Floating Point Representation – Chap 3

Prepared By Fairoz Nower Khan

Floating Point

§3.5 Floating Point

- Representation for non-integral numbers
  - Including very small and very large numbers
- Like scientific notation
  - -2.34×10^{56}
  - $+0.002 \times 10^{-4}$
  - $+987.02 \times 10^{9}$

normalized

not normalized

- In binary
  - $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
- Types **float** and **double** in C

Floating Point Standard

- Defined by IEEE Std 754-1985
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Now almost universally adopted
- Two representations
  - Single precision (32-bit)
  - Double precision (64-bit)

Normalized Number

- Only One and Non-Zero number before .(decimal point)

$5.64 \times 10^{33} \rightarrow$ Normalized

$109.64 \times 10^{33} \rightarrow 1.0964 \times 10^{33+2}$

1.0964×10^{35}

The number of times we left shift the (.), will be added with the exponent

$0.675 \rightarrow 6.75 \times 10^{-1}$

The number of times we right shift the (.), will be subtracted from the exponent

- Only One and Non-Zero number before .(binary point)

$1011.1101 \times 2^{33} \rightarrow 1.011101 \times 2^{33+3} = 1.011101 \times 2^{36}$

In Binary the Base is 2

$0.0111101 \times 2^{-5} \rightarrow 1.11101 \times 2^{-5-2} = 1.11101 \times 2^{-7}$

Decimal to Floating Point Conversion

- Step 1: Convert the Decimal Number into Binary Number
- Step 2: Normalize the Binary Number
- Step 3: Find out the Biased Exponent
- Step 4: Find out Sign bit and Fraction
- Step 5: Write the Sign bit, Biased Exponent and Fraction in IEEE-754 Floating Point Representation

IEEE Floating-Point Format

	single: 8 bits double: 11 bits	single: 23 bits double: 52 bits
S	Exponent	Fraction

$$x = (-1)^{S} \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Normalize significand: 1.0 ≤ |significand| < 2.0
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1203

Single Precision (32 bit)

Sign bit	Exponent (Biased)	Fraction/ Mantissa
1 bit	8 bit	23 bit

If n = bit length of Exponent Field,
Bias = $2^{(n-1)}-1$

Sign Bit: (0 ⇒ Positive, 1 ⇒ Negative)

Exponent:

8 bit unsigned binary Range= 0 to $2^8 - 1$ = 0 to 255

Exponents 00000000 and 11111111 reserved, So the Range for Biased Exponent Becomes = **1 - 254**

For Exponent being 8 bit, Bias = $2^{(8-1)} - 1 = 127$

For Single Precision

Biased Exponent = Actual Exponent of the Binary number + Bias (127)

Range for Exponent = $2^{-126} - 2^{127}$

1.11101×2^{35}

Biased Exponent = 35 + 127
= 162 = 10100010

1.11101×2^{-8}

Biased Exponent = -8 + 127
= 119
=01110111

Example

- **Convert 50.6749 to 32 bit IEEE-754 Floating Point Representation**
- **Step -1 Convert the Decimal Number To Binary Number**

50.67490

Binary of 50 = 110010

Binary of .6749 = 1010110011

Binary of 50.6749 = 110010.1010110011

- **Step -2 Normalize the Binary Number**

Binary of 50.6749 = 110010.1010110011 x 2^0

Normalized Binary Number =

$$1.\underbrace{100101010110011}_{\text{Fraction}} \times 2^5$$

Binary of .6749

= .6749 x 2 = 1.3498 =1

= .3498 x 2 = 0.6996 =0

= .6996 x 2 = 1.3992 =1

= .3992 x 2 = 0.7984 =0

= .7984 x 2 = 1.5968 =1

= .5968 x 2 = 1.1936 =1

. **= 0**

. **= 0**

. **= 1**

. **= 1**

- **Convert 50.6749 to 32 bit IEEE-754 Floating Point Representation**

Normalized Binary Number = 1.10010101011001 x 2^5

- **Step -3 Find Out The Biased Exponent**

Exponent = 5

Biased Exponent = 5+127

=132

=10000100

- **Step -3 Find Out Sign Bit and Fraction**

Sign Bit = 0

Fraction= 10010101011001 00000000

- **Step -4 IEEE-754 Floating Point Representation**

(Biased)

Sign bit	Exponent	Fraction/ Mantissa
1 bit	8 bit	23 bit
0	10000100	10010101011001100000000

IEEE-754 Floating Point Representation of 50.6749

01000010010010101011001100000000

= 0100 0010 0100 1010 1011 0011 0000 0000

= **0x424AB300**

Double Precision (64 bit)

	(Biased)	
Sign bit	Exponent	Fraction/ Mantissa
1 bit	11 bit	52 bit

If n = bit length of Exponent Field,
Bias = $2^{(n-1)}-1$

Sign Bit: (0 ⇒ Positive, 1 ⇒ Negative)

Exponent:

11 bit unsigned binary Range= 0 to $2^{11} - 1$

Exponents 00000000 and 11111111 reserved, So the Range for Biased Exponent Becomes = **1 - 2046**

For Exponent being 11 bit, Bias = $2^{(11-1)} - 1 = 1023$

For Double Precision

Biased Exponent = Actual Exponent of the Binary number + Bias (1023)

- **Convert -0.232 to 12 bit IEEE-754 Floating Point Representation, Where Exponent is 4 bit**

	(Biased)	
Sign bit	Exponent	Fraction/ Mantissa
1 bit	4 bit	7 bit

Binary of 0.232 = 0.00111011

Normalized Binary of 0.232 = 1.11011 x 2^{-3}

Exponent:

For Exponent being 4 bit, Bias = $2^{(4-1)}-1 = 7$

Exponent = -3

Biased Exponent = -3+7 =4 = 0100

If n = bit length of Exponent Field,
Bias = $2^{(n-1)}-1$

Sign Bit and Fraction:

Sign Bit = 1

Fraction= 1101100

Floating Point Representation

1 0100 1101100
1010 0110 1100 = 0xA6C

- **Convert 1.232x10^2 to 15 bit IEEE-754 Floating Point Representation, Where Exponent is 5 bit**

Sign bit	Exponent (Biased)	Fraction/ Mantissa
1 bit	5 bit	9 bit

Binary of 1.232x10^2 = 123.2 = 1111011.0011

Normalized Binary of 123.2 = 1.1110110011 x 2^6

Exponent:

For Exponent being 3 bit, Bias = $2^{(5-1)}-1 = 15$

Exponent = 6

Biased Exponent = 6+15 = 21 = 10101

If n = bit length of Exponent Field,
Bias = $2^{(n-1)}-1$

Sign Bit and Fraction:

Sign Bit = 0

Fraction= 111011001

Floating Point Representation

0 10101 111011001
0 10101 1110110010 = 57B2

Hexadecimal

- Base 16
  - Compact representation of bit strings
  - 4 bits per hex digit

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

- Example: eca8 6420
  - 1110 1100 1010 1000 0110 0100 0010 0000

Floating Point (Single Precision) to Decimal

- 0xF2400120

Sign bit	Exponent (Biased)	Fraction/ Mantissa
1 bit	8 bit	23 bit

Step 1: Hexadecimal to Binary

1111 0010 0100 0000 0000 0010 0100 0000

Step 2: Set the Binary Number as Format

1 11100100 10000000000001001000000

Step 3: Find Out Exponent and Fraction

Biased Exponent =11100100

Biased Exponent (Decimal)=228

Exponent (Decimal)= 228 - 127

=101

For Exponent being 8 bit, Bias = $2^{(8-1)}-1 = 127$

Fraction/ Mantissa = 0.10000000000001001000000

$= 2^{-1}+2^{-15}+2^{-18}$

= 0.5000343323

$$\text{Decimal Value} = (-1)^{\text{SignBit}} \times (1+\text{Fraction}) \times 2^{(\text{Exponent})}$$

$(-1)^1 \times (1+0.5000343323) \times 2^{101}$

$= -1.5000343223 \times 2^{101}$

$= -3.803038843 \times 10^{30}$

Upto 5 decimal point with Rounding= -3.803034×10^{30}

Upto 5 decimal point without Rounding= -3.803033×10^{30}

Practice

Consider the value 63.7813

a) Let’s assume you have a 21-bit register having 6-bit for exponent. Now convert this value using IEEE floating-point representation. Also, convert this into hexadecimal form.

Ans: 49FC8

b) Let’s assume you have a 12-bit register having 4-bit for exponent. Now convert this value using IEEE floating-point representation. Also, convert this into hexadecimal form

Ans: 67F

Floating Point Arithmetic

Floating Point Addition/ Subtraction

- 35.23142 + 0.00053

X = 35.23142

X (Binary)
= 100011.0011101111

X (Binary Normalized) =
$1.000110011101111 \text{x } 2^5$

Y = 0.00053

Y (Binary)
= 0.000000000010001011

Y (Binary Normalized) =
$1.0001011 \text{ x } 2^{-11}$

Y (Binary) =
$0.00000000000000010001011 \text{x } 2^5$

Rule: Match the Lower Exponent with the Higher Exponent

X + Y = $(1.000110011101111 \text{x } 2^{5)}\ + (0.00000000000000010001011 \text{x } 2^5\ )$

$= (1.000110011101111 + 0.00000000000000010001011) \text{ x } 2^5$

$= 1.00011001110111110001011 \text{ x } 2^5$

= 100011.001110111110001011

= 35.2342224121 **(Decimal)**

Floating Point Multiplication

- 5.234 x (-0.003)

X = 5.234

X (Binary)
= 101.0011101111

X (Binary Normalized) =
$1.010011101111 \times 2^{2}$

Y = 0.003

Y (Binary)
= 0.0000000011000100101

Y (Binary Normalized) =
$1.1000100101 \times 2^{-9}$

X x Y = $-(1.010011101111 \times 2^{2}) \times (1.1000100101 \times 2^{-9})$

$= -(1.010011101111 \times 1.1000100101) \times 2^{(2+(-9))}$

$= -(1.010011101111 \times 1.1000100101) \times 2^{(-7)}$

$= -10.000000101 \times 2^{(-7)}$

$= -0.0000010000000101 \times 2^{(0)}$

= - 0.0157012939 (Decimal)

Floating Point Arithmetic

- 51500000 – BA10A000

	(Biased)	
Sign bit	Exponent	Fraction/ Mantissa
1 bit	8 bit	23 bit

X = 51500000

X (Binary)

= 0101 0001 0101 0000 0000 0000 0000 0000 [32 bit]

= 0 10100010 10100000000000000000000

Find Out Exponent and Fraction

Biased Exponent = 10100010

Biased Exponent (Decimal)= 162 For Exponent being 8 bit, Bias = $2^{(8-1)}$ -1 = 127

Exponent (Decimal)= 162 - 127

= 35

Fraction/ Mantissa = 0. 10100000000000000000000

X (Binary Normalized) = 1.Franction x $2^{(Exponent)}$

X (Binary Normalized) = 1. 10100000000000000000000 x 2^{35} *Sign bit = 0 (Positive)

Floating Point Arithmetic

- 51500000 – BA10A000

	(Biased)	
Sign bit	Exponent	Fraction/ Mantissa
1 bit	8 bit	23 bit

Y = BA10A000

Y (Binary)

= 1011 1010 0001 0000 1010 0000 0000 0000 [32 bit]

= 1 01110100 00100001010000000000000

Find Out Exponent and Fraction

Biased Exponent = 01110100

Biased Exponent (Decimal)= 116 For Exponent being 8 bit, Bias = $2^{(8-1)} - 1 = 127$

Exponent (Decimal)= 116 - 127

= -11

Fraction/ Mantissa = 0. 00100001010000000000000

Y (Binary Normalized) = 1.Franction x $2^{(Exponent)}$

Y (Binary Normalized) = - 1.00100001010000000000000 x 2^{-11} *Sign bit = 1 (Negative)

Floating Point Addition/ Subtraction

- 51500000 – BA10A000

Rule: Match the Lower Exponent with the Higher Exponent

X = 51500000

X (Binary Normalized) =
$1.\ 10100000000000000000000 \times 2^{35}$

Y = BA10A000

Y (Binary Normalized) =
$\mathbf{-}\ 1.00100001010000000000000 \times 2^{-11}$

Y (Binary Normalized) =
$\mathbf{-}\ 0.[45\ 0s..]\ 100100001010000000000000 \times 2^{35}$

X – (-Y) =
X + Y =

$= (1.\ 10100000000000000000000 \times 2^{35}) + (0.[45\ 0s..]\ 100100001010000000000000 \times 2^{35})$

$= (1.\ 10100000000000000000000 + 0.[45\ 0s..]\ 100100001010000000000000) \times 2^{35}$

$= 1.101[42\ 0s..]\ 100100001010000000000000 \times 2^{35}$

Floating Point Arithmetic

- 7ACD0000 + 5BCA0000

	(Biased)	
Sign bit	Exponent	Fraction/ Mantissa
1 bit	8 bit	23 bit

X = 7ACD0000

X (Binary)

= 0111 1010 1100 1101 0000 0000 0000 0000 [32 bit]

= 0 11110101 10011010000000000000000

Find Out Exponent and Fraction

Biased Exponent = 11110101

Biased Exponent (Decimal)= 245 For Exponent being 8 bit, Bias = $2^{(8-1)}-1 = 127$

Exponent (Decimal)= 245 - 127

= 118

Fraction/ Mantissa = 0.10011010000000000000000

X (Binary Normalized) = 1.Franction x $2^{(Exponent)}$

X (Binary Normalized) = 1. 10011010000000000000000 x 2^{118} *Sign bit = 0 (Positive)

Floating Point Arithmetic

- 7ACD0000 + 5BCA0000

	(Biased)	
Sign bit	Exponent	Fraction/ Mantissa
1 bit	8 bit	23 bit

Y =5BCA0000

Y (Binary)

= 0101 1011 1100 1010 0000 0000 0000 0000 [32 bit]

= 0 10110111 10010100000000000000000

Find Out Exponent and Fraction

Biased Exponent = 10110111

Biased Exponent (Decimal)= 183

For Exponent being 8 bit, Bias = $2^{(8-1)} - 1 = 127$

Exponent (Decimal)= 183 - 127

= 56

Fraction/ Mantissa = 0. 10010100000000000000000

Y (Binary Normalized) = 1.Franction x 2^(Exponent)

Y (Binary Normalized) = 1. 10010100000000000000000 x 2^56

Floating Point Addition/ Subtraction

- 7ACD0000 + 5BCA0000

Rule: Match the Lower Exponent with the Higher Exponent

X = 51500000

X (Binary Normalized) =
1. 10011010000000000000000 x 2^{118}

Y = 3A10A000

Y (Binary Normalized) =
1. 10010100000000000000000 x 2^{56}

Y (Binary Normalized) =
0.[61 0s..] **10010100000000000000000** x 2^{118}

X + Y =

=**1. 10011010000000000000000 x 2^{118} +0.[61 0s..] 10010000101000000000000 x 2^{118}**

= (**1. 10011010000000000000000** + **0.[61 0s..] 10010000101000000000000**) x 2^{118}

= 1.1001101[54 0s..] **1**0010000101000000000000 x 2^{118}

8. Suppose X=19.454 and Y=3.0124, perform X*Y using IEEE floating-point representation.

Answer8:

X= 19.454

X (Binary) = 10011.01110100

X (Normalized) = 1.001101110100 x 2^4

Y= 3.012

X (Binary) = 11.0000001100

X (Normalized) = 1.10000001100x 2^1

X * Y =

(1.001101110100 x 2^4) x (1.10000001100x 2^1)

= (1.001101110100 x 1.10000001100) x 2^{4+1}

= 1.11010100101100101110000 x 2^5

= 111010.100101100101110000

= 58.58734…(Decimal)

```
             1.001101110100
           x  1.10000001100
---------------------------
             0000000000000
                         x
          1001101110100xx
         1001101110100xxx
    1001101110100 xxxxxxxxxx
   1001101110100 xxxxxxxxxxx
---------------------------
1.11010100101100101110000
```

Practice

Consider the value 63.7813

a) Let’s assume you have a 21-bit register having 6-bit for exponent. Now convert this value using IEEE floating-point representation. Also, convert this into hexadecimal form.

Ans: 49FC8

b) Let’s assume you have a 12-bit register having 4-bit for exponent. Now convert this value using IEEE floating-point representation. Also, convert this into hexadecimal form

Ans: 67F

c) Suppose X= -9.435 and Y= 15.129, perform X*Y using IEEE floating-point representation.

Ans: -142.719955... (Decimal)

MIPS Division

- Use HI/LO registers for result
  - HI: 32-bit remainder
  - LO: 32-bit quotient
- Instructions
  - div rs, rt / divu rs, rt
  - No overflow or divide-by-0 checking
    - Software must perform checks if required
  - Use mfhi, mflo to access result

Floating Point Registers and Instructions - Chap 3

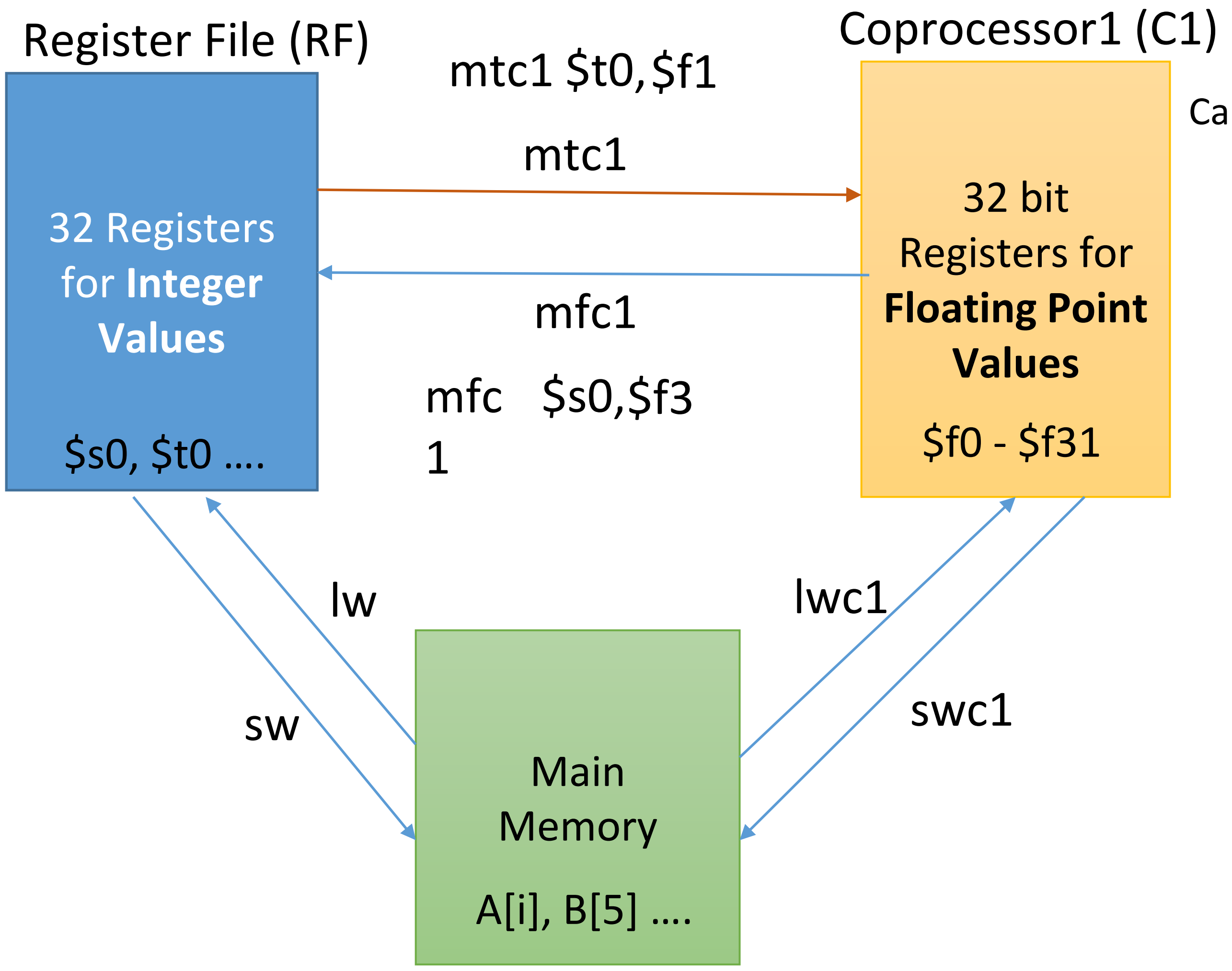
Prepared by Fairoz Nower Khan

Floating Point

mtc1, mfc1 convention ->
Instruction $register, $coprocessor Register

Move the value of $t0 to $f1

Can not contain Float Values



Can contain Integer Values

- A[5] = (float)x

Where x is in $f0 and base address of A in $s1

swc1 $f0, 20($s1)

- (float)a = B[2]

Where a is in $f2 and base address of B in $s0

lwc1 $f2, 8($s0)

MIPS Instruction for Floating Point Register

For Single Precision Float (32 bit)

add.s $f0, $f1, $f2 $f0 = $f1 + $f2

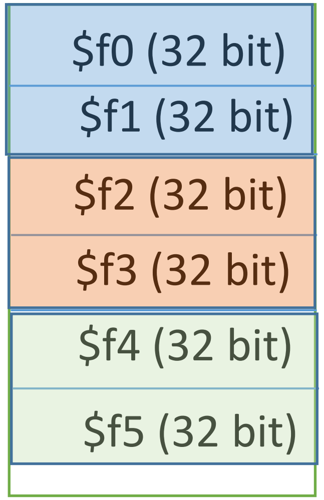
sub.s $f0, $f1, $f2 $f0 = $f1 - $f2

For Double Precision Float (64 bit)

add.d $f0, $f2, $f4

Since each register is 32 bit, 2 consecutive Registers are considered

$f0, $f1 (64 bit) = $f2, $f3 (64 bit) + $f4, f5 (64 bit)



Load from and Store to Main Memory

Single Precision (32 bit)

lwc1 $f0, 0($s1)
swc1 $f1, 0($s1)

Double Precision (64 bit)

ldc1 $f0, 0($s1)
sdc1 $f1, 0($s1)

MIPS Instruction for Floating Point Register

Moving Value from Registers

mtc1	$s0, $f1	Move the value from $s0 (Register file) to $f1 (Coprocessor1)
mfc1	$s1, $f2	Move the value from $f2(Coprocessor1) to $s1 (Register File)

Convert a Register value from Float to Integer for Single Precision Float (32 bit)

cvt.w.s	$f0, $f1	Converts the value of $f1 from Float to Integer and stores in $f0

Convert a Register value from Integer to Float for Single Precision Float (32 bit)

cvt.s.w	$f0, $f1	Converts the value of $f1 from Integer to Float and stores in $f0
cvt.s.d	$f3, $f1	Converts the value of $f1 from double to single precision
cvt.d.s	$f3, $f1	Converts the value of $f1 from single to double precision

Practice Problem

C Code:

(float)x = (int)y + (float)z
(int)y = (float)x - (float)z

Suppose x, y and z is in $f1, $s1 and $f2.

MIPS Code:

As we can move a int value in float register,

```
mtc1  $s1,  $f3
cvt.s.w $f3,  $f3
add.s  $f1, $f3, $f2
sub.s  $f4, $f1, $f2
cvt.w.s $f4,  $f4
mfc1  $s1,  $f4
```

(moving integer y to float register f3)

(converting integer y to Single precision float and store in register f3 to perform arithmetic operation)

Since we cannot store a float in integer Register we need to covert the float to integer

3. Consider the equation and write MIPS code for it: X=(A[4]+B[2])+(B[3]-5X); Assume array A stores floating-point values and its base address in $s0 and array B stores integer values and its base address is in $s1 register. X is in register $s2.

Answer3:

```
lwc1 $f1, 16($s0)
lwc1 $f2, 8($s1)
cvt.s.w $f2, $f2
add.s $f1, $f1, $f2            # f1 = A[4] + B[2]

lw $t0, 12($s1)
sll $t1, $s2, 2
add $t1, $t1, $s2
sub $t1, $t0, $t1              # t1 = B[3] - 5X
mtc1 $t1, $f2
cvt.s.w $f2, $f2

add.s $f1, $f1, $f2            # f1 = (A[4]+B[2])+(B[3]-5X)
cvt.w.s $f1, $f1
mfc1 $s2, $f1                  # X = $s2 = $f1
```