Section 3

Number Formats and Registers

Binary Number Formatting

Binary Notation in DSP's

The ADSP-BF533 is a fixed point processor that performs operations using a two's complement binary notation. Therefore, to efficiently program a DSP it is important to understand the following concepts:

1) **Signed vs Unsigned formats**

2) **Fractional vs Integer formats**

3) **Ranges of Fractional Numbers**

While the Blackfin Processor does process 8, 16 and 32 bit data, only 16-bit examples are shown in this section for brevity.

Binary - Hexadecimal - Decimal Number Conversion Table

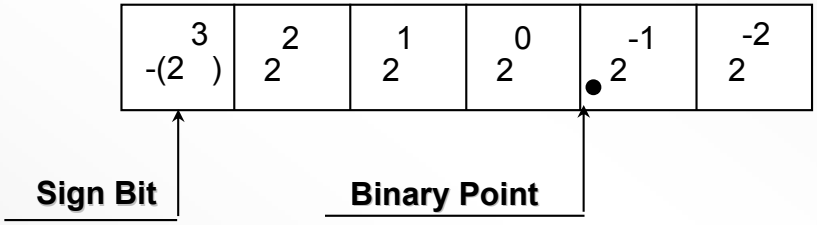
Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Example of Data Formats

Binary Bit Pattern		Example System
Unsigned	0x0000	0V (- FULL SCALE)
	0xFFFF	5V (+ FULL SCALE)
Signed	0x8000	-5V (- FULL SCALE)
	0x0000	0V
	0x7FFF	5V (+ FULL SCALE)

2's Complement Representation

In 2's complement representation, the bit weight of the sign bit (MSB) of a number is seen as -(2)^(M-1) where M is the number of bits left of the binary point. For a 4.2 number, the sign scale is -(2^3).



Example: **0101.01 = 0 * (-8) + 1 * (4) + 0 * (2) + 1 * (1) + 0 * (1/2) + 1 * (1/4)**
= 5.25

1101.01 = 1 * (-8) + 1 * (4) + 0 * (2) + 1 * (1) + 0 * (1/2) + 1 * (1/4)
= - 8 + 5.25
= - 2.75

2's Complement Representation

Changing the sign of a 2‘s Complement Number

-X = NOT(X) + 1 LSB (invert all the bits and add an LSB)

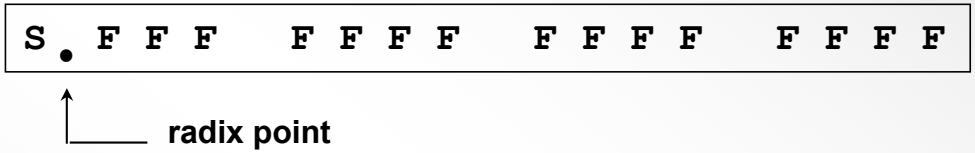
Example:

-5.25 = -(b#0101.01)
= NOT(b#0101.01) + b#0000.01
= b#1010.10 + b#0000.01
= b#1010.11

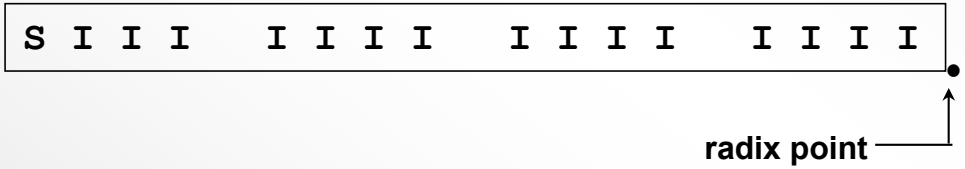
1010.11 = 1 * (-8) + 0 * (4) + 1 * (2) + 0 * (1) + 1 * (1/2) + 1 * (1/4)
= - 8 + 2.75
= - 5.25

Fractional versus Integer Notation

- Fractional format is 1.15 notation



- Integer format is 16.0 notation



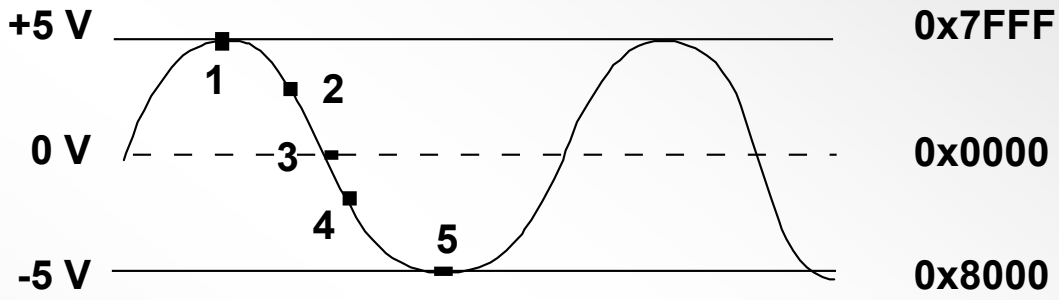
DSP is optimized for fractional notation

DSP supports integer notation

Ranges for 16 bit Formats

FORMAT		Largest Positive Value (0x7FFF) In Decimal	Largest Negative Value (0x8000) In Decimal	Value of 1 LSB (0x0001) In Decimal
1.15	Fractional	0.999969482421875	–1.0	0.000030517578125
2.14		1.999938964843750	–2.0	0.000061035156250
3.13		3.999877929687500	–4.0	0.000122070312500
4.12		7.999755859375000	–8.0	0.000244140625000
5.11		15.999511718750000	–16.0	0.000488281250000
6.10		31.999023437500000	–32.0	0.000976562500000
7.9		63.998046875000000	–64.0	0.001953125000000
8.8		127.996093750000000	–128.0	0.003906250000000
9.7		255.992187500000000	–256.0	0.007812500000000
10.6		511.984375000000000	–512.0	0.015625000000000
11.5		1023.968750000000000	–1024.0	0.031250000000000
12.4		2047.937500000000000	–2048.0	0.062500000000000
13.3		4095.875000000000000	–4096.0	0.125000000000000
14.2		8191.750000000000000	–8192.0	0.250000000000000
15.1		16383.500000000000000	–16384.0	0.500000000000000
16.0	Integer	32767.000000000000000	–32768.0	1.000000000000000

Format Example



		FORMAT 16.0	FORMAT 1.15
1)	0x7FFF	= 32767 -> 5 V	0.999969482... -> 5 V
2)	0x3FFF	= 16383 -> 2.5 V	0.499969482... -> 2.5 V
3)	0x0000	= 0 -> 0 V	0.0000000... -> 0 V
4)	0xCCCD	= -13107 -> -2.0 V	-0.399993986... -> -2.0 V
5)	0x8000	= -32768 -> -5.0 V	-1.0000000.... -> -5.0 V

Registers and Register File

Accessing Registers

- **Blackfin Processors are register-intensive devices**
  - **All computations are performed on data contained in registers**
  - **All peripherals are setup using registers**
  - **Memory is accessed using pointers in address registers**

- **There are two ways to access registers on the ADSP-BF533**
  - Directly by name
  - Memory-mapped registers (MMRs)

ADSP-BF533 Core Registers

- **Core registers accessed by name**
  - **Data Registers:** **R0-R7**
  - **Accumulator Registers:** **A0, A1**
  - **Pointer Registers:** **P0-P5, FP, SP,USP**
  - **DAG Registers:** **I0-I3, M0-M3, B0-B3, L0-L3**
  - **Cycle Counters:** **Cycles, cycles2**
  - **Program Sequencer:** **SEQSTAT**
  - **System Configuration Register:** **SYSCFG**
  - **Loop Registers:** **LT[1:0], LB[1:0], LC[1:0]**
  - **Interrupt Return Registers:** **RETI, RETX, RETN, RETE**

Example:

R0 = SYSCFG; // Load data register with contents of SYSCFG register

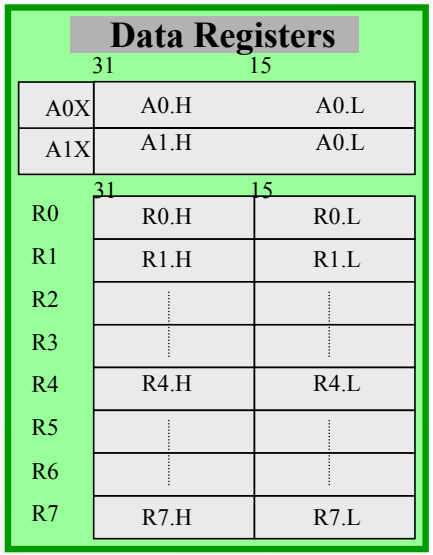
Core Registers

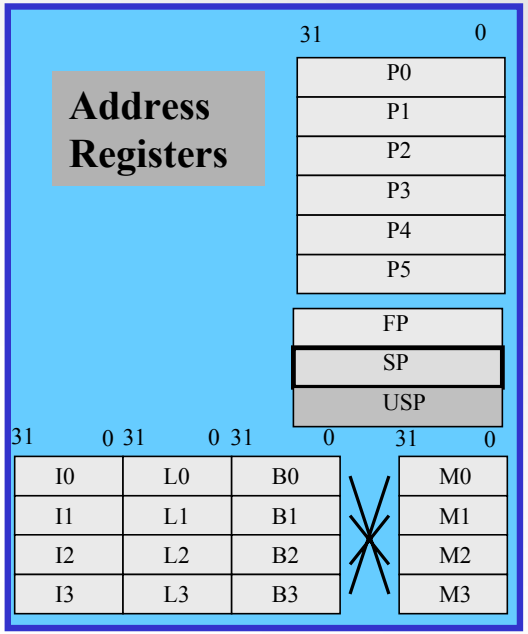
Data Registers:

R0-R7 **are referred to as "dreg"**

_lo refers to .L and

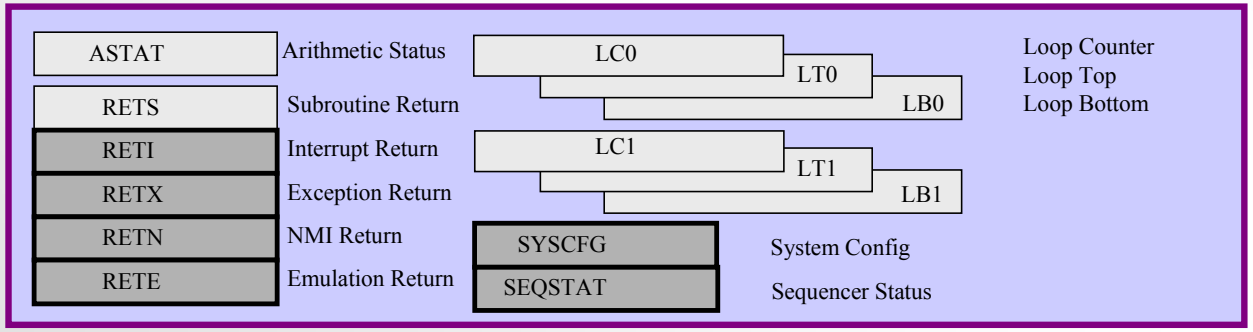
_hi refers to .H





Pointer Registers: ***P0-P5*** **are referred to as "preg"**

Index Registers: ***I0-I3*** **are referred to as "ireg"**



Shaded registers only accessible in Supervisor mode

Memory Mapped Registers (MMRs)

- **A majority of registers are memory mapped and must be accessed indirectly**
  - **Core MMRs are used to configure the core registers**
    - **They are listed in Appendix A of the HRM**
    - **All Core MMRs must be accessed with 32-bit reads or writes**
  - **System MMRs are used to configure all other peripherals**
    - **They are listed in Appendix B of the HRM**
    - **Some System MMRs must be accessed with 32-bit reads or writes and others with 16-bit reads or writes (See the HRM for details)**
- **The addresses of the core and system MMRs are defined in the defBF533.h, defBF532.h, defBF531.h and defLPblkfin.h header files**
- **MMRs can only be accessed in Supervisor mode**

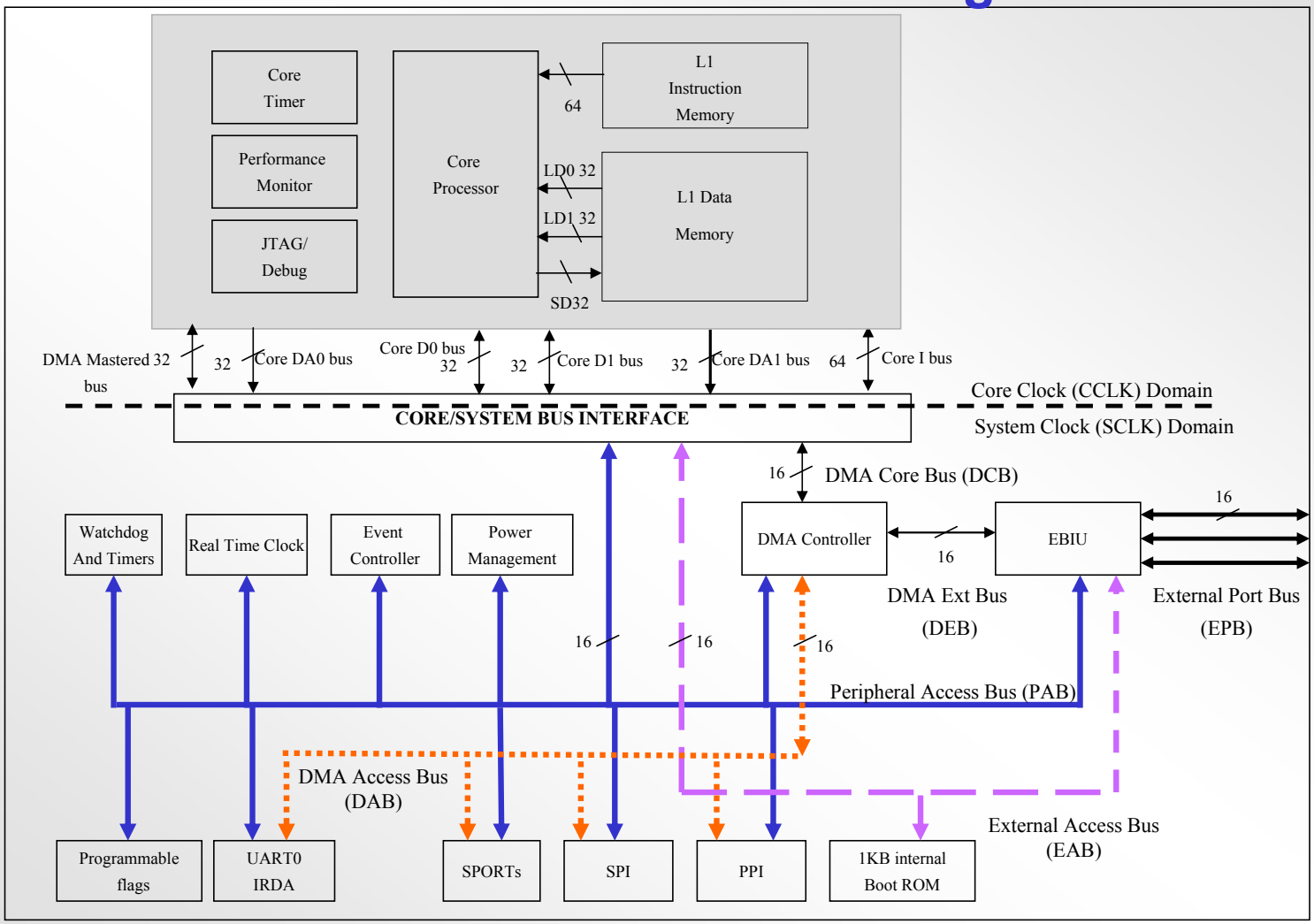
Example:

```
P0.H = hi(MMR_NAME);  // load upper 16-bits of MMR address to address register
P0.L = lo(MMR_NAME);  // load lower 16-bits of MMR address to address register
R0 = w[P0] (z);       // load data register with contents of MMR
```

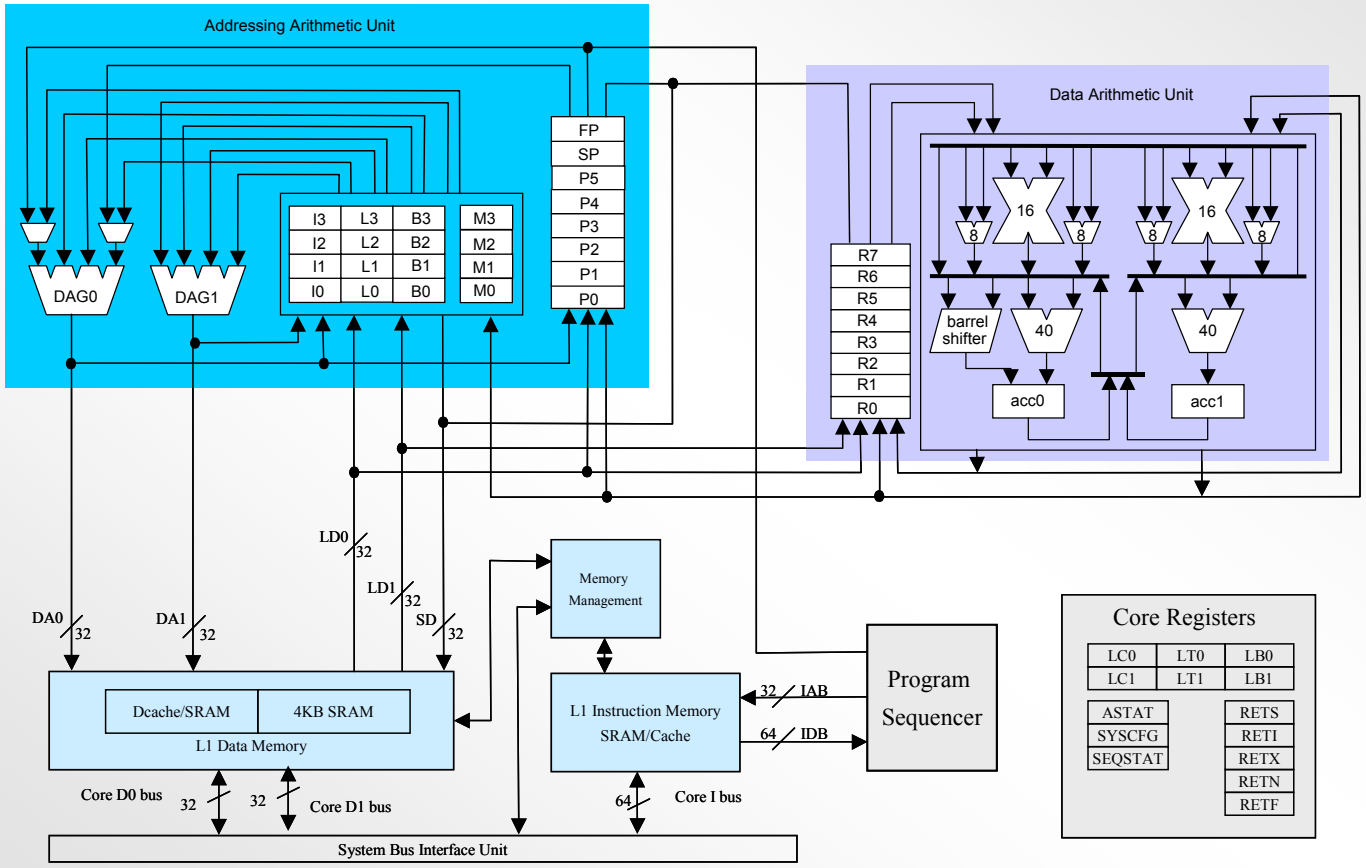
ADSP-BF533 Register Groups

- Core Memory-mapped Registers
  - L1 Data Memory Registers
  - L1 Instruction Memory Registers
  - Core Timer Registers
  - Core Interrupt Controller Registers
  - Debug Registers
- System Memory-mapped Registers
  - Watchdog Timer Registers
  - Timer Registers
  - RTC Registers
  - PPI Registers
  - EBIU Registers
  - GPIO Registers (Programmable flags)
  - UART Registers
  - SPORT Registers
  - SPI Registers
  - DMA Registers
  - PLL Registers

ADSP-BF533 Block Diagram

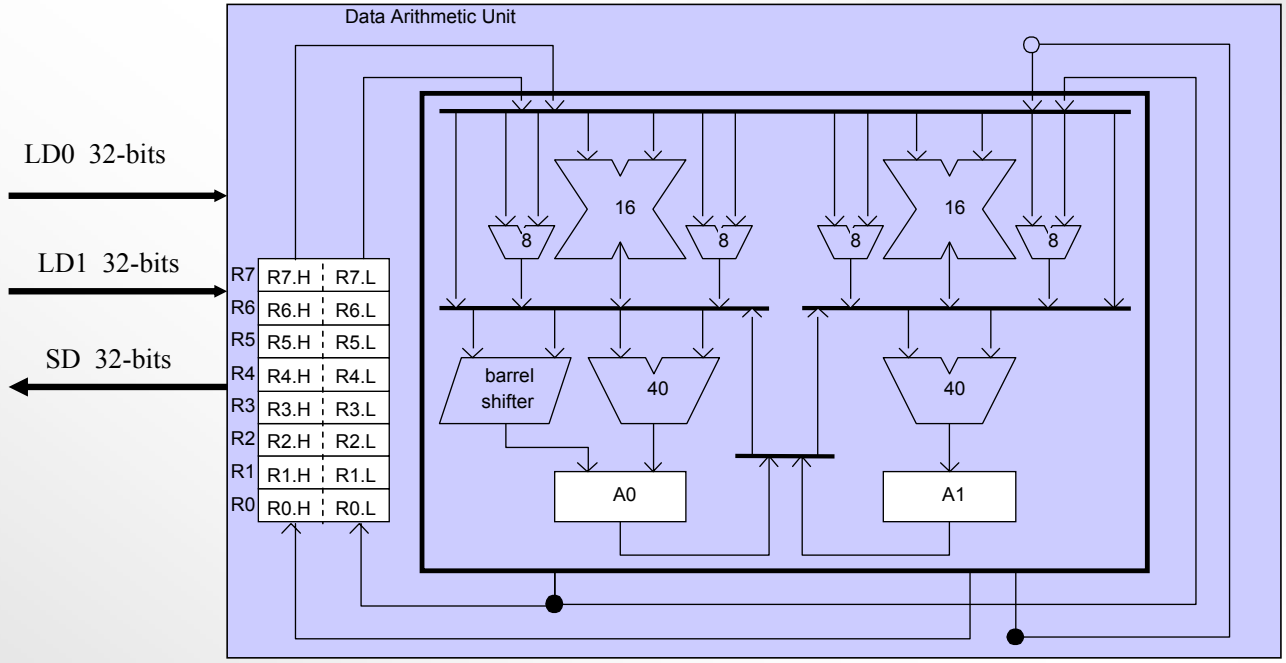


ADSP-BF533 Core



Register File

Register File



Debugger Introduction and Register File Exercises

LABs 1 & 2