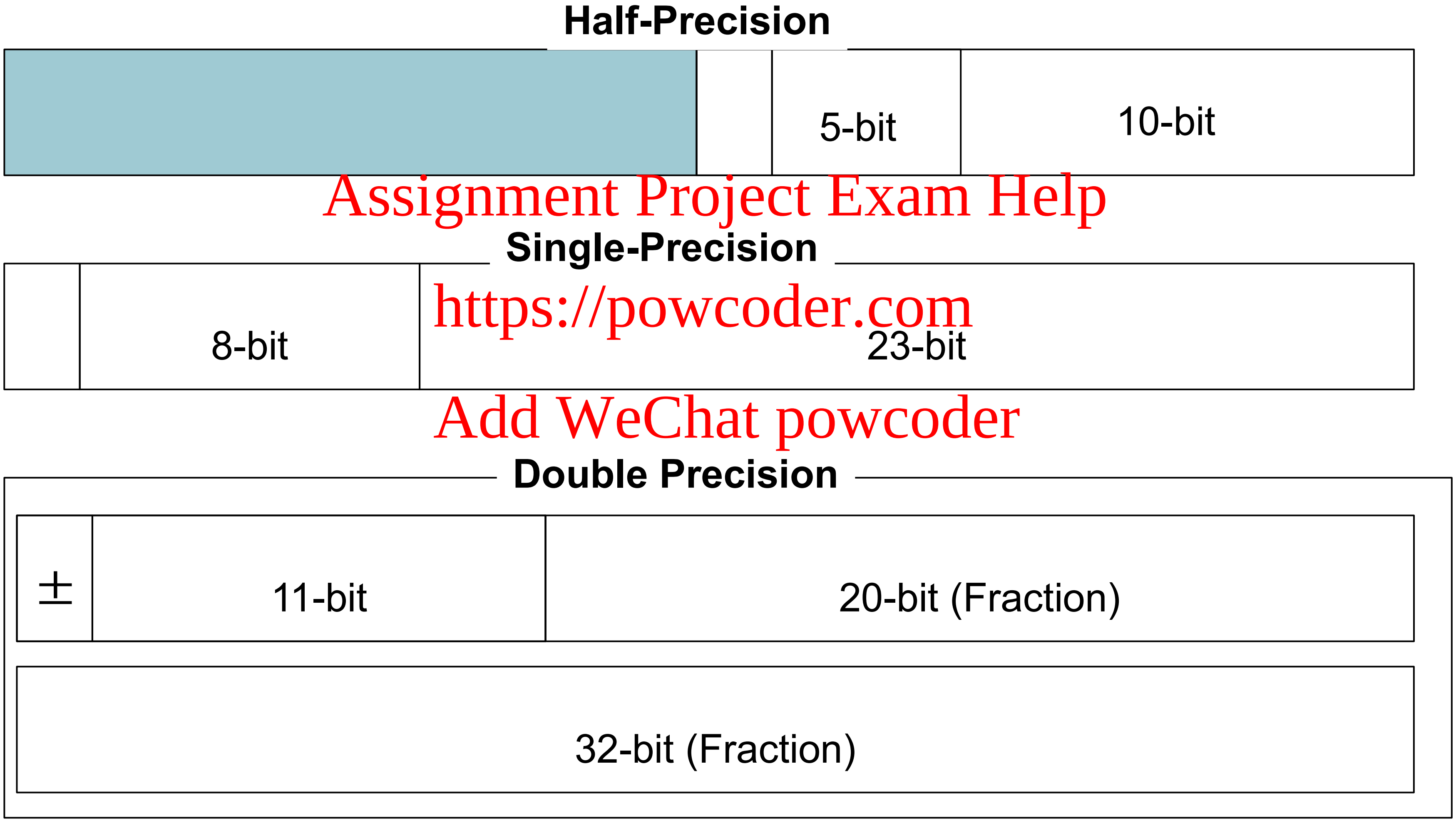
Project #2:
Half Precision Arithmetic

IEEE Floating-Point Format



Half-Precision

5-bit

10-bit

Single-Precision

8-bit

23-bit

Double Precision

±

11-bit

20-bit (Fraction)

32-bit (Fraction)

Pseudo Half-Precision (PHP)

- PHP is the same as single precision except for
  - At least 13 rightmost bits in the fraction are zeroed out
    - More bits are zeroed out in denorm numbers
  - The range of exponent for normal numbers is limited either 0, 255, or between -14 and 15
- Both PHP and single precision number have identical special numbers .
- In summary, **PHP is simply a subset of single precision numbers which are all the numbers in half-precision**

Pseudo Half –Precision (PHP)

±	8-bit 0,[-14,15],255	10-bit	(Extra bits used for rounding)

Why PHP (Pseudo Half Precision)?

- Hold intermediate results during conversion from single precision to true half precision
- Single precision arithmetic instructions can work on PHP numbers, not on true half precision numbers.
  - PHP **is simply a subset of single precision numbers.**
  - The 23 bits in fraction can hold intermediate results for rounding purpose
  - Can print PHP like single precision

```
li          $v0,  2 # print both single precision and PHP
syscall
```

Pseudo Half –Precision (PHP)

±	8-bit 0,[-14,15],255	10-bit	(Extra bits used for rounding)

Single Precision → PHP

Single–Precision X (input in $f12)

	8-bit	23-bit

- If X = infinity, NaN, then y=x
- If |X| > 65504, then y = infinity
- If $|X| < 2^{-24}$, then , y = 0
- If 2^{-24} $|X| < 2^{-14}$, then y is denorm
  - y=x with fraction rounded to < 10 bits

    Otherwise,
  - y=x with fraction rounded to 10 bits

Function Call

jal **cvt.php.s**

Pseudo Half–Precision Y (output in $f0)

±	8-bit 0,[-14,15],255	10-bit	(Extra bits used for rounding)

Example: Single Precision → PHP

Single–Precision X (input in $f12)

	0111 1111	1111 1111 11, 00 0000 0000 111

X =1.1111 1111 11, ~~0000000000 111~~ 2^0

(Round down)

cvt.php.s:

```
mfc1      $t0,     $f12
# mark off 13 LSB bits
andi      $t0,    $$t0,  0xFFFFE000
mtc1      $t0,     $f0
Jr        $ra
```

Y =1.11111111111 0000000000000 2^0

Pseudo Half –Precision Y (output in $f0)

±	0111 1111	1111 1111 11	0000 0000 0000 0

Example: Single Precision → PHP

Single–Precision X (input in $f12)

	0111 1111	1001 1111 11, 10 0000 0000 111

X =1.1001 1111 11, ~~10 0000 0000 111~~ 2^0

(Round up)

cvt.php.s:

```
mfc1      $t0,     $f12
# mark off 13 LSB bits
andi      $t0,     $$t0, 0xFFFFE000
addi      $t0,     0x00002000
mtc1      $t0,     $f0
Jr        $ra
```

Y =1.1010 0000 00 0000000000000 2^0

Pseudo Half –Precision Y (output in $f0)

±	0111 1111	1010 0000 00	0000 0000 0000 0

Example: Single Precision → PHP

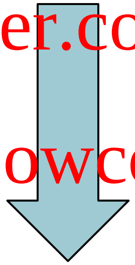
Single–Precision X (input in $f12)

	1000 1111	1111 1111 11, 10 0000 0000 111

```
cvt.php.s: # infinity
l.s         $f13,   LARGEST
c.le.s      $f12,   $f13
bc1t        normal
l.s         $f0,    infinity
Jr          $ra
Normal:
  # consider other cases
```

X = 1.1111111111, 1000000000111 2^{16}

(> largest PHP 1.1111111111 2^{15})



Y = Infinity

Pseudo Half–Precision (output in $f0)

±	1111 1111	0000 0000 00	0000 0000 0000 0

Example: Single Precision → PHP

Single–Precision X (input in $f12)

	1111 1111	0011 1111 11, 10 0000 0000 111

cvt.php.s:

```
l.s       $f13,   LARGEST
c.gt.s    $f12,   $f13
bc1t      else
Move.s    $f0,    $f12
Jr        $ra
else:
  # consider other cases
```

Y = NaN

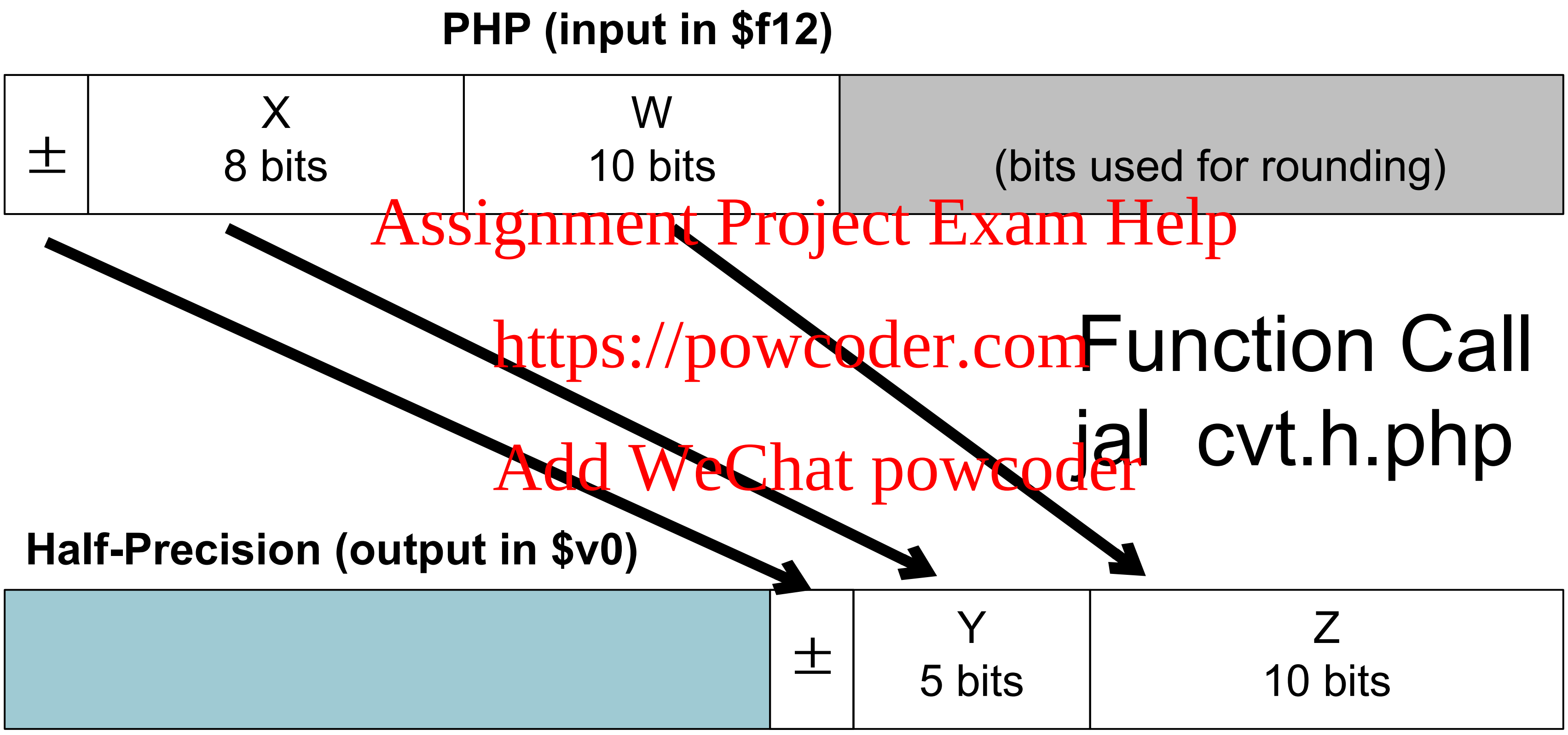
Pseudo Half–Precision (output in $f0)

±	1111 1111	1111 1111 11	0000 0000 0000 0

~~PHP → Single Precision~~

- There is no need to convert PHP to single precision numbers.
  - PHP numbers are simply a subset of single precision numbers.

Half Precision ← PHP



PHP (input in $f12)

± | X 8 bits | W 10 bits | (bits used for rounding)

Half-Precision (output in $v0)

± | Y 5 bits | Z 10 bits

Function Call

jal cvt.h.php

If X=255 → Y=31 (infinity, NaN)

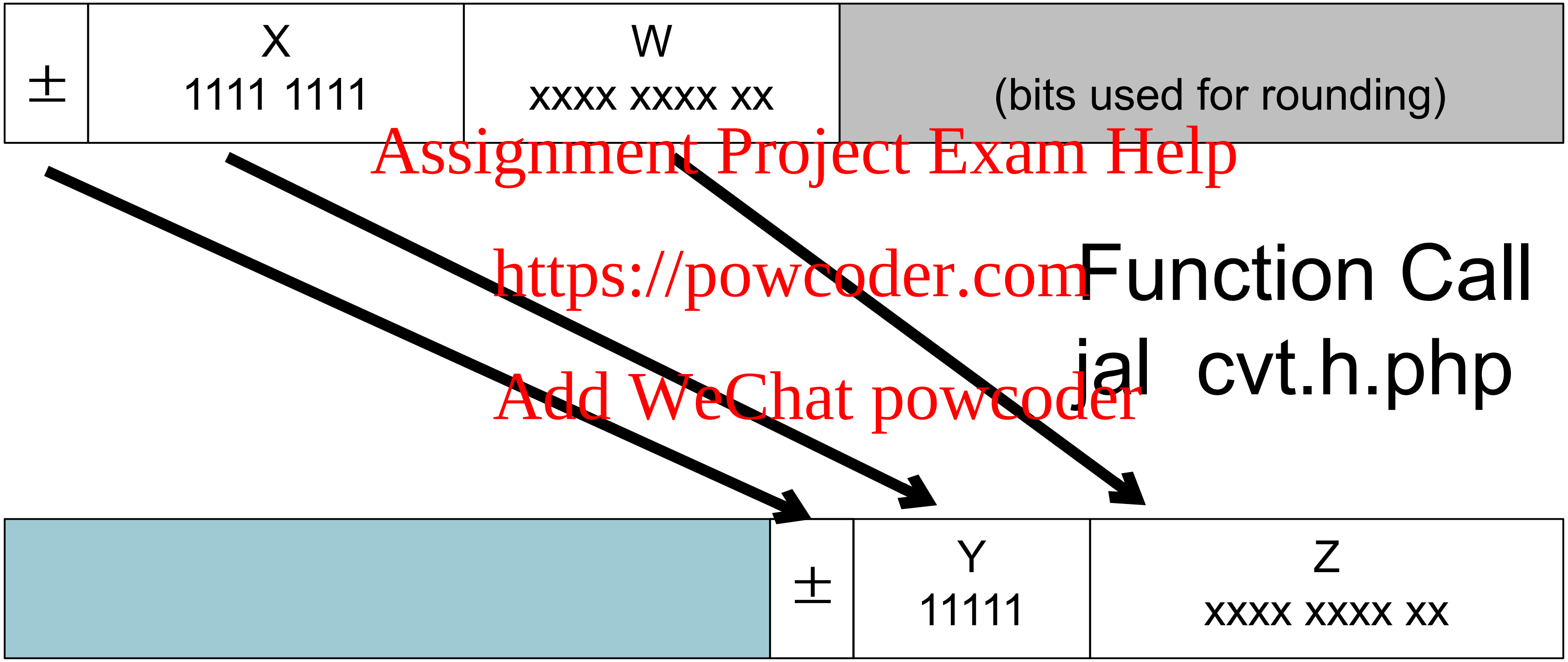
If |f12| 10^{-24} → Y = Z = 0 (Zero)

If |f12| 10^{-14} (normal) → Y=X-127 +15, Z = W

Otherwise (f12 is denorm) → Y=0, Z= 1.W >> -(X-127+15)

Example: Half Precision ← PHP

PHP (input in $f12)

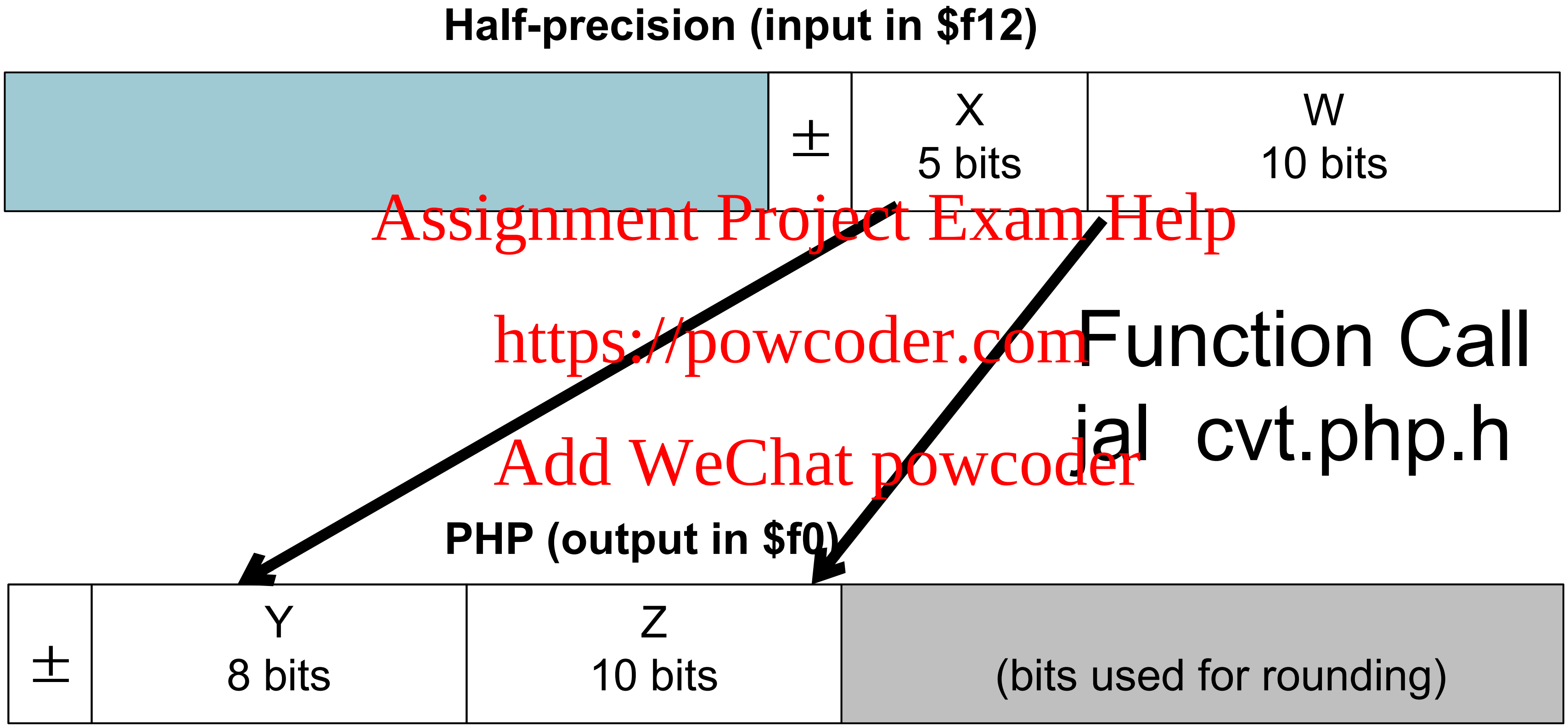


Function Call
jal cvt.h.php

Half-Precision (output in $v0)

If X=255 → Y=31 (infinity, NaN)

Half Precision → PHP



Half-precision (input in $f12)

	±	X 5 bits	W 10 bits

PHP (output in $f0)

±	Y 8 bits	Z 10 bits	(bits used for rounding)

Function Call

jal cvt.php.h

- If X=31 → Y= 255, Z=W (infinity, NaN)
- If 0<X<31 (f12 normal) → **Y=X-15 + 127, Z=W**
- If **X=0 (f12 denormal)** → **Y=0-15+127-n, Z = W<<n,**
  - **where n=the position of rightmost 1 bit in W.**
  - **e.g. if W= 0010011000, then n = 3**

Special Numbers in PHP

Both PHP and single precision use the same representation and arithmetic rules for special numbers infinity and NaN

Infinity infinity = infinity
1   0 = infinity
infinity   finite number = infinity
Infinity – infinity = NaN

Functions to Implement

- Conversion between float and PHP
  - **cvt.php.s**
    - **Input**: single precision number **$f12**,
    - **ouput**: php number **$f0**
  - **cvt.h.php**:
    - Input: a PHP number $f12,
    - Output: a half precision $f0
  - **cvt.php.h:**
    - Input: a half precision $a0,
    - Output: a PHP number $f0

- Take care of **special** numbers (Infinity, NaN, denorm)
- No need to implement cvt.s.php (why?)

Functions to Implement

- Half Precision Arithmetic:
  - **add.php,**
  - sub.php,
  - **mul.php,**
  - div.php
- **Input**: Single precision numbers A,B in **$f12, $f13**
- **Output**: PHP number C = A op B in **$f0**

Testing

Testing programs are provided to

- Test cvt.php.h and cvt.h.php
- Test for special numbers
  - Infinity + Infinity = Infinity
  - Infinity - Infinity = NaN
  - NaN + X = NaN for any number X
- Test the half precision arithmetic using the examples from exercises 3.30-3.39 in the textbook

Test Single → PHP → Half Precision

```
# ex.3.27.asm:   Exercise 3.27
.data
      A:        .float    -0.15625       # Single Precision
.text
      l.s       $f12,       A
      jal       cvt.php.s                # Single → PHP

      mov.s     $f12,       $f0
      jal       cvt.h.php                # PHP      → Half

      mov.s     $a0,        $f0
      li        $v0,        34           # print HEX
      syscall                            # encoding
```

add.php: C = A+B in PHP

```
add.php:
          # input:       float  A and B         in $f12, $f13
          # output:      php  C = A+B           in $f0
          add            $sp,    $sp,    -4
          sw             $ra,    ($sp)
          #              input A is in $f12
          jal            cvt.php.s      # Convert A to PHP
          mov.s           $f2,    $f0    # move A to $f2
          mov.s           $f12,   $f13 #
          jal            cvt.php.s      # Convert B to PHP
          add.s          $f12,   $f2,    $f0   # C=A+B
          jal            cvt.php.s # convert C to PHP
          lw             $ra,    ($sp)
          add            $sp,    $sp,    4
          jr             $ra
```

mul.php: C = A*B in PHP

```
mul.php:
        # input:      float  A and B        in $f12, $f13
        # output:     php  C = A+B          in $f0
        add           $sp,    $sp,    -4
        sw            $ra,    ($sp)
        #             input A is in $f12
        jal           cvt.php.s       # Convert A to PHP
        mov.s          $f2,    $f0    # move A to $f2
        mov.s         $f12,    $f13 #
        jal           cvt.php.s       # Convert B to PHP
        mul.s         $f12,   $f2,    $f0   # C=A+B
        jal           cvt.php.s # convert C to PHP
        lw            $ra,    ($sp)
        add           $sp,    $sp,    4
        jr            $ra
```

Your works

- sub.php: C = AB in PHP format
- div.php: C = AB in PHP format

Exercise 3.30: C=A*B

```
.data
          A:        .float              –8.0546875,
          B :       .float              1.79931640625 x10^–1
.text
          i.s         $f12,     A
          i.s         $f12,     B
#   No need to convert A and B to PHP here
#   since mul.php converts floats to PHP automatically
          jal                   mul.php               # php C=A*B in $f0
         mov.s                   $f12,      $f0     # put C in $f12
          li                    $v0, 2                # print decimal value of C
          syscall

          jal                    cvt.h.php
          mfc1                  $a0,      $f0        # half-precision C in $a0
          mfc1                  $a0,      $f0
          li                    $v0,       34
          syscall                # print half-precision encoding of C
```

Exercise 3.31: C=A/B

```
.data
          A:        .float              –8.0546875,
          B :       .float               1.79931640625 x10^–1
.text
          i.s         $f12,     A
          l.s         $f13,     B
#   No need to convert A and B to PHP here
#   since div.php will convert floats to PHP first
          jal         div.php                 # php C = A/B in $f0
          mov.s                 $f12,   $f0     # move C to $f12
          li                    $v0, 2        # print decimal value of C=A*B
          syscall
          Jal                   cvt.h.php    # convert to half precsion
          mfc1                  $a0,     $v0   $ move php A to $a0
          li                    $v0,      34
          syscall                # print half-precision encoding of C
```

Exercise 3.32 and 3.33 (A+B)+C = A+(B+C) ???

- Testing program ex.3.32.asm
  - Compute (A+B)+C
- Testing program ex.3.33.asm
  - Compute A+(B+C)
- Verify the program outputs against the posted answer key