*



Executable code starts at 0x200

* Sample Program Memory Map shown here for the dsPIC33FJ256GP710 device.

Data Memory Organization*



* Sample Data Memory Map shown here is for the dsPIC33FJ256GP710 device.

Instruction Set Overview

- **84 base instructions, many variants**
  - Most are one-word (24 bits), only 4 are two-word
  - Most instructions execute in 1 Cycle, except:
    - **Program Flow Changes, Table instructions, Double-word moves, DO instruction: 2 cycles**
    - **Divide instruction: 18 cycles**
- **Instruction groups**
  - Move, Math, Logic, Rotate/Shift, Bit Manipulation, Compare/Skip, Program Flow, Shadow/Stack, Control, DSP

Basic Addressing Modes

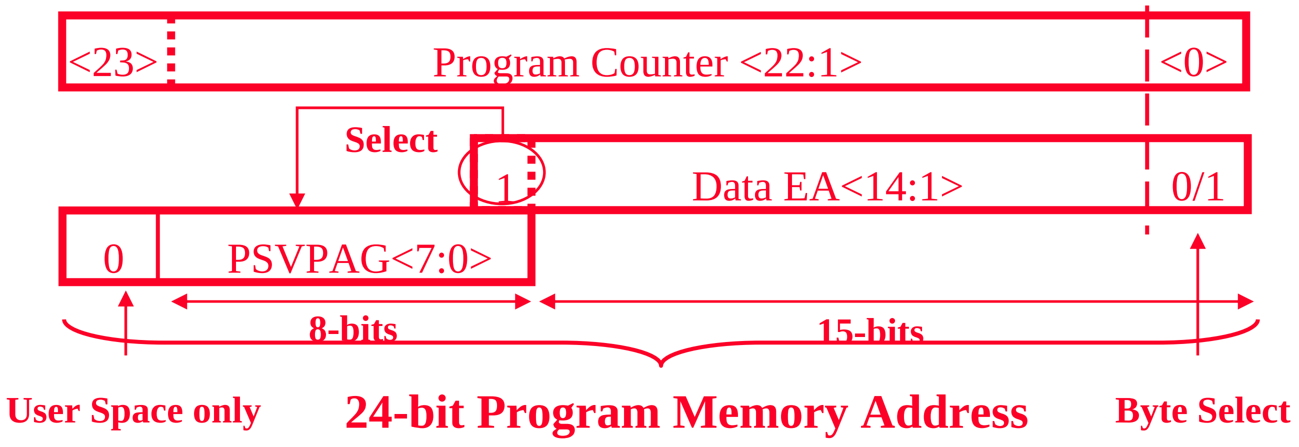
- **File Register / Memory Direct**
- **Register Direct**
- **Register Indirect**
  - with pre-increment or pre-decrement
  - with post-increment or post-decrement
  - with register offset ‡
  - with literal offset ‡
- **Immediate Operand**

‡ Supported in some instructions only

Program Space Visibility

- **32 Kbytes of data space may be mapped into any 16K word ‘page’ in program Flash memory**
  - If PSV bit = 1 and Bit 15 of the data address pointer is ‘1’, then data space window in program space is used
  - PSVPAG (PSV Page register) supplies the upper byte in the 24-bit PS address
  - Supported for reads only
  - May be used for accessing constant coefficients in a FIR or FFT algorithm

Program Space Visibility

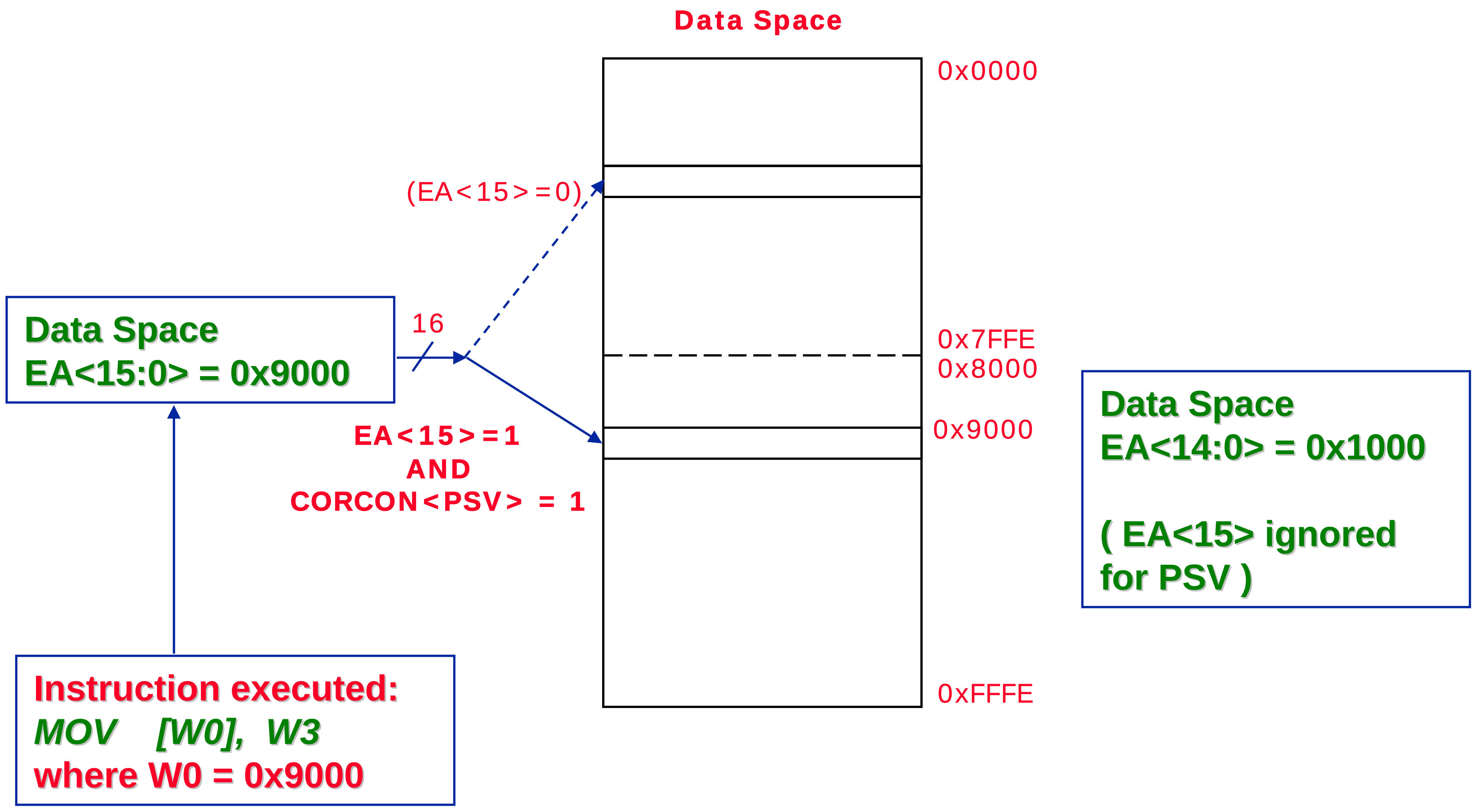


- Only lower 16 bits of program memory location are utilized for mapping
  - Put NOP or illegal opcode in upper byte
- Data accesses from program memory will add 1 or 2 cycles to the instruction

PSV Addressing Example

```
; Enable PSV addressing
BSET    CORCON, #PSV

; Setup pointer
MOV     #psvpage(PROG_ADDR),W0

; Initialize PM Page Boundary SFR
MOV     W0,PSVPAG

; Initialize in-page 16-bit effective address pointer
MOV     #psvoffset(PROG_ADDR),W0

; Read a word from program memory location
MOV     [W0],W3       ; Read the word into W3
```

PSV Addressing Example



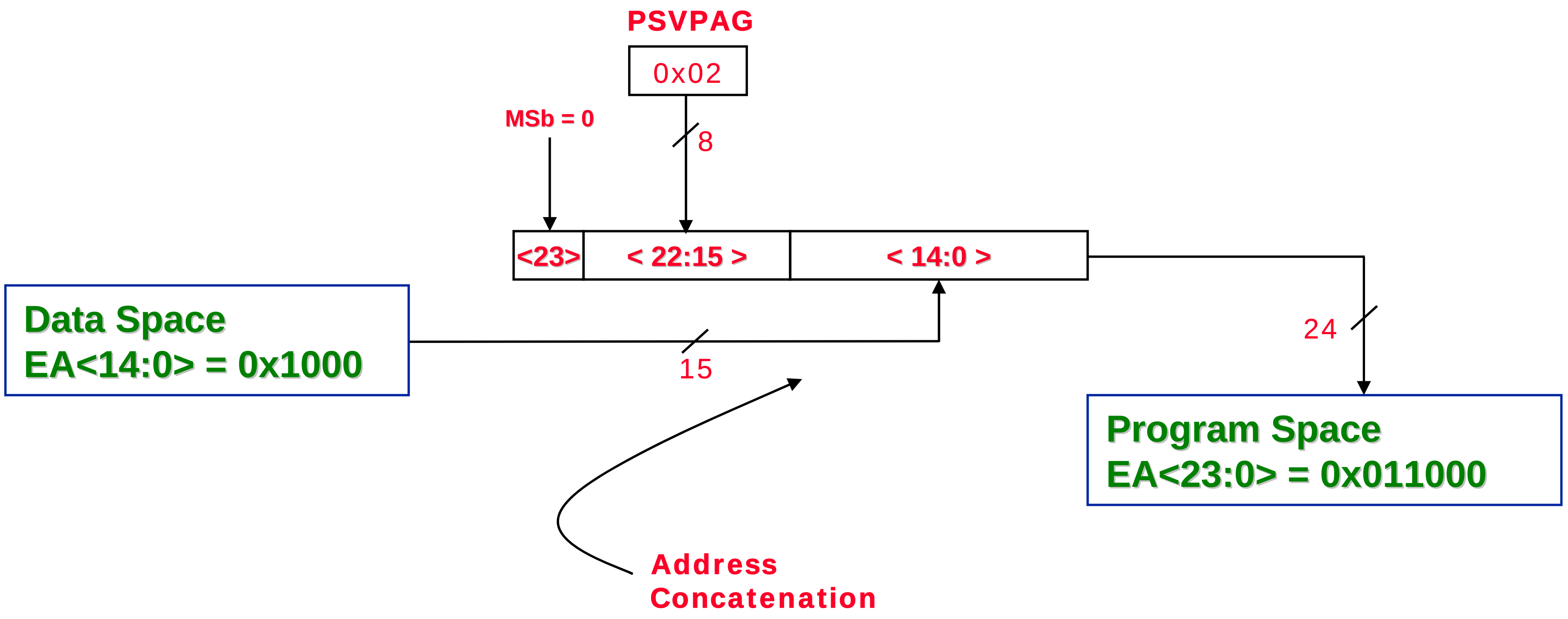
Data Space
EA<15:0> = 0x9000

Data Space
EA<14:0> = 0x1000

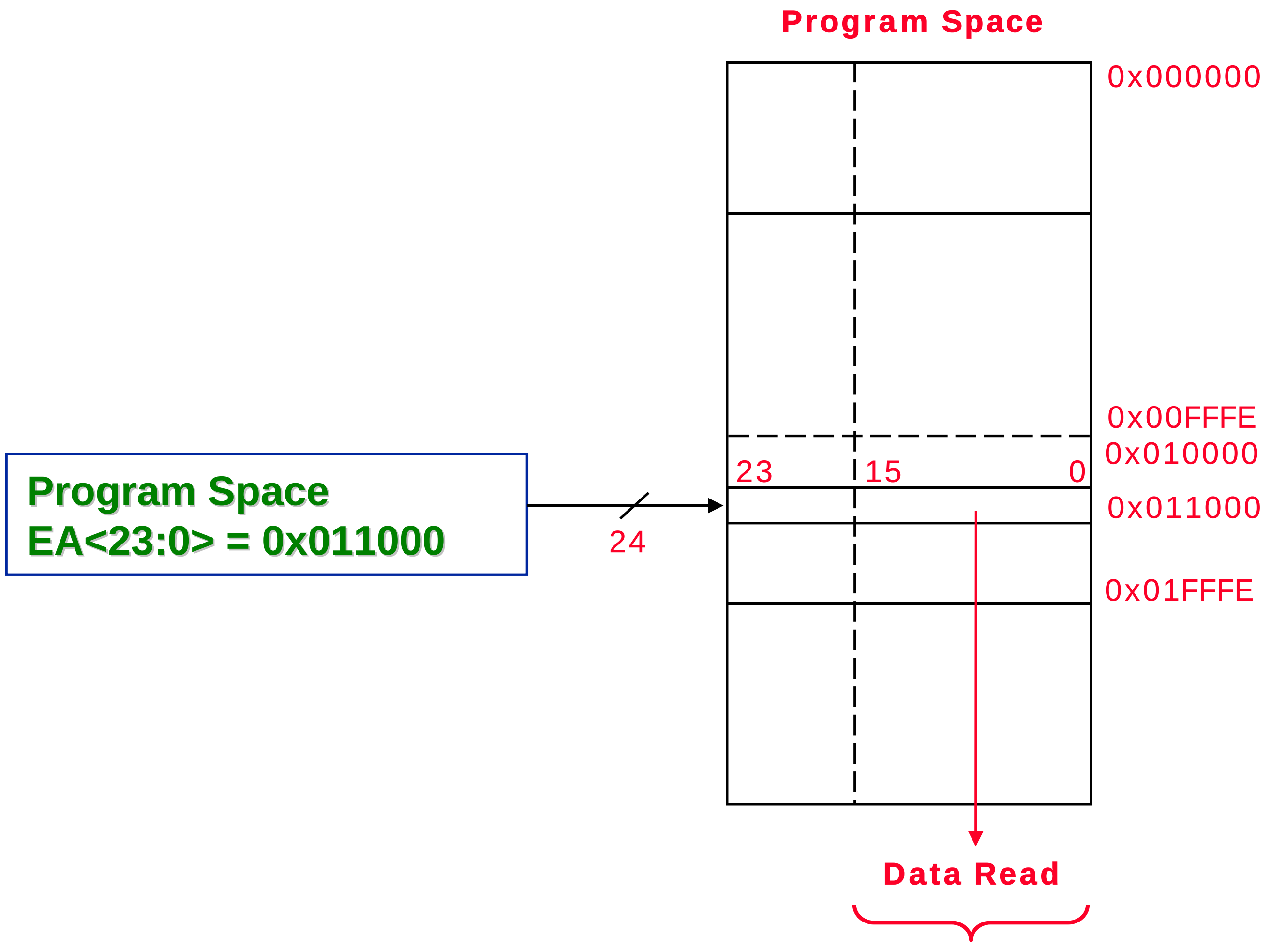
(EA<15> ignored
for PSV)

Instruction executed:
MOV [W0], W3
where W0 = 0x9000

PSV Addressing Example



PSV Addressing Example



Program Space Data Access using Table Instructions

- **Can be used to access**
  - Constants stored in Program Memory
  - Device Configuration Registers, Unit ID, Device ID
- **Table instructions include:**
  - TBLRDL{.b} Ws, Wd
  - TBLRDH{.b} Ws, Wd
  - TBLWTL{.b} Ws, Wd
  - TBLWTH{.b} Ws, Wd

Program Space Data Access using Table Instructions

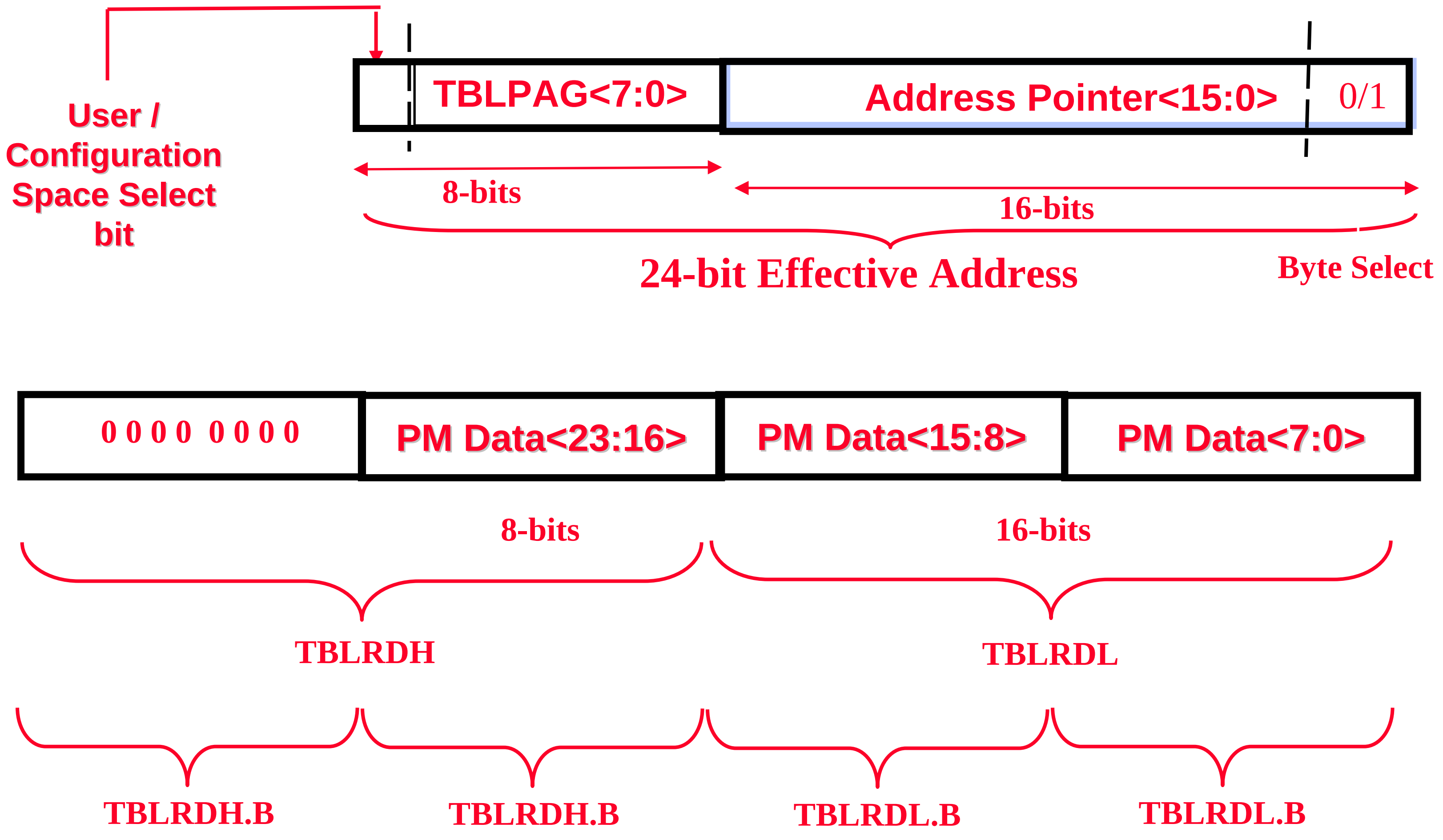


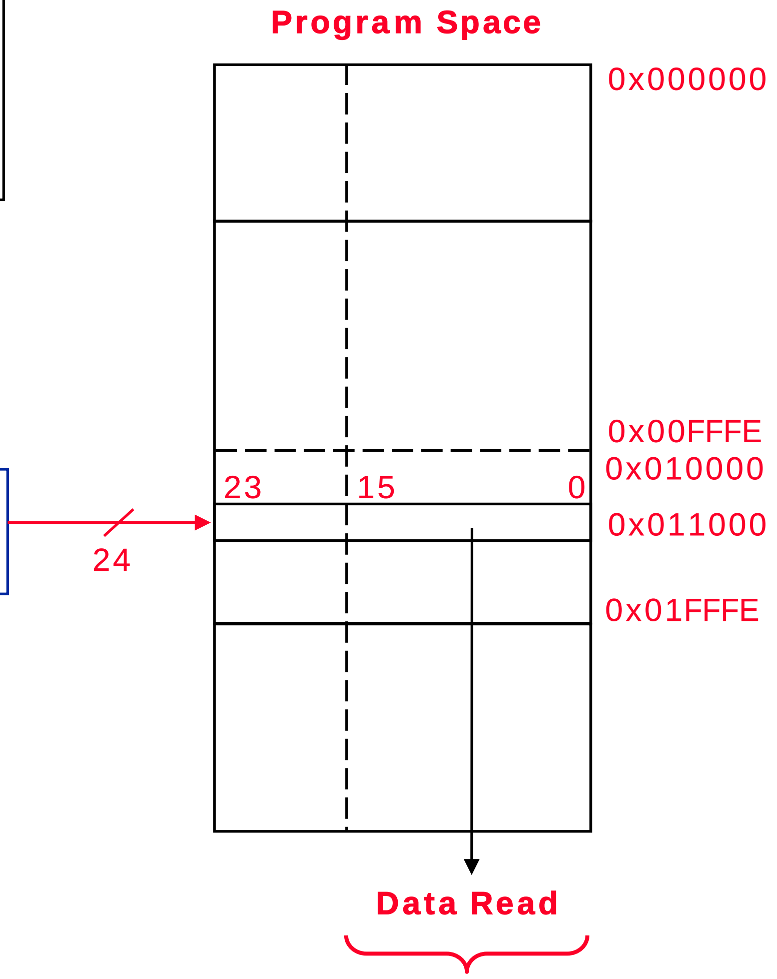
Table Addressing Example

```
; Initialize table page pointer
MOV        #tblpage(PROG_ADDR),W0

; Initialize PM Page Boundary SFR
MOV        W0,TBLPAG

; Initialize in-page 16-bit effective address pointer
MOV        #tbloffset(PROG_ADDR),W0

; Read 3 bytes from program memory location
TBLRDH     [W0],W3      ; Read the high word into W3
TBLRDL.b   [W0++],W4    ; Read the low byte into W4
TBLRDL.b   [W0++],W5    ; Read the middle byte into W5
```

Table Addressing Example

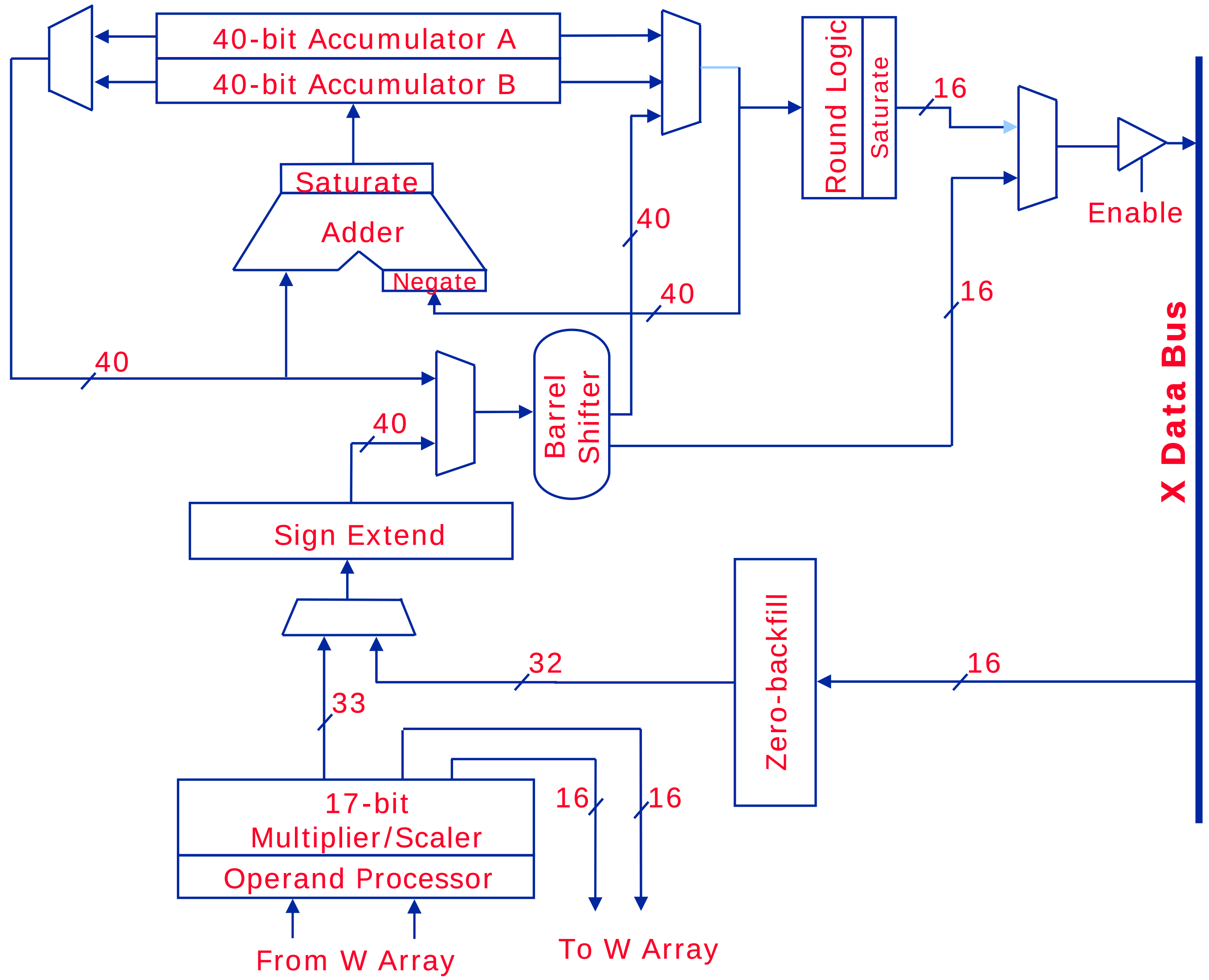
Instruction executed:
TBLRDL [W0], W3
where W0 = 0x1000
and TBLPAG = 0x01

Program Space
EA<23:0> = 0x011000



DSP Accumulator Operations

DSP Engine Block Diagram



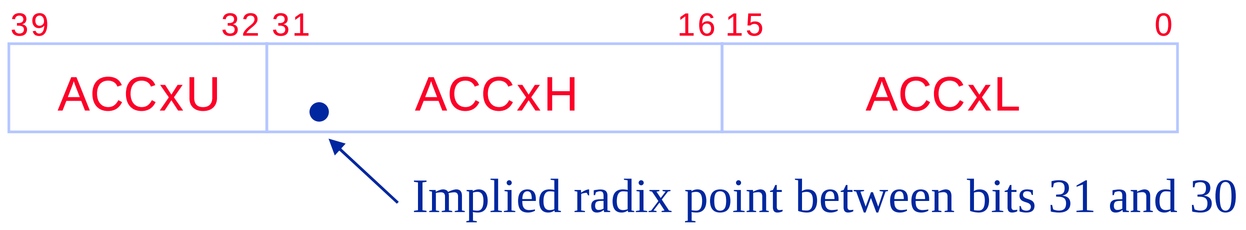
40-bit Accumulators

- **Two 40-bit Accumulators (ACCA, ACCB)**
  - Useful for complex math and multiple filters
  - Overflow detection for accumulator values
  - Multiple accumulator data saturation modes
  - 8 guard bits (ACCxU) for greater dynamic range
  - **LAC** and **SAC** load and store ACCAH or ACCBH
- **All bytes in ACCA/ACCB are memory mapped**

39 32	31 16	15 0
ACCAU	ACCAH	ACCAL
ACCBU	ACCBH	ACCBL

Zero Backfill and Sign Extension

- **Control logic used to interface 16-bit and 32-bit data with the 40-bit accumulator**
  - Think of it as a “data formatter” which allows data to be properly placed in the 40-bit accumulator
- **Sign extension logic operates on ACCxU**
  - ACCxU = 0xFF for negative values
  - ACCxU = 0x00 for positive values
- **Radix point resides between bits 31 and 30**



Zero Backfill and Sign Extension: Examples

- LAC W1, A
  - If W1 = 0xFFF7, ACCA = 0xFFFFF70000
  - W1<15>='1', so ACCAU = 0xFF
  - Lower word of ACCA is cleared (ACCAL = 0x0000)

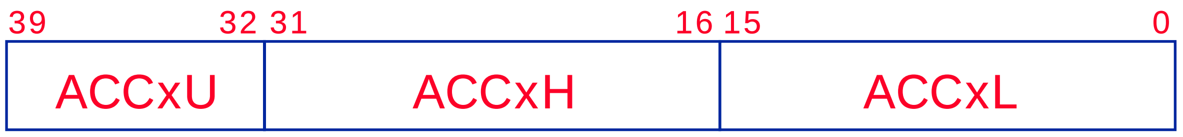
Sign Extend

Zero Backfill

- LAC W2, B
  - If W2 = 0x7FF7, ACCB = 0x007FF70000
  - W2<15>='0', so ACCBU = 0x00
  - Lower word of ACCB is cleared (ACCBL = 0x0000)

Sign Extend

Zero Backfill



40-bit DSP Adder

- **Supports 3 different 40-bit inputs**
  - Zero (used for the DSP **CLR** and **NEG**)
  - Accumulator A / Accumulator B
  - Output of the Sign Extension Logic
- **One adder input may be complemented**
  - Required by **MPY.N**, **MSC**, **NEG** and **SUB**
- **Adder generates output status signals**
  - Overflow & Saturation bits in Status Register
  - Conditional branch instruction support

Accumulator Saturation

- **Two optional saturation modes**
  - 32-bit saturation (Normal Saturation)
  - 40-bit saturation (Super Saturation)
- **When enabled, saturation occurs automatically on any Adder output**
- **Saturation mode selected by CORCON**
  - SATA, SATB, ACCSAT bits
    - **Saturation is disabled by default**
  - SATDW = data space write saturation
    - **Only for** SAC **and Accumulator Write-Back**

Normal Saturation (1.31)

- **ACCxU is always a sign extension of ACCx<31>**
- **Correct resultant sign bit is always preserved during operations**
  - Provides “Signal Clipping” not Sign-Wrapping

- **Largest positive value is 0x007FFFFFFF (~+1.0)**

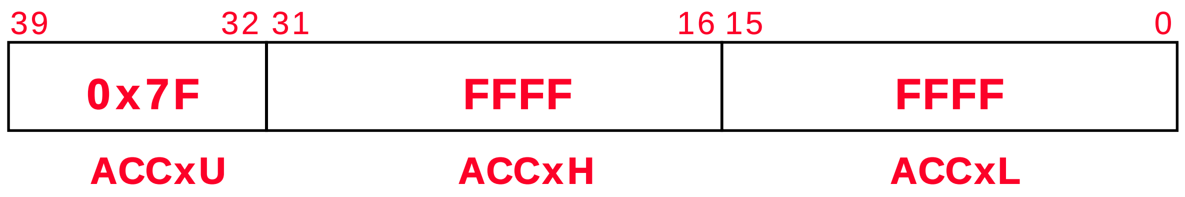
39 32	31 16	15 0
0x00	**7FFF**	**FFFF**
ACCxU	**ACCxH**	**ACCxL**

- **Largest Negative Value is 0xFF80000000 (-1.0)**

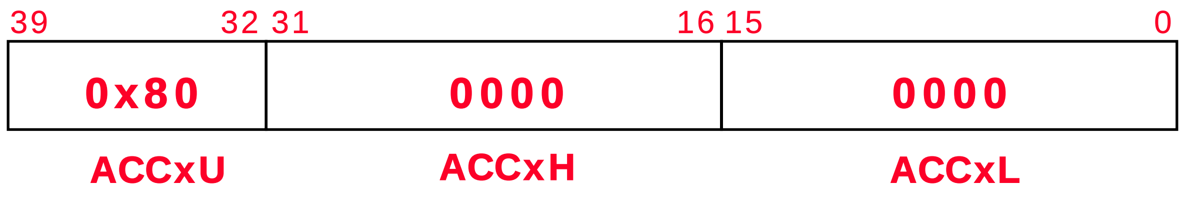
39 32	31 16	15 0
0xFF	**8000**	**0000**
ACCxU	**ACCxH**	**ACCxL**

Super Saturation (9.31)

- **ACCxU contains integer magnitude data**
- **Sign bit located at bit #39 (bit #31 = 2^0)**
- **Allows one to exceed fixed-point range of (-1:1)**
  – Provides greater computational "head room"
  – Can improve the implementation of certain algorithms
- **Largest positive value is 0x7FFFFFFFFF (~ +256.0)**



- **Largest negative value is** 0x8000000000 **(-256.0)**



Saturation Disabled

- **Similar to Super Saturation**
  - ACCxU contains magnitude data
  - Fixed-point range of (-256.0:~+256.0)
- **Adder results are never modified**
- **If sign-bit (bit 39) is destroyed, Overflow bit is set**
  - “Catastrophic Overflow”
  - May generate a math error trap (if COVTE bit is set)

Review Question

- **Which bits in the 40-bit accumulator get loaded or stored when using the LAC or SAC instructions?**
  - A) Bits 0 through 39
  - B) Bits 0 through 15
  - C) Bits 16 through 31

Review Question

- **What is the data range for the 40-bit saturation mode?**
  - A) -256.00 to ~+255.9999
  - B) -1.0 to ~+1.0
  - C) 0x80 0000 0000 to 0x7F FFFF FFFF
  - D) -Pi to +Infinity
  - E) Answers A & C

Four Methods of Accumulator Storage

- SAC
  - Stores truncated ACCxH (optional pre-shift)
- SAC.R
  - Stores rounded ACCxH (optional pre-shift)
- **Accumulator Write-Back**
  - Via **CLR**, **MAC**, **MOVSAC** and **MSC** instructions
  - Stores rounded ACCxH (to W13 or [W13]+=2)

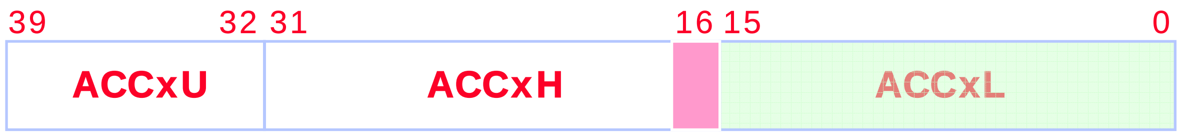
UNIVERSITY OF MICROCHIP UOFM MASTERs 2007

Four Methods of Accumulator Storage

- **`MOV` from memory mapped Accumulator register**
  - Contents of specified register are stored
  - Does not allow for rounding or Data Write Saturation
  - Not recommended for most applications (integer multiplies ok)

Accumulator Rounding

- **Two rounding modes**
  - Conventional (biased) rounds up if bit 15 is set
  - Convergent (unbiased) rounds up if...
    - ACCxL is 0x8000 and ACCxH<0> is “1” **OR**
    - ACCxL is greater than 0x8000
- **Mode is selected by RND bit, CORCON<1>**
  - Default rounding mode is convergent
- **Only utilized by** SAC.R **and ACC Write-Back**
- **Rounding does not affect Accumulator**



Data Write Saturation

- **Provides data saturation even when 1.31 Accumulator Saturation is not used**
- **For SAC, SAC.R and Accumulator Write-Backs, the 1.15 saturated contents of ACCxH are stored**
- **Mode is enabled by SATDW bit**
- **Examples (SATDW = 1, ACCSAT = 0):**
  - **SAC.R   A, W0   ; Stores ACCA to W0**
    - **If ACCA = 0x01 0FFF 1234**
    - **The value stored to W0 = 0x7FFF**
  - **MAC   W4*W5, A, W13   ; Stores ACCB to W13**
    - **If ACCB = 0x9B 0764 4410**
    - **The value stored to W13 = 0x8000**

40-Bit Barrel Shifter

- **Shift range of 16 bits left / 16 bits right**
- **Register shifts (arithmetic and logical)**
  - **SL Wb, #lit4, Wnd** (also **ASR**, **LSR**)
  - **SL Wb, Wns, Wnd** (also **ASR**, **LSR**)
- **Accumulator shifts (arithmetic only)**
  - Saturation always takes effect (if enabled)
  - **SFTAC Acc, #Slit6**
  - **SFTAC Acc, Wb**
  - **ADD**, **LAC**, **SAC** and **SAC.R** support
    - **Reduced range of 8 bits left / 7 bits right**

Review Question

- **Does Data Space Write Saturation modify the source Accumulator?**

  A) Yes, all the time

  B) Yes, but only bits <31:16>

  C) Yes, but only if the value was rounded up

  D) No

Review Question

- **Which instruction will left-shift the contents of Accumulator A by 4 and store the shifted value in W0?**

  A) SFTAC A, #-4

  B) SL A, #4, W0

  C) SAC A, #-4, W0

  D) None of the above

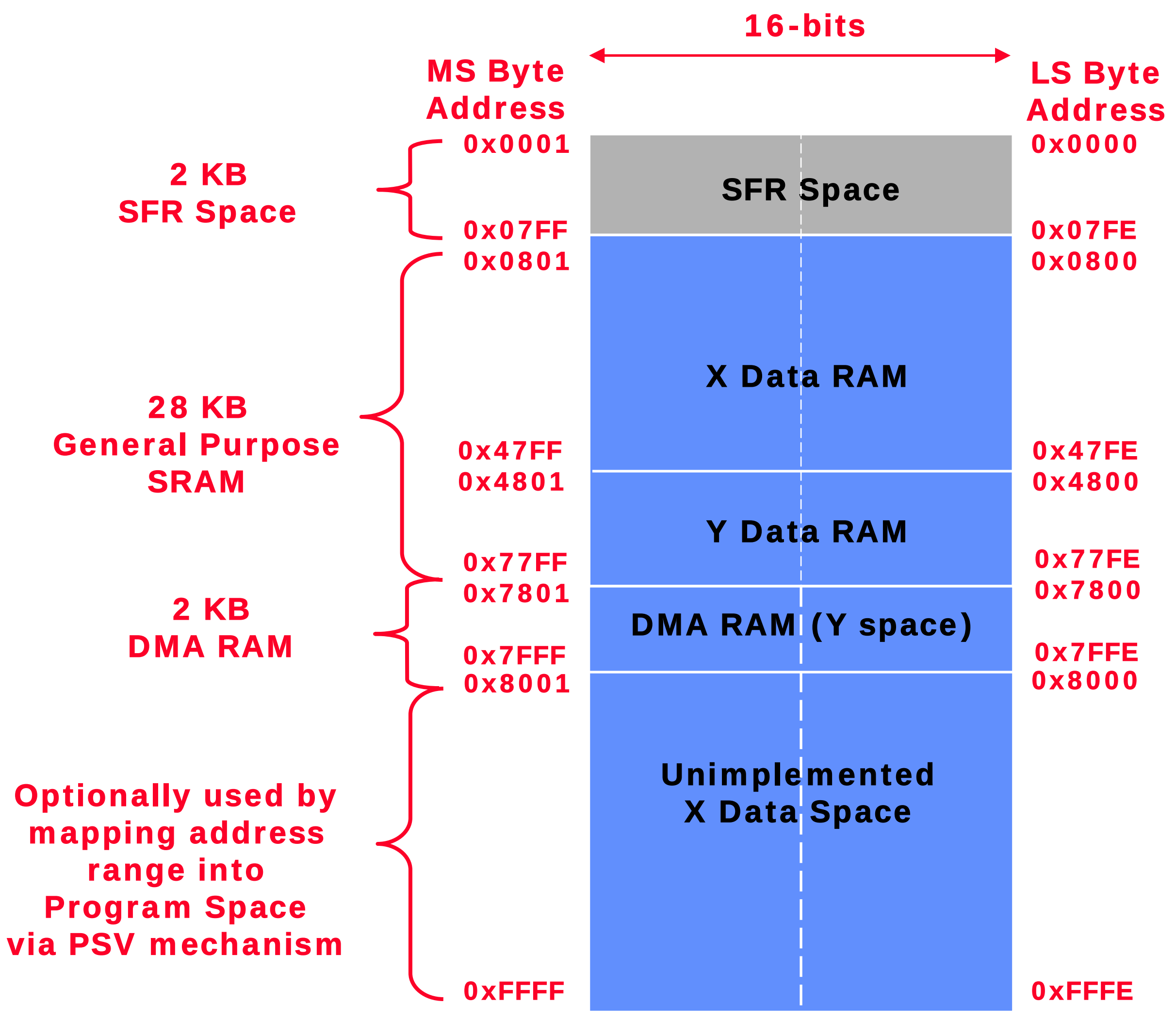
University of Microchip UofM MASTERs 2007

Hands-on Exercise #1

- **Using DSP accumulator-based operations**
- **Process Temperature Sensor (K-type thermocouple) output**
  - Sample and digitize data using ADC
  - Compute block average
  - Perform Cold Junction Compensation using TC1047 temperature sensor IC

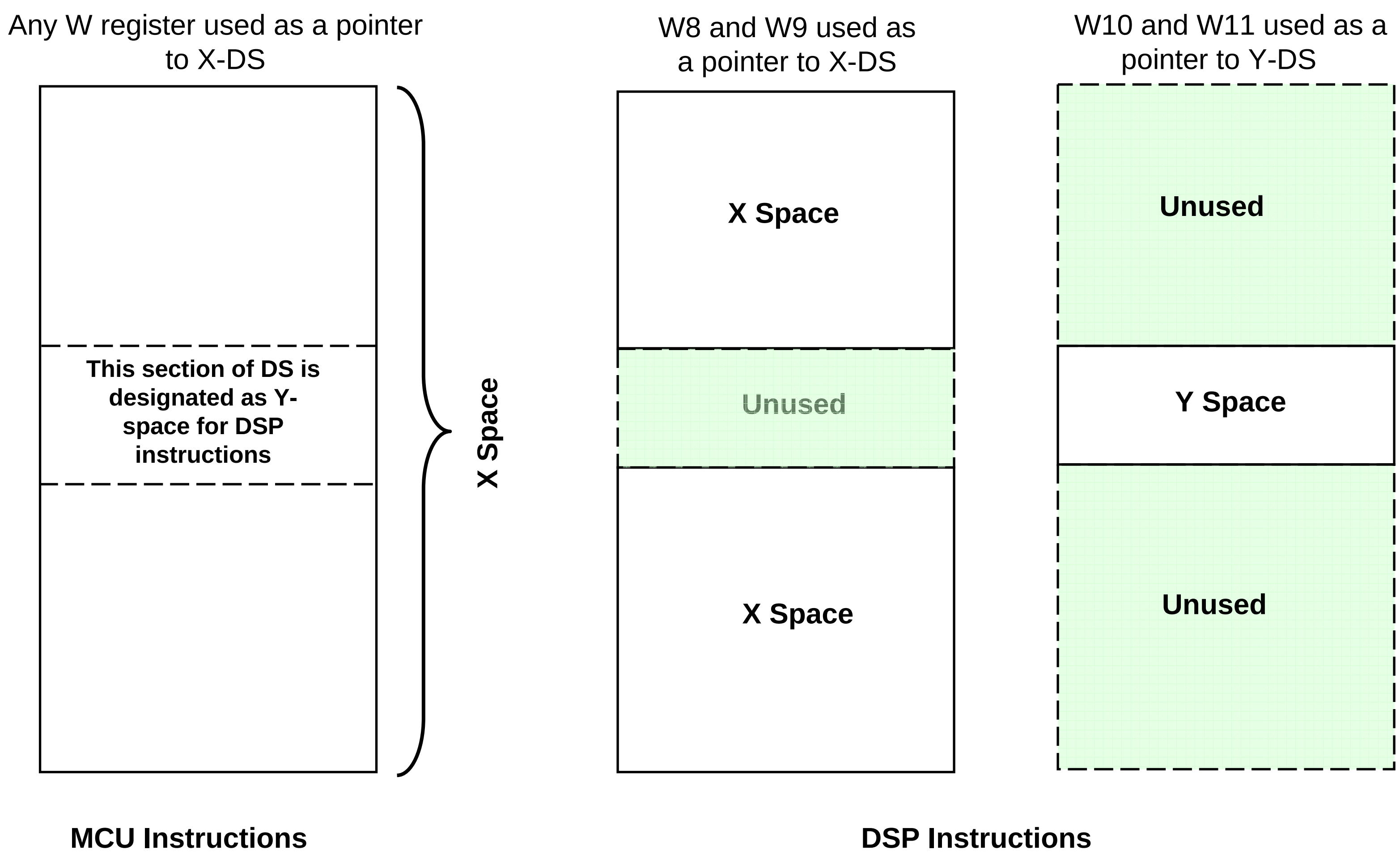
DSP Multiplier and MAC Instructions

Data Memory Map for MAC Instructions – Example*



* Sample Data Memory Map shown here is for the dsPIC33FJ256GP710 device.

Data Memory Organization for DSP MAC Instructions



Any W register used as a pointer to X-DS
W8 and W9 used as a pointer to X-DS
W10 and W11 used as a pointer to Y-DS
This section of DS is designated as Y-space for DSP instructions
X Space
X Space
Unused
X Space
Unused
Y Space
Unused
MCU Instructions
DSP Instructions

Note: 32 KB Section of PS mapped to DS using PSV, is actually mapped to X-DS

17x17-bit Multiplier

- **Multiplier used for both MCU and DSP multiplication instructions**
- **Single cycle operation for all multiplications**
- **Can perform 16x16 multiplication**
  - Operands are 16-bit integer or 1.15 fractional
  - Result is 32-bit integer or 1.31 fractional
- **Why use a 17x17 multiplier when data is 16-bit?**
  - Simplifies signed-unsigned MCU multiplication
  - Correctly performs (-1.0) * (-1.0) operation

MCU Multiplication Operations

- **16-bit Integer Multiplication**
  - Unsigned, Signed and Mixed mode
  - **MUL.UU**, **MUL.SS**, **MUL.US** and **MUL.SU**
  - Produces a 32-bit result
  - Product stored in adjacent working registers
    - **W1:W0, W3:W2, … , W13:W12**
- **WREG Multiply (**MUL.B **or** MUL.W**)**
  - Always uses WREG & any file register
  - Supports 8x8 multiply - generates 16-bit product
  - Results stored in W2 (MUL.B) or W3:W2 (MUL.W)
  - Provides PIC® MCU backward compatibility

DSP Multiplication Operations

- MAC, MPY, MPY.N, MSC, ED and EDAC instructions
- Result is always stored in ACCA/ACCB
- Control bit ‘US’ selects a unsigned/signed multiply operation
- Control bit ‘IF’ selects an integer/fractional multiply operation

DSP Multiplication Operations

- **Fractional multiplication typically used for DSP algorithms**
  - Scaler shifts the multiplier result one bit to the left
    - **Output is 33 bits!**
    - **LS-bit of a fractional result is always cleared**
    - **This maintains proper alignment of the radix point (product has a 2.31 numerical format)**

UNIVERSITY OF MICROCHIP UOFM MASTERs 2007

Signed Fractional Multiply Example (IF = 0, US = 0)

- MPY W4*W5, A
  - If W4 = 0xFFF7 (-0.000274658)
  - If W5 = 0x7FF7 (0.999725341)
  - Multiplier result is 0xFFFB 8051
  - Scaler output is 0xFFF7 00A2

$$\begin{array}{r} -\ 0.000274658 \\ \underline{0.999725341} \\ -.000274582 \end{array}$$

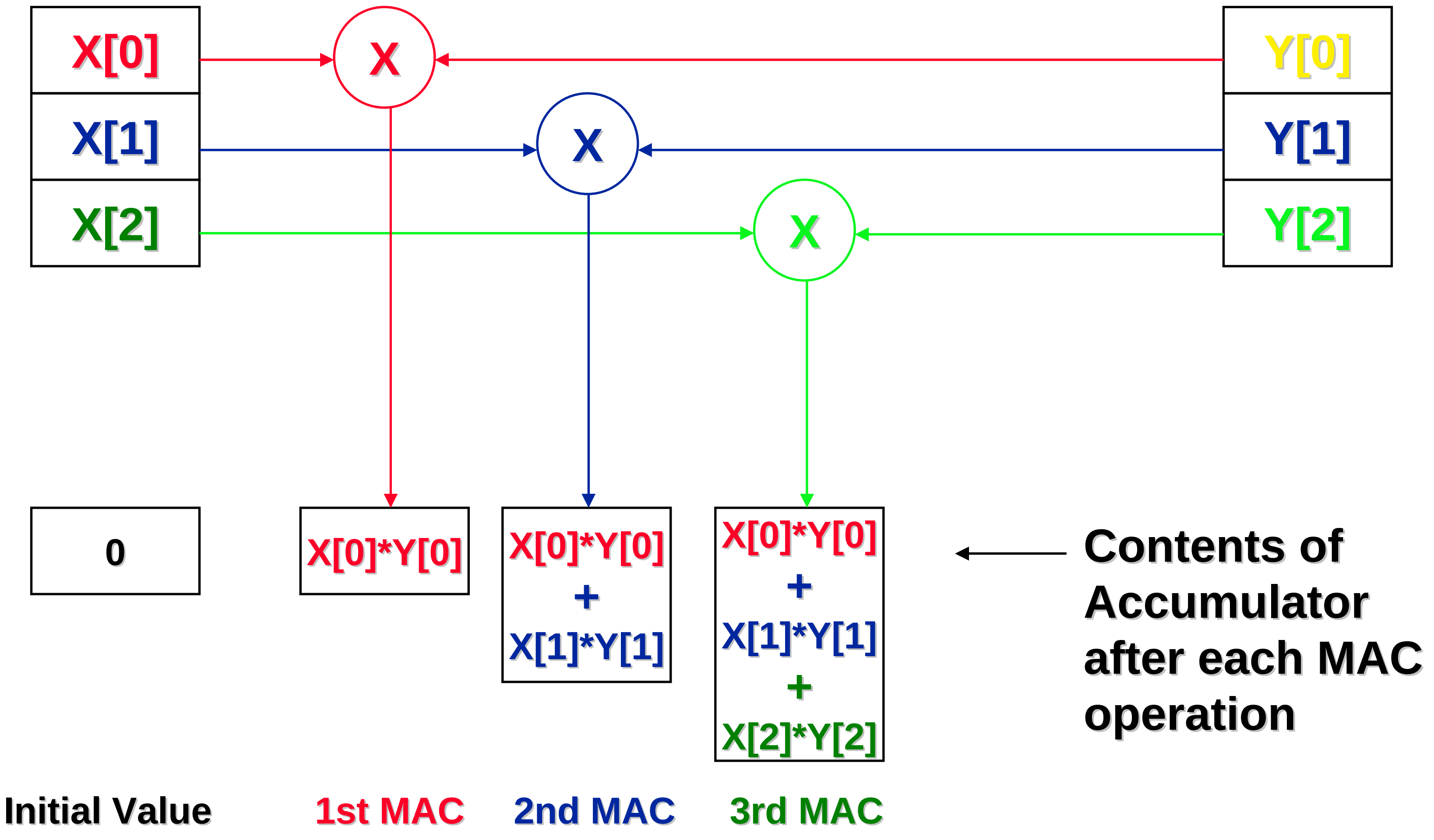
```
           11.111 1111 1111 0111
           00.111 1111 1111 0111
 11.11 1111 1111 1011 1000 0000 0101 0001   ← 2.30 results
11.111 1111 1111 0111 0000 0000 1010 0010   ← 2.31 results after scaler shift
```

Extra sign-extension bit (required for 0x8000*0x8000)

MAC Instructions Overview

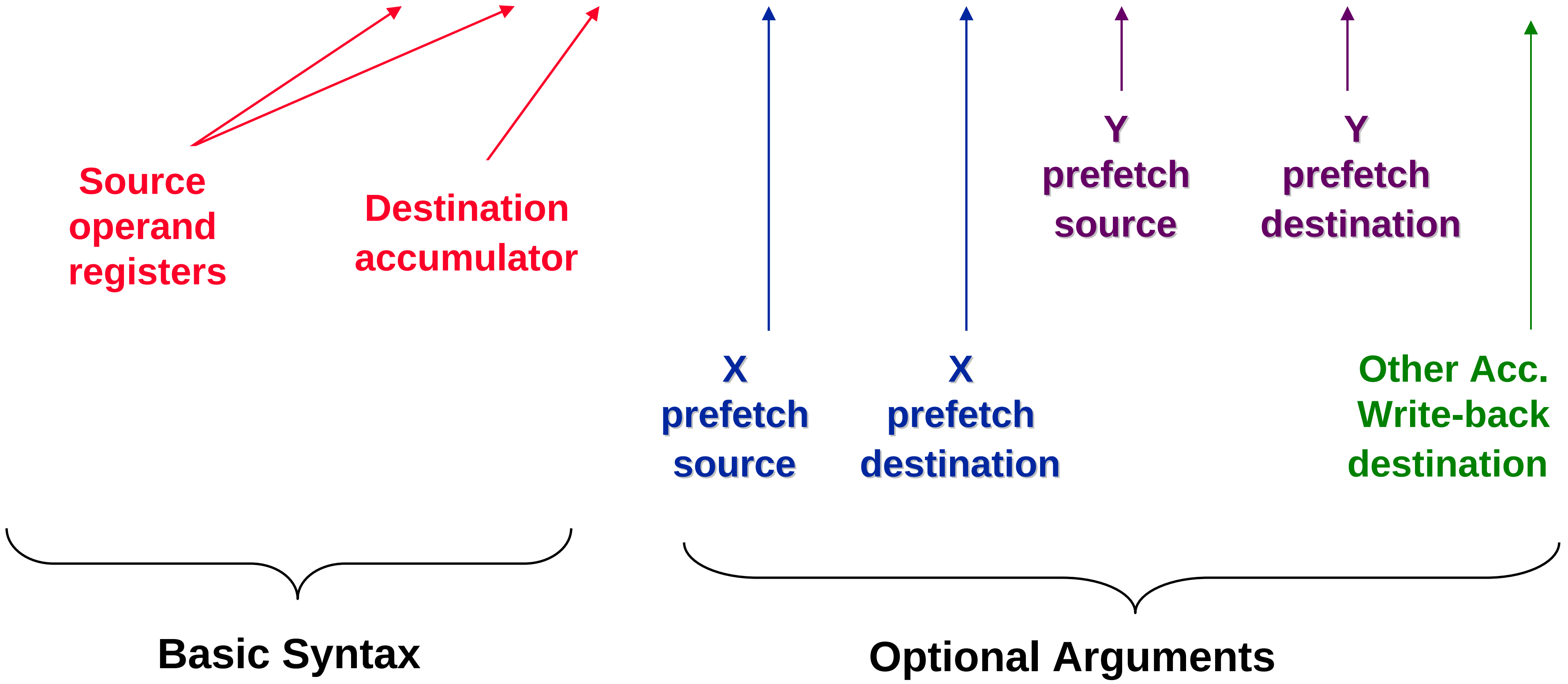
- **MAC = Multiply and Accumulate operation**
  - Basic operation is A = A + x*y
    - **Most fundamental operation used in DSP**
    - **Operates on 2 W registers, chosen from the set W4-W7**
  - Can optionally prefetch 2 source operands for next MAC
    - **1 operand from X memory using W8 or W9**
    - **1 operand from Y memory using W10 or W11**

MAC Instructions and Dot Product



MAC Instruction Syntax

- Sample Instruction
  - MAC W4*W5, A, [W8]+=2, W4, [W10]-=6, W5, W13



MAC Instruction Examples

(1) CLR A, [W8]+=2, W4, [W10]+=2, W5

	Before	After
Accumulator A	0x 60 3425 9718	0x 00 0000 0000
W4	0x 0000	0x 1111
W5	0x 0000	0x 2222

(2) MAC W4*W5, A, [W8]+=2, W4, [W10]+=2, W5

	Before	After
Accumulator A	0x 00 0000 0000	0x 00 048D 0C84
W4	0x 1111	0x 3333
W5	0x 2222	0x 4444

MAC Operand Prefetch Addressing Modes

- **Indirect addressing sub-modes supported in prefetch:**
  - Post-modify (-6, -4, -2, 0, 2, 4 or 6)
  - Indirect with Register offset using W12
    - **[W9+W12] in X space**
    - **[W11+W12] in Y space**
- **Operand addressing examples**

1) MAC   W4*W7, A
2) MAC   W5*W7, B, [W10], W5
3) MAC   W4*W5, A, [W8]+=2, W4, [W10]-=6, W5
4) MPY   W5*W6, B, [W8]-=6, W5, [W11+W12], W6

UNIVERSITY OF MICROCHIP UofM MASTERs 2007

MAC Write-Back Addressing Modes

- **Write-Back (WB) stores the "other" accumulator**
  - if operating on ACCA, ACCB is stored
  - if operating on ACCB, ACCA is stored
  - specified as an optional operand in the MAC
- **WB supports direct or indirect addressing**
  - WB destination is W13 or [W13]+=2
  - [W13] may point anywhere in data memory
- **WB addressing examples**
  - **MSC W4*W5, A, [W10]-=6, W5, W13**
  - **CLR B, [W8]-=6, W4, [W11+W12], W5, [W13]+=2**

UNIVERSITY OF MICROCHIP
U OF M
MASTERs 2007

MAC Instructions Summary

Instruction	*Algebraic Operation*	*ACC WB?*
CLR	A = 0	Yes
ED	$A = (x-y)^2$	No
EDAC	$A = A + (x-y)^2$	No
MAC	$A = A + x*y$	Yes
MAC	$A = A + x^2$	No
MOVSAC	No change in A	Yes
MPY	$A = x*y$	No
MPY.N	$A = – x*y$	No
MSC	$A = A – x*y$	Yes

Note: All MAC class instructions execute in 1 cycle.

Review Question

- **What size is the output of the DSP MPY operation?**
  - A) 8 bits
  - B) 16 bits
  - C) 16 or 32 bits (instruction dependent)
  - D) 33 or 34 bits (depends how you see it)
  - E) None of the above

Review Question

- **Which of the following is a valid syntax?**
  - A) MAC A, W4*W5
  - B) MAC W4*W5, A, [W8++], W4
  - C) MAC W4*W5, A, [W8]+=2, W4

Hands-on Exercise #2

- **Using DSP multiplier-based operations**
  - Use unsigned fractional mode
- **Linearize Temperature Sensor (K-type thermocouple) output**
  - Compute 2nd order polynomial using MAC-class operations and standard coefficient values

Other DSP Features

REPEAT Instruction

- **REPEAT #lit14 or REPEAT Wn**
  - Repeat next instruction “N+1” times, where “N” may be a 14-bit constant or any W register (14 least significant bits)
  - Instruction to be repeated is latched
  - Loop count held in RCOUNT register
  - RA status bit gets set when RCOUNT>1, to disable instruction prefetches
  - RA status bit gets cleared when RCOUNT=1 to enable instruction prefetches

REPEAT Instruction

- **REPEAT Instruction is interruptible**
  - RA bit is stacked (SRL) on interrupt
  - RCOUNT must be stacked before executing any REPEAT loops within ISR
  - Clearing RCOUNT helps exit the loop
- **REPEAT loop restrictions**
  - Any instruction can follow a REPEAT except:
    - **Program Flow instructions, `DO` or `REPEAT`**
    - **`DISI`, `LNK`, `PWRSAV`, `ULNK`, `RESET`**

DO Instruction

- **DO will repeat a set of instructions "N+1" times**
  - "N" may be a 14-bit constant or any working register (lower 14 bits)
  - Loop start address stored in DOSTART
  - Loop end address stored in DOEND
  - Loop count held in DCOUNT register
- **Loop size up to 32K instructions**
  - Sequential order not required

DO Instruction

- **7 levels of DO loops can exist in SW**
  - Implies 6 levels of nested DO loops allowed
  - DO SFRs are shadowed automatically when a second DO instruction is executed (single level of nesting)
  - Additional levels of nesting require user to stack the DO SFRs
  - DO level status bits, DL<2:0>, are automatically updated by hardware
  - DA bit set when DL<2:0>≠'0' and DCOUNT > 1

DO Instruction

- **Setting the EDT bit terminates DO loop execution (drops the DO level by one)**
  - DL bits modified by HW to reflect new DO level
- **DO loop is interruptible**
  - ISRs can contain DO loops
    - **On entry into ISR, the DA or DL<2:0> bits should be checked to determine if the DO SFRs need to be stacked prior to executing any DO loops within the ISR**

Divide Operations

- **Integer Divide Support**
  - DIV.S and DIV.SD for signed integer divide
  - DIV.U and DIV.UD for unsigned integer divide
- **Fractional Divide Support**
  - DIVF signed fractional divide
- **Iterative divide - must be repeated 18 times**

```
REPEAT #17
DIVF     W8, W9
```

Data Normalization

- **Why is data scaling needed?**
  - Suppose sign-extended fractional data is obtained from a 12-bit ADC
  - Data processing may be performed on the ADC samples
  - Scaling data up prior to data processing allows us to use the full 16-bit dynamic range

Normalization Example: Positive Fractional Data

0x06AF = 0000 0110 1010 1111_b = 0.0522_d

Left-shift by 4 bits

Load the value into a register

FBCL returns -4 or 0xFFFC

Shift the register by -4

0x6AF0 = 0110 1010 1111 0000_b = 0.8355_d

Normalization Example: Negative Fractional Data

0xFF64 = 1111 1111 0110 0100_b = -0.0048_d

Load the value into a register

FBCL returns -7 or 0xFFF9

Shift the register by -7

0xB200 = 1011 0010 0000 0000_b = -0.6094

Block Data Normalization

- **By what quantity should a block of samples be scaled?**
  - For example, if the largest and smallest samples in the set are :
    - **0x06AF (+ve) =** $\mathbf{000\ \underbrace{0110\ 1010\ 1111_b}_{\text{Original 12-bit sample from ADC}}}$
      - This can be scaled up by 4 bits
    - **0xFF64 (-ve) =** $\mathbf{111\ \underbrace{1111\ 0110\ 0100_b}_{\text{Original 12-bit sample from ADC}}}$
      - Similarly, this can be scaled up by 7 bits
  - Therefore, use 4 as the “scale-factor” (not 7)

A Normalization Technique

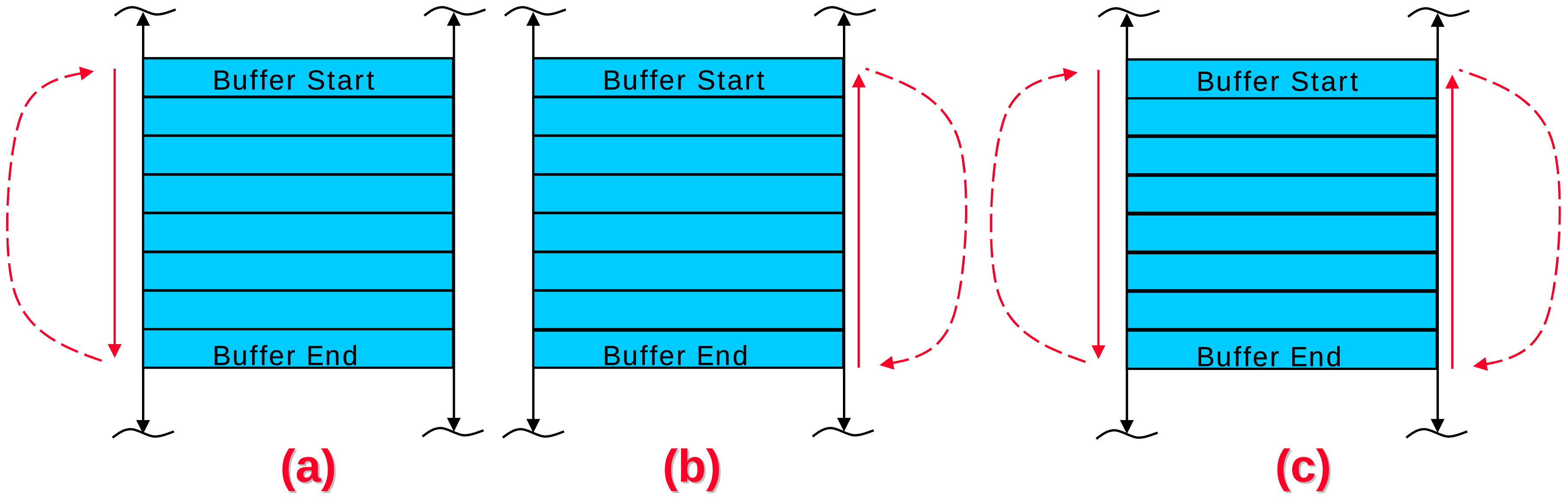
- **Obtain scale-factor for the entire block, using FBCL instruction**
- **How is block scaling performed?**
  - Load each sample into the accumulator
    - **Use the LAC instruction**
  - Left-Shift the accumulator by the scale-factor obtained earlier
    - **Use the SFTAC instruction**
  - Store high word of accumulator to buffer
    - **Use the SAC instruction**

Review Question

- **Can a DO instruction be used to repeat a single instruction?**
  - A) Yes
  - B) No
  - C) Only for a non-zero loop count

Modulo Addressing: Introduction

- **Modulo or “Circular” Addressing**
  - Eliminates the software overhead associated with Effective Address (EA) correction, e.g. in FIR filters
  - Data address boundary checks are performed in hardware
  - Three types of modulo buffers: (a) Incrementing, (b) Decrementing, and (c) Bidirectional.



Modulo Addressing: Introduction

- Only operates with the Indirect addressing mode
- Two independent modulo buffers may be set up for X and Y data space
- Operates for words as well as bytes
- Maximum possible buffer size in data memory is 64 KB

Modulo Addressing: Configuration

- **Addressing mode is set in the MODCON SFR**
  - XMODEN enables mod addressing for X Data Space
  - YMODEN enables mod addressing for Y Data Space
  - Linear addressing is the default addressing mode
- **One working register is selected for modulo addressing in X space, and one for Y space**
  - XWM<3:0> selects modulo addressing W reg. for X Data Space
  - YWM <3:0> selects modulo addressing W reg. for Y Data Space

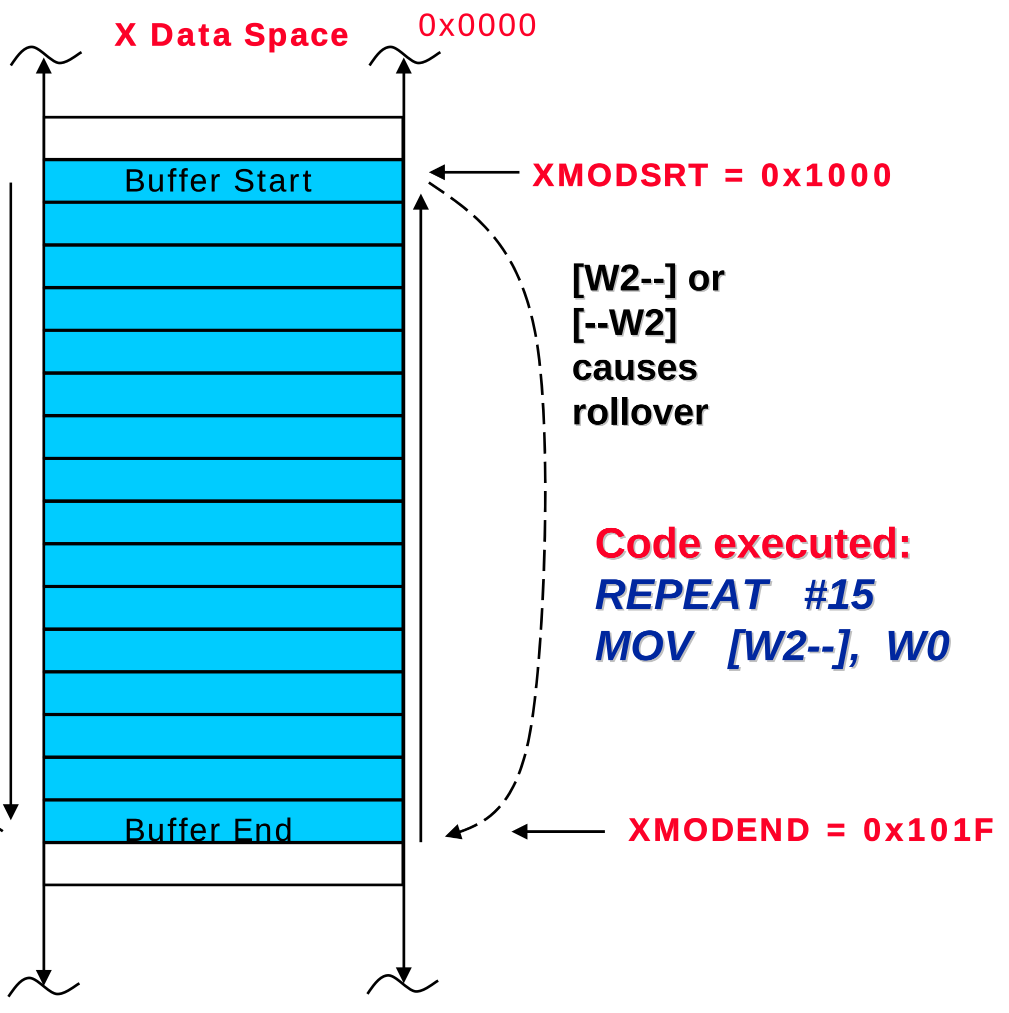
15															0
XMODEN	YMODEN							YWM3	YWM2	YWM1	YWM0	XWM3	XWM2	XWM1	XWM0

MODCON REGISTER

Modulo Addressing: Configuration

- **XMODSRT, YMODSRT**
  - Contains the buffer start address
  - Address must satisfy certain conditions, depending on the type of buffer needed
- **XMODEND, YMODEND**
  - Contains the address of last byte occupied by the buffer
  - Address must satisfy certain conditions, depending on the type of buffer needed

Modulo Addressing: Modulo Buffer Example



[W2--] or [--W2] causes rollover

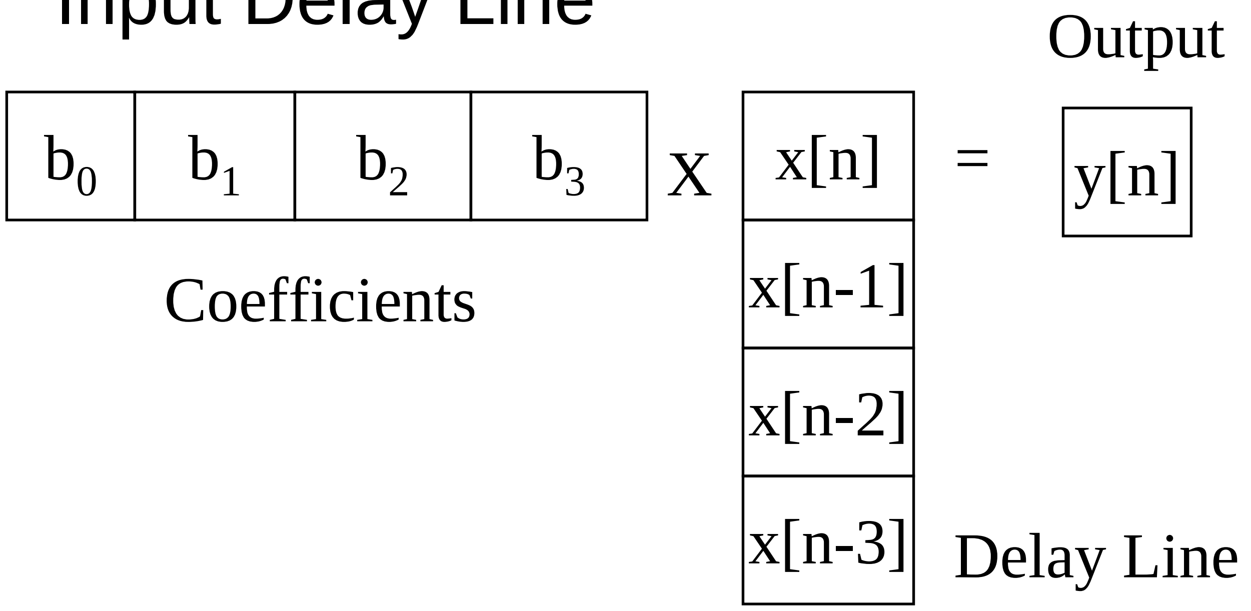
Code executed:
REPEAT #15
MOV [W2--], W0

Code executed:
REPEAT #15
MOV [W2++], W0

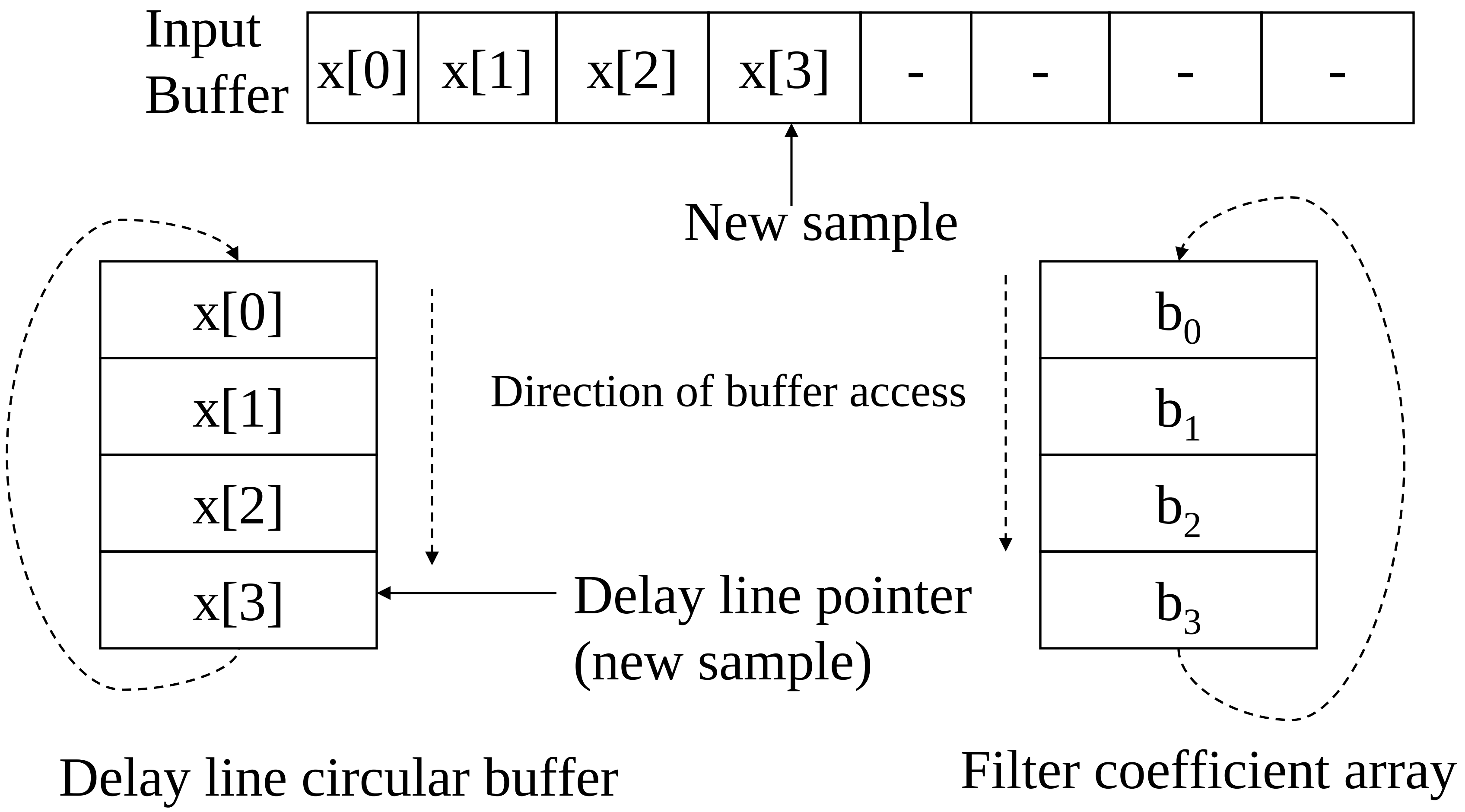
[W2++] or [++W2] causes rollover

FIR Filter and Modulo Buffers

- **Single Sample FIR Filter Operation (4 taps)**
- **Vector Dot Product of…**
  - Coefficient Array
  - Input Delay Line



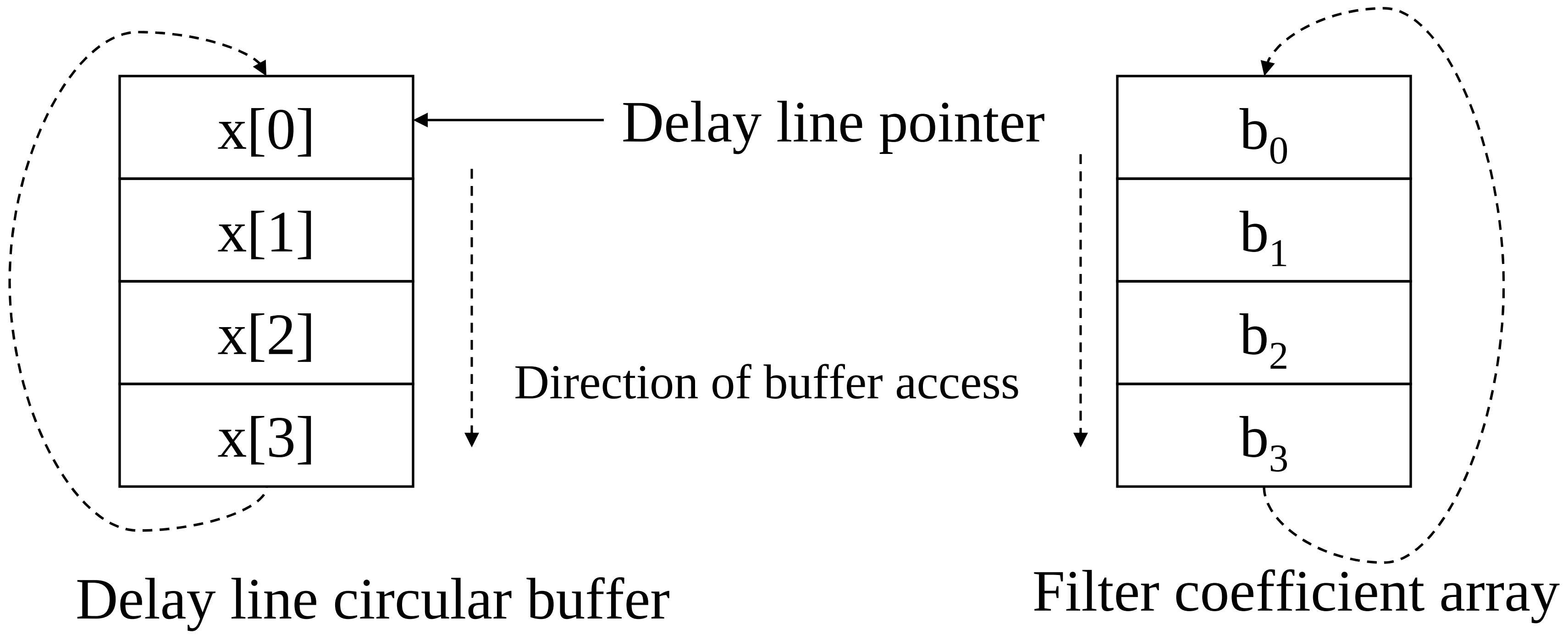
FIR Filter and Modulo Buffers



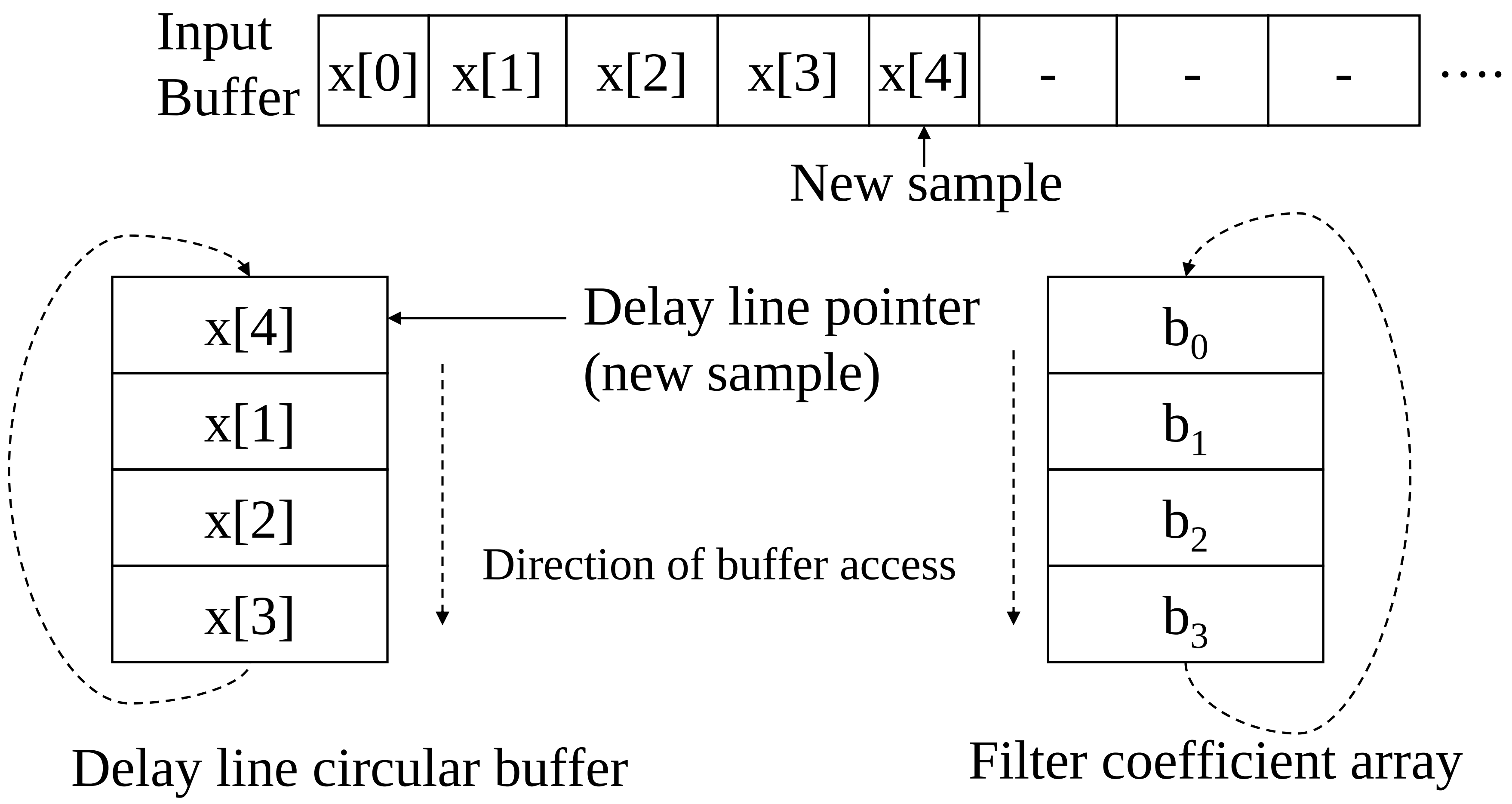
FIR Filter and Modulo Buffers

FIR output for n = 3:

$y[3] = x[3]*b_0+x[2]*b_1+x[1]*b_2+x[0]*b_3$



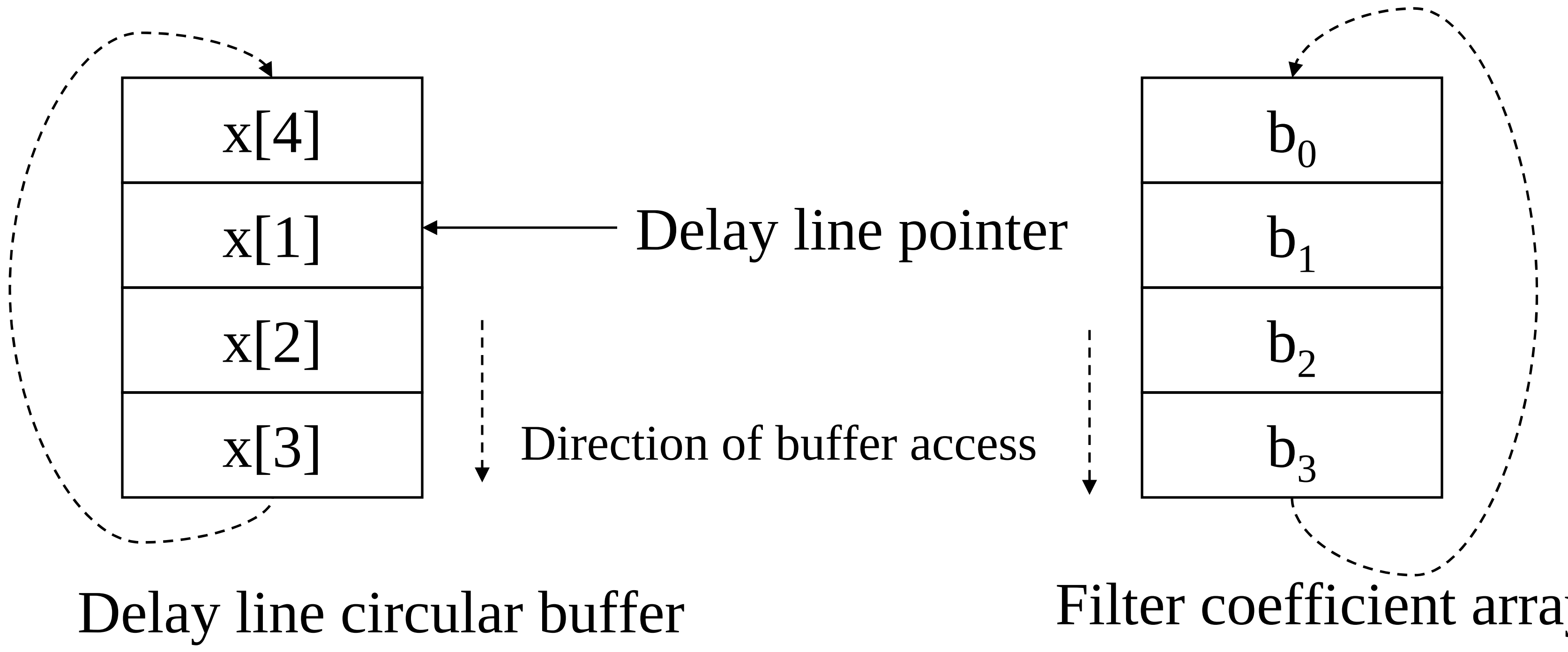
FIR Filter and Modulo Buffers



FIR Filter and Modulo Buffers

FIR output for n = 4:

$y[4] = x[4]*b_0+x[3]*b_1+x[2]*b_2+x[1]*b_3$



Bit-Reversed Addressing: Introduction

- **Bit-Reversed (BR) addressing is used for efficiently implementing radix-2 FFT**

Decimal	Binary
(0)	000
(1)	001
(2)	010
(3)	011
(4)	100
(5)	101
(6)	110
(7)	111

Bit-Reversed Reordering

Decimal	Binary
(0)	000
(4)	100
(2)	010
(6)	110
(1)	001
(5)	101
(3)	011
(7)	111

Bit-Reversed Addressing: Configuration

15															0
				BWM3	BWM2	BWM1	BWM0								

BWM - Bit Reversed Addressing Register Select

MODCON REGISTER

15															0
BREN	XB14	XB13	XB12	XB11	XB10	XB9	XB8	XB7	XB6	XB5	XB4	XB3	XB2	XB1	XB0

BREN - Bit Reversed Addressing Enable

XB<14:0> - X AGU Bit Reversed Modifier

XBREV REGISTER

Bit-Reversed Addressing: 16-word Buffer Example

(0)	0x1000
(1)	0x1002
(2)	0x1004
(3)	0x1006
(4)	0x1008
(5)	0x100A
(6)	0x100C
(7)	0x100E
(8)	0x1010
(9)	0x1012
(10)	0x1014
(11)	0x1016
(12)	0x1018
(13)	0x101A
(14)	0x101C
(15)	0x101E

Example :

MODCON = 0x0800
(W8 is bit-reversed buffer pointer)

Current W8 = 0x1000

XBREV = 0x0008
Modifier = XB*2 = 0x0010

```
     0001 0000 0000 0000   (0x1000)
 +   0000 0000 0001 0000   (0x0010)
--------------------------------------
     0001 0000 0001 0000   (0x1010)
--------------------------------------
```

CARRY DIRECTION >>>>>

Modified W8 = 0x1010

(0)	0x1000
(8)	0x1010
(4)	0x1008
(12)	0x1018
(2)	0x1004
(10)	0x1014
(6)	0x100C
(14)	0x101C
(1)	0x1002
(9)	0x1012
(5)	0x100A
(13)	0x101A
(3)	0x1006
(11)	0x1016
(7)	0x100E
(15)	0x101E

Review Question

- **Which of the following addressing modes does not support modulo address correction?**
  - A) Indirect with Register Offset
  - B) Indirect with Post-Increment
  - C) File Register Addressing

Hands-on Exercise #3

- **Notch Filter**
  - Remove single-tone (1700 Hz) noise from a 3200 Hz signal obtained through the generic sensor interface
  - Perform block data normalization using method described earlier
  - Configure Modulo Addressing prior to performing FIR filtering

UNIVERSITY OF MICROCHIP UOFM MASTERS 2007

Summary

- dsPIC® DSC architecture optimized for computationally intensive tasks
- DSP Accumulator operations
- DSP Multiplier operations
- Hardware loop control
- Modulo and Bit-Reversed Addressing modes
- Extensive suite of DSP software tools and libraries

References

- **dsPIC33F Architecture Documentation**
  - dsPIC30F/33F Programmer's Reference Manual (DS70157)
  - Family Reference Manual – CPU (DS70204)
  - Family Reference Manual – 10/12-bit ADC without DMA (DS70210)
- **dsPIC® DSC Software Tools Documentation**
  - MPLAB® C30 C Compiler User's Guide (DS51284)
  - MPLAB ASM30, MPLAB LINK30 and Utilities User's Guide (DS51317)

UNIVERSITY OF MICROCHIP
UOFM
MASTERs 2007

Trademarks