INTRODUCTION TO ADSP-2181

Sundar G. Sankaran
Systems Group - DSP Research Laboratory
The Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg VA 24061-0111.
e-mail: ssgandhi@ee.vt.edu

PROCESSOR OVERVIEW

- 16 Bit Fixed-Point Processor
- On-chip Memory, Timer, Serial Ports
- Three Independent Computational Units
- Hardware Circular Buffer (Filtering)
- Automatic Bit Reversed Addressing (FFT)
- Zero-Overhead Looping and Branching

MEMORY

- Modified Harvard Architecture
  - » 24 Bit Program Memory (PM) to Store Both Instructions and Data
  - » 16 Bit Data Memory (DM) to Store Data
- On-Chip Memory
  - » 16 Kwords each of on-chip PM and DM
- External Byte Memory
  - » Upto 4 Mbytes
  - » Used as Boot Memory Also

REGISTERS

- Inputs for Computations
  - » Computational Registers (16 Bits)
    - AX0, AX1, AY0, AY1, AR, AF
    - MX0, MX1, MY0, MY1, MF, MR0, MR1, MR2
    - SR1, SR0, SI, SE, SB
- To Hold Address of Memory Locations
  - » Index Registers (14 Bits)
    - I0, I1, …, I7
  - » Following also help in Addressing
    - M0, M1,…,M7; L0, L1,…,L7

Two-sets of computational registers are available. Why?

COMPUTATIONAL UNITS

- Arithmetic and Logic Unit (ALU)
  - » Addition, Subtraction, NOT, AND, OR, XOR, and Division
- Multiply and Accumulate Unit (MAC)
  - » Multiply, Multiply-Add (Filtering), and Multiply-Subtract
- Shifter
  - » Logical and Arithmetic Shifts and Exponent Detect (Block Floating Point Adjustment)

ADDRESSING

- Direct Addressing
  - » ax0 = dm (0x1FFF) ;
  - » my0 = pm (0x07F0) ;
  - » dm (0x0400) = ar ;
- Indirect Addressing
- Circular Addressing (Filtering)
- Bit-Reversed Addressing (FFT)
  - » ena bit_rev ; { Enables bit-reversed addressing }
  - » dis bit_rev ; { Disables bit-reversed addressing }

All instructions end with semi-colon.

Curly Braces Enclose Comments.

INDIRECT ADDRESSING

- Uses Index Registers (i0 - i7) and Modify Registers (m0 - m7)
  - » Index register points to accessed memory location
  - » Modify register holds the increment/decrement value to be used
- Example
  - » sr1 = pm (i4, m4) ;
  - » dm (i0, m1) = ar ;

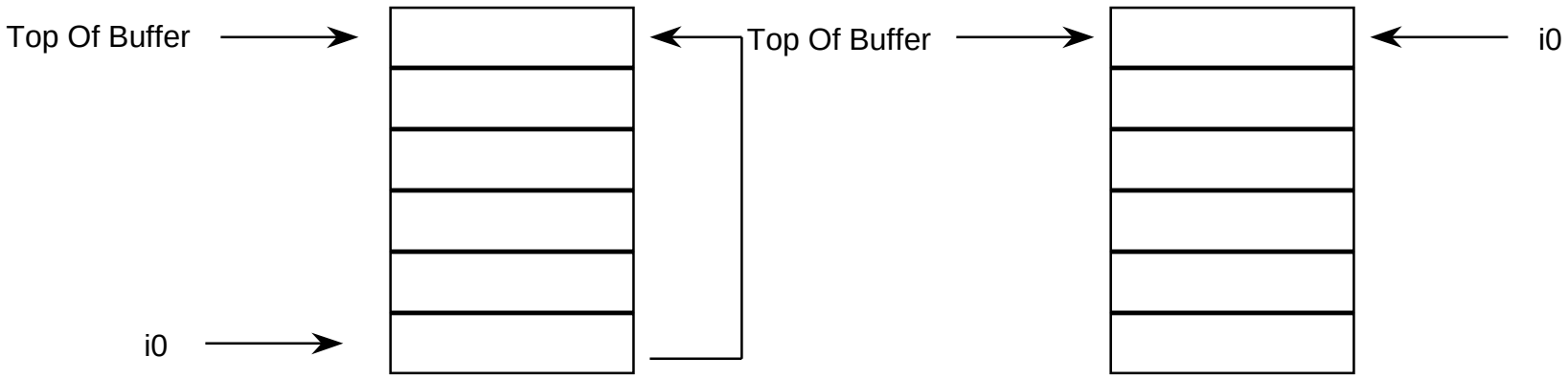
Equivalent to:
sr1 = pm (i4), i4 = i4 + m4 ;
Both Read and Modify are Done in a Single Cycle.

MORE ON INDEX REG

- Modify Only Operation also Available
  - » modify (i2, m0) ; { Equivalent to i2 = i2 + m0 }
- 0-3 form one group and 4-7 form another
  - » can not mix I and M registers from different groups
    - ar = dm (i0, m5) ; { Invalid }
- DM can be accessed using either group
- PM using second group only
    - ar = dm (i0, m3) ; { Valid }
    - ar = pm (i0, m3) ; { Invalid }

CIRCULAR ADDRESSING

- Creates Circular Buffers
- L Register Holds the Length of Circular Buffer Pointed by Corresponding I Register



L0 = 6
M0 = 1

Result of MODIFY(i0,m0)

BASE ADDRESS

- Base Address of a Circular Buffer of Length L is 2^n or a multiple of 2^n, where n satisfies

$$2^{n-1} < L \leq 2^n$$

- Buffer of Length 8 can have Following Addresses
  - 0x0008, 0x0010, 0x0018

Linker automatically places circular buffer at a valid address.

DECLARATION

- Variable
  - » .var/dm/ram Gain_ ;
- Non-Circular Buffer
  - » .var/pm/ram FilterCoefficients_[128] ;
- Circular Buffer
  - » .var/dm/ram/circ FilterInput_[128] ;
- Making a variable global
  - » .global Gain_ ;

If declared global, the variable can be accessed in other files. To access Gain_ from a different file, declare it as an EXTERNAL variable in the new file:
.external Gain_ ;

ALU

- 16 Bits Wide with Two 16-Bit Input Ports (X and Y) and One Output Port (R)
  - » Source for X Input Port
    - AX0, AX1, AR, MR0, MR1, MR2, SR0, SR1
  - » Source for Y Input Port
    - AY0, AY1, AF
  - » Destination for R Output Port
    - AR, AF

ALU FUNCTIONS

- R = X + Y
- R = X + Y + CI
- R = X - Y
- R = X - Y + CI - 1
- R = Y - X
- R = Y - X + CI - 1
- R = - X
- R = - Y
- R = Y + 1

Add with Carry

Subtract with Borrow

- R = Y - 1
- R = PASS X
- R = PASS Y
- R = ABS X
- R = X AND Y
- R = X OR Y
- R = X XOR Y
- R = NOT X
- R = NOT Y

These two instructions are useful in testing signs

OTHER ALU-FEATURES

- Saturation Arithmetic
  - » Result set to maximum positive (negative) value in case of overflow (underflow)
  - » Supported only by AR Register (If AF is the destination, wrap-around occurs always)
- All Operations affect ALU Flags
  - » AZ, AN, AV, AC
- Division
  - » ≈ Successive Subtractions (Time Consuming)

NUMBER FORMAT

- 16 Bit Processor
  - » Ideal for 16 Bit Arithmetic (a.k.a. Single Precision Arithmetic)
    - Reserve 1 Bit for Sign (Two's Complement)
      - – Can Store Integers in the range $[-2^{15}, 2^{15})$
    - Single Cycle Add/Sub/Mult
  - » Extended Precision Arithmetic Possible
    - Better Dynamic Range
    - Add/Sub/Mult Complex

“INTEGER PROBLEM”

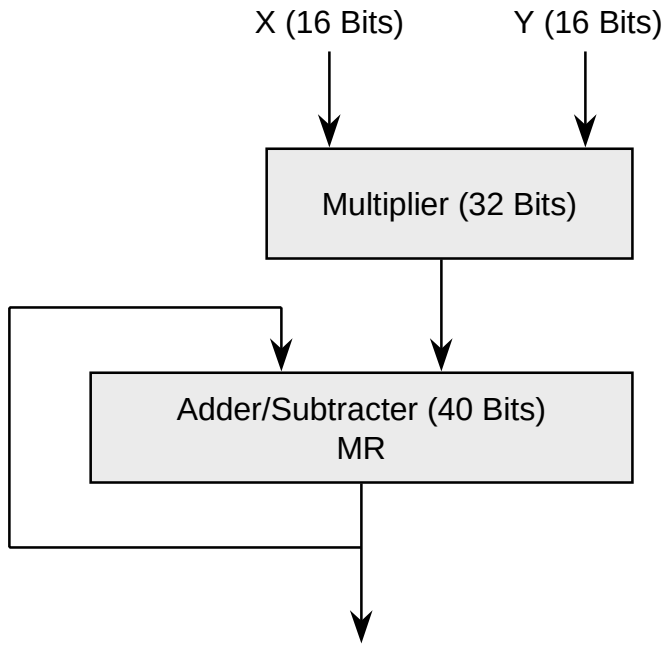
- Imagine a 2 digit “Decimal Processor”
  - » Can Store Integers in the Range [0, 99]
  - » Suppose we want to find 87 * 93
    - 87 * 93 = 8091 (4 digit Result)
  - » How to Store the Result?
    - Store 8091 in Two-word Format
    - Store the 2 MSDs only and “remember” the Exponent (Acceptable Truncation Error)
      - – Getting into Complex Extended Precision Arithmetic

SOLUTION

- Use 2 Digits to Store Fractions
  - » Can Store .00, .01, .02, … , .98, .99
    - .87 * .93 = .8091
  - » Truncate the Result and Store 2 MSDs
    - Acceptable Error
    - Can Continue Using Single Precision Arithmetic
- Better to Work with Fractions than Integers
  - » All Variables need to be Normalized

MAC

- Two 16-Bit Inputs
  - » 32-Bit Product Output
  - » Product fed to a 40-Bit adder/subtracter MR
  - » MR = MR2 MR1 MR0
    - MR0, MR1 - 16 bits
    - MR2 - 8 bits wide
- X - Inputs
  - » MX0, MX1, AR, MR0, MR1, MR2, SR0, SR1
- Y-Inputs
  - » MY0, MY1, MF



- Destination
  - »MR (MR0, MR1, MR2), MF

MAC FUNCTIONS

- Functions
  - » R = X * Y
  - » R = MR + X * Y
  - » R = MR - X * Y
- Multiplication Mode
  - » Integer Mode (16.0 * 16.0 = 32.0)
  - » Fractional Mode (1.15 * 1.15 = 1.31)
    - Multiplier result shifted to the left by 1

Check the multiplication mode if the result is off by a factor of 2

OTHER MAC-FEATURES

- Rounding Modes
  - » Biased Rounding
  - » Unbiased Rounding

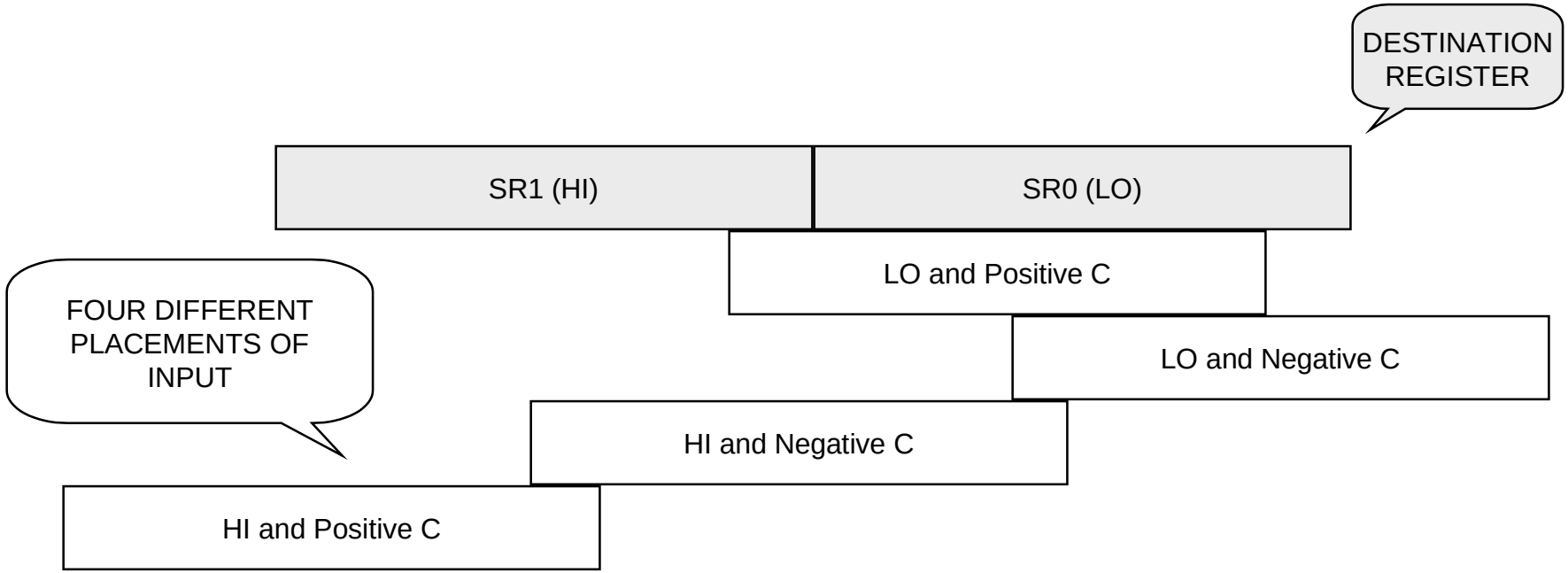
These two differ only when MR0=0x8000

MR Value Before RND	Biased RND Result	Unbiased RND Result
00-0000-8000	00-0001-8000	00-0000-8000
00-0001-8000	00-0002-8000	00-0002-8000
00-0000-8001	00-0001-8001	00-0001-8001
00-0001-8001	00-0002-8001	00-0002-8001
00-0000-7FFF	00-0000-7FFF	00-0000-7FFF
00-0001-7FFF	00-0001-7FFF	00-0001-7FFF

SHIFTER

- Accepts a 16-Bit Input and Can Place it Anywhere in the 32-Bit Output Register
  - » 49 Possible Placements
  - » Placement Determined by a Control Code and HI/LO Signal
- Shifter Input Registers
  - » SI, AR, MR0, MR1, MR2, SR0, SR1
- Destination for Shifter Output
  - » SR (SR0, SR1)

SHIFT OPERATION



SHIFTER FUNCTIONS

- Arithmetic Shift (ASHIFT)
  - » MSB's filled with Sign for -ve C
- Logical Shift (LSHIFT)
  - » MSB's filled with Zeros for -ve C
- Block Exponent Adjust (EXPADJ)
  - » When performed on a series of numbers, derives the effective exponent of the number largest in magnitude

SHIFTER REGISTERS

- SI Register (Shifter Input)
  - » One of the Possible Inputs to Shifter
- SE Register (Shifter Exponent) 8-Bits Wide
  - » Holds the Exponent for Shift Operations
- SB Register (Shifter Block) 5-Bits Wide
  - » Holds the Block Exponent Value
    - Used in EXPADJ Instruction

PROGRAM CONTROL

- Instructions Executed Sequentially
- Execution Sequence Modified Using
  - » DO UNTIL (Zero-Overhead Looping)
  - » JUMP
  - » CALL
  - » RTS (Return From Subroutine)
  - » RTI (Return From Interrupt)
  - » IDLE

PROGRAM COUNTER (PC)

- PC 14 Bits Wide
  - » Holds the Address of Currently Executing Instruction
- Contents of PC (automatically) Pushed into PC Stack when
  - » CALL Instruction Executed
  - » DO UNTIL Instruction Executed
  - » Interrupt Occurs
- Popping Also Occurs Automatically

CNTR (COUNTER)

- 14-Bit Register
- Useful In Looping
  - » Example:
    - DO [*Address*] UNTIL CE (Counter Expires)
    - Load N into CNTR, if N passes needed
- Nesting of Loops Possible

```
CNTR = 64 ;
DO OuterLoop UNTIL CE ;
                ---------- ;
                ---------- ;
                CNTR = 128 ;
                DO InnerLoop UNTIL CE ;
                                ---------- ;
                                ---------- ;
                InnerLoop:      ---------- ;
                ---------- ;
OuterLoop :     ---------- ;
```

Outer-loop Counter Value Lost when Inner-Loop Count Loaded.
So, the old count is automatically pushed into the LOOP Stack before loading new value

DO UNTIL LOOPS

Last Instruction Address and Condition Stored in Loop Comparator Stack and Start Address in PC Stack

- During each cycle
  - » Check if last instruction of loop
    - If yes, jump to the start of the loop conditionally
      - – Termination Conditions Available
        - EQ, NE, LT, GT, LE, GE, AC, NOT AC, AV, MV, NOT AV, NOT MV, CE, FOREVER
- Example
  - » DO LastInst UNTIL FOREVER
- Avoid Following as Last Instruction
  - » JUMP, CALL, RETURN, IDLE

MORE ON STACKS

- PC Stack Depth 16
- Status Stack Depth 16
- CNTR Stack Depth 4
- Loop Comparator Stack Depth 4
  - » Max. number of loops that can be nested = 4
- Manual Popping of Stacks Possible
  - » Useful in immature exit from loops

If your program behaves strangely, check if there is any stack overflow.

INTERRUPTS

- 12 Interrupts Available
- Provision to Mask Interrupts (IMASK Reg)
  - » Power-down is only non-maskable interrupt
- Interrupt Transfers Control to Appropriate Interrupt Vector Address
- RTI - Last Instruction of Interrupt Service Routine (ISR)

INTERRUPT VECTORS

- Listed in Descending Order of Priority

Interrupt Source	Interrupt Vector Address
RESET (Startup)	0x0000
Powerdown	0x002C
IRQ2	0x0004
IRQL1	0x0008
IRQL0	0x000C
SPORT0 Transmit	0x0010
SPORT0 Receive	0x0014
IRQE	0x0018
BDMA Interrupt	0x001C
SPORT1 Transmit or IRQ1	0x0020
SPORT1 Receive or TRQ0	0x0024
Timer	0x0028

Note that the interrupt vector locations are four apart. If the ISR has less than 5 instructions, use those 4 locations. Else, use a JUMP.

MORE ON INTERRUPTS

- When Interrupt Occurs
  - » Current PC pushed into PC stack
  - » Current values of ASTAT, MSTAT, and IMASK Registers pushed into status stack
  - » PC set to appropriate interrupt vector address
- Interrupts Configurable (ICNTL Reg)
  - » Nesting
  - » Edge- or Level-Sensitive
- Force/Clear Interrupts Using IFC Register

Avoid Nesting of Interrupts.

SERIAL PORT

- Two Serial Ports (SPORT0 and SPORT1)
- Double Buffered

What will be the problem, if it is not double buffered?

- Transmit and Receive Registers
  - » TX0, RX0, TX1, RX1
- Interrupt on completion of TX / RX
- Multichannel Capability
- Fully Configurable

SPORT CONFIGURATION

- Done Using Memory Mapped Registers
- Can Configure the Following:
  - » Word Length (1 - 16)
  - » Clock Source (Internal or External)
    - If internal, programmable frequency
  - » Framing (Internal or External Synchronization)
  - » Autobuffering

TIMER

- Periodic Interrupt with Programmable Period
- Based on 3 Registers
  - » TCOUNT (16 Bits)
  - » TPERIOD (16 Bits)
  - » TSCALE (8 Bits)
- To Enable / Disable Timer Use
  - » ena Timer ;
  - » dis Timer ;

TIMER OPERATION

- When Enabled
  - » TSCALE Decremented Once Every Clock Cycle
  - » When TSCALE reaches zero
    - TCOUNT Decremented by One
    - TSCALE Reset to Original Value
  - » When TCOUNT reaches zero
    - TIMER Interrupt Generated
    - TCOUNT Reinitialized to Contents of TPERIOD