CSci370 Computer Architecture: Homework 1 Solutions

Due date: On or before Thursday, February 13, 2020 in class
Absolutely no copying others' works **Name:** ___________________________

- The purpose of homeworks is for students to practice for the exams without others' help, so the penalty of mistakes will be minor.
- Without practicing for the exams properly, students would not be able to do well on the exams.

---

1. The following table shows manufacturing data of a processor:

Wafer Diameter	Dies per Wafer	Defects per Unit Area	Cost per Wafer
40 cm	80	0.04 defects/cm^2	30

a. (15%) Find the yield.
Ans>

```
  Wafer area
= 3.14159 × 20.0² cm²
= 1256.636 cm²

  Die area
= 1256.636 cm² / 80
= 15.71 cm²

  Yield
= 1 / ( 1 + Defects per area × Die area / 2 )²
= 1 / [ 1 + ( 0.04 defects/cm² ) × ( 15.71 cm² ) / 2 ]²
= 0.58
```

b. (10%) Find the cost per die.
Ans>

```
  Cost per die
= Cost per wafer / ( Dies per wafer × Yield )
= 30 / ( 80 × 0.58 )
= 0.65
```

c. (15%) If the number of dies per wafer is increased by 20% and the defects per area unit increase by 30%, find the die area and yield.
Ans>

```
  Wafer area
= 3.14159 × 20.0² cm²
= 1256.636 cm²

  Die area
= 1256.636 cm² / ( 80 × 1.2 )
= 13.09 cm²

  Yield
= 1 / ( 1 + Defects per area × Die area / 2 )²
= 1 / [ 1 + ( 0.040 × 1.30 defects/cm² ) × ( 13.09 cm² ) / 2 ]²
= 0.56
```

2. Compilers can have a profound impact on the performance of an application. Consider the following two compilers for a program:

Compiler A		Compiler B	
Instruction count	Execution time	Instruction count	Execution time
1.2×10^9	4.5 s	1.8×10^9	6.0 s

a. (20%) Find the average CPI for each program given that the processor has a clock cycle time of 3 ns (or 3×10^{-9} s).
Ans>

$\underline{CPI_A}$
$= \underline{\text{CPU time}_A} / (\underline{\text{Instruction count}_A} \times \underline{\text{Clock cycle time}_A})$
$= 4.5\text{ s} / (1.2\times10^9 \text{ instructions} \times 3\times10^{-9} \text{ s/cycle})$
= 1.25 cycles/instruction

$\underline{CPI_B}$
$= \underline{\text{CPU time}_B} / (\underline{\text{Instruction count}_B} \times \underline{\text{Clock cycle time}_B})$
$= 6.0\text{ s} / (1.8\times10^9 \text{ instructions} \times 3\times10^{-9} \text{ s/cycle})$
= 1.11 cycles/instruction

b. (20%) Assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?
Ans>

$\underline{\text{Clock rate}_A} / \underline{\text{Clock rate}_B}$ (or $\underline{\text{Clock cycle time}_B} / \underline{\text{Clock cycle time}_A}$)
$= [(\underline{CPI_A} \times \underline{\text{Instruction count}_A}) / \underline{\text{CPU time}}] / [(\underline{CPI_B} \times \underline{\text{Instruction count}_B}) / \underline{\text{CPU time}}]$
$= (1.25 \text{ cycles/instruction} \times 1.2\times10^9 \text{ instructions}) /$
$(1.11 \text{ cycles/instruction} \times 1.8\times10^9 \text{ instructions})$
= 0.75

Therefore, the processor A is 0.75 times faster than the processor B.

c. (20%) A new compiler is developed that uses 5.0×10^8 instructions and has an average CPI of 1.5. What is the speedup of using this new compiler versus using Compiler A on the original processor?
†Hint: The speedup here is equal to T_A/T_n where
- T_A: the execution time by using the A compiler and
- T_n: the execution time by using the new compiler.

Ans>

T_A/T_n
$= (\underline{\text{Instruction count}_A} \times \underline{CPI_A} \times \underline{\text{Clock cycle time}_A}) /$
$(\underline{\text{Instruction count}_n} \times \underline{CPI_n} \times \underline{\text{Clock cycle time}_n})$
$= (1.2\times10^9 \text{ instructions} \times 1.25 \text{ cycles/instruction} \times 3\times10^{-9} \text{ s/cycle}) /$
$(5.0\times10^8 \text{ instructions} \times 1.50 \text{ cycles/instruction} \times 3\times10^{-9} \text{ s/cycle})$
= 2.0

CSci370 Computer Architecture: Homework 3

Due date: On or before Monday, April 06, 2020 in class
Absolutely no copying others' works **Name: ____________________________**

- The purpose of homeworks is for students to practice for the exams without others' help, so the penalty of mistakes will be minor.
- Without practicing for the exams properly, students would not be able to do well on the exams.

---

1. **(Floating-point representation: 20%)** What decimal number does the hexadecimal number $(\text{AC9D 4800 0000 0000})_{16}$ represent if it is a floating point number? Use the IEEE 754 standard.
Ans>

1	0	1	0	1	1	0	0	1	0	0	1	1	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Sign = 1 is negative.
Exponent = $01011001001_2 = 2^0 + 2^3 + 2^6 + 2^7 + 2^9 = 1 + 8 + 64 + 128 + 512 = 713$
E - bias = 713 - 1023 = -310
Significand = $1.1101010010000000..._2 = 1 + 2^{-1} + 2^{-2} + 2^{-4} + 2^{-6} + 2^{-9} = 1 + 0.5 + 0.25 + 0.0625 + 0.015625 + 0.001953125 = 1.830078125$
Therefore, the value in decimal is $-1.830078125 \times 2^{-310}$

2. **(Floating-point arithmetic: 80%)** IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Assuming the three numbers, A=1.237219, B=2.524723, and C=6.742837, are stored in the 16-bit IEEE 754-2008 format, calculate A×B-C by hand. Assume 1 guard bit, 1 round bit, and 1 sticky bit, and round to the nearest even (only on the Step 4. Rounding the Significand). Show all the steps in normalized floating point numbers with 11 bits of precision (or 10 bits of fraction); i.e., $(-1)^S \times (1.F)_2 \times 2^E$, where S is the sign bit, $|F|=10$, and E is without including the bias. Note that
   - have to show all five steps step-by-step,
   - the calculation should be similar to those in Slide 9.11 and Slide 9.16; i.e., the numbers should use a base-2 scientific notation and include 3 extra bits each (e.g., IEEE format and bias need NOT be used),
   - to answer this question, simulate the hardware by including 3 extra bits for each operand and result and using the 3 extra bits as possible as you could,
   - if the number has to be shifted to the right on the Step 1. Making Exponents Equal, round (just using the regular rounding method) the number to 10 bits of fraction and 3 extra bits before calculation,
   - rounding to the nearest even will be applied only once after finishing all calculation because the 3 extra bits need to be kept for further calculation, and
   - 2's complement representation has to be used if needed.

   Ans>

   A.
```
  1.237219
= 1.00111100101110..._2
≈ 1.0011110010 1 1 1_2 × 2^0 (Guard=1, Round=1, Sticky=1)

because

 0.237219 × 2 = 0.474438 ⇒ 0
 0.474438 × 2 = 0.948876 ⇒ 0
 0.948876 × 2 = 1.897752 ⇒ 1
 0.897752 × 2 = 1.795504 ⇒ 1
 0.795504 × 2 = 1.591008 ⇒ 1
 0.591008 × 2 = 1.182016 ⇒ 1
 0.182016 × 2 = 0.364032 ⇒ 0
 0.364032 × 2 = 0.728064 ⇒ 0
 0.728064 × 2 = 1.456128 ⇒ 1
 0.456128 × 2 = 0.912256 ⇒ 0
 0.912256 × 2 = 1.824512 ⇒ 1
 0.824512 × 2 = 1.649024 ⇒ 1
 0.649024 × 2 = 1.298048 ⇒ 1
 0.298048 × 2 = 0.596096 ⇒ 0
 ...

 So, 0.237219 = 0.00111100101110..._2
```

   B.
```
  2.524723
= 10.10000110010101..._2
≈ 1.0100001100 1 0 1_2 × 2^1 (Guard=1, Round=0, Sticky=1)

because

 0.524723 × 2 = 1.049446 ⇒ 1
 0.049446 × 2 = 0.098892 ⇒ 0
 0.098892 × 2 = 0.197784 ⇒ 0
 0.197784 × 2 = 0.395568 ⇒ 0
 0.395568 × 2 = 0.791136 ⇒ 0
 0.791136 × 2 = 1.582272 ⇒ 1
 0.582272 × 2 = 1.164544 ⇒ 1
 0.164544 × 2 = 0.329088 ⇒ 0
 0.329088 × 2 = 0.658176 ⇒ 0
 0.658176 × 2 = 1.316352 ⇒ 1
 0.316352 × 2 = 0.632704 ⇒ 0
 0.632704 × 2 = 1.265408 ⇒ 1
 0.265408 × 2 = 0.530816 ⇒ 0
 0.530816 × 2 = 1.061632 ⇒ 1
 ...

 So, 0.524723 = 0.10000110010101..._2
```

   C.
```
  6.742837
= 110.10111110001010..._2
≈ 1.1010111110 0 0 1_2 × 2^2 (Guard=0, Round=0, Sticky=1)

because

 0.742837 × 2 = 1.485674 ⇒ 1
 0.485674 × 2 = 0.971348 ⇒ 0
 0.971348 × 2 = 1.942696 ⇒ 1
 0.942696 × 2 = 1.885392 ⇒ 1
 0.885392 × 2 = 1.770784 ⇒ 1
 0.770784 × 2 = 1.541568 ⇒ 1
 0.541568 × 2 = 1.083136 ⇒ 1
 0.083136 × 2 = 0.166272 ⇒ 0
 0.166272 × 2 = 0.332544 ⇒ 0
 0.332544 × 2 = 0.665088 ⇒ 0
 0.665088 × 2 = 1.330176 ⇒ 1
 0.330176 × 2 = 0.660352 ⇒ 0
 0.660352 × 2 = 1.320704 ⇒ 1
 0.320704 × 2 = 0.641408 ⇒ 0
 ...

 So, 0.742837 = 0.10111110001010..._2
```

```
  A × B
= 1.237219 × 2.524723
≈ (1.0011110010 1 1 1_2 × 2^0) × (1.0100001100 1 0 1_2 × 2^1)
```

Step 1. Calculating the exponent of the product

```
0 + 1 = 1
```

Step 2. Multiplying the significands

```
                  1.0011110010111
 ×                1.0100001100101
 -------------------------------------
                   10011110010111
                  00000000000000
                 10011110010111
                00000000000000
               00000000000000
              10011110010111
             10011110010111
            00000000000000
           00000000000000
          00000000000000
         00000000000000
        10011110010111
       00000000000000
 +    10011110010111
 -------------------------------------
      110001111110011011010010011 or
     1.10001111110011011010010011

  (1.0011110010 1 1 1_2 × 2^0) × (1.0100001100 1 0 1_2 × 2^1)
= 1.10001111110011011010010011_2 × 2^1
```

Step 3. Normalizing the product

```
  1.10001111110011011010010011_2 × 2^1
≈ 1.1000111111 0 0 1_2 × 2^1 (Guard=0, Round=0, Sticky=1)
```

```
  A × B − C
≈ (1.1000111111 0 0 1_2 × 2^1) − (1.1010111110 0 0 1_2 × 2^2)
```

Step 1. Making exponents equal

```
  1.1000111111 0 0 1_2 × 2^1
= 0.1100011111 1 0 1_2 × 2^2 (Guard=1, Round=0, Sticky=1)
```

Step 2. Performing subtraction

```
   0.1100011111 1 0 1_2 × 2^2 (Guard=1, Round=0, Sticky=1)
−  1.1010111110 0 0 1_2 × 2^2 (Guard=0, Round=0, Sticky=1)
                                ⇓
  00.1100011111 1 0 1_2 × 2^2
+ 10.0101000001 1 1 1_2 × 2^2 (2's complement)
--------------------------------------------------
  11.0001100001 1 0 0_2 × 2^2 (negative)
= −0.1110011110 1 0 0_2 × 2^2 (Guard=1, Round=0, Sticky=0)
```

Step 3. Normalizing the difference

```
   −0.1110011110 1 0 0_2 × 2^2 (Guard=1, Round=0, Sticky=0)
 = −1.1100111101 0 0_2   × 2^1 (Round=0, Sticky=0)
```

Step 4. Rounding the significand

```
   −1.1100111101 0 0_2 × 2^1 (Round=0, Sticky=0)
 ≈ −1.1100111101_2     × 2^1
```

Step 5. Checking for overflow or underflow

```
  −1.1100111101_2 × 2^1
= −11.100111101_2
= −(2^1 + 2^0 + 2^-1 + 2^-4 + 2^-5 + 2^-6 + 2^-7 + 2^-9)
= −(2 + 1 + 0.5 + 0.0625 + 0.03125 + 0.015625 + 0.0078125 + 0.001953125)
= −3.619140625
≈  1.237219 × 2.524723 − 6.742837
= −3.61920173
```

CSci370 Computer Architecture: Homework 2 (double-sided)

Due date: On or before Monday, March 30, 2020 in class
Absolutely no copying others' works **Name:** ____________________________

- There are four algorithms discussed for multiplication and division. Make sure you are using the correct ones for the two questions below.
- The purpose of homeworks is for students to practice for the exams without others' help, so the penalty of mistakes will be minor.
- Without practicing for the exams properly, students would not be able to do well on the exams.

1. **(Refined multiplication: 50%)** Using a table similar to that shown in the Slide 8.6, calculate the product of the octal unsigned 6-bit integers 65_8 (or 110101_2) and 53_8 (or 101011_2) using the hardware and algorithm described in the figures of Slide 8.6. You should show the content of each register on each step.
Ans>

Iteration		Multiplicand	Carry	Product = HI, LO
0	Initialize (LO=Multiplier, HI=0)	110101		000000 101011
1	LO[0]=1 ⇒ Add.		0	000000 101011 + 110101 000000 110101 101011
	Shift product right by 1 bit.	110101		011010 110101
2	LO[0]=1 ⇒ Add.		1	011010 110101 + 110101 000000 001111 110101
	Shift product right by 1 bit.	110101		100111 111010
3	LO[0]=0 ⇒ Do nothing.			
	Shift product right by 1 bit.	110101		010011 111101
4	LO[0]=1 ⇒ Add.		1	010011 111101 + 110101 000000 001000 111101
	Shift product right by 1 bit.	110101		100100 011110
5	LO[0]=0 ⇒ Do nothing.			
	Shift product right by 1 bit.	110101		010010 001111
6	LO[0]=1 ⇒ Add.		1	010010 001111 + 110101 000000 000111 001111
	Shift product right by 1 bit.	110101		100011 100111

2. **(First-version division: 50%)** Using a table similar to that shown in the Slide 8.10, calculate the octal unsigned 6-bit integer 65_8 (or 110101_2) divided by another octal unsigned 6-bit integer 16_8 (or 001110_2) using the hardware and algorithm described in the figures of Slide 8.10. You should show the content of each register on each step.
†Note that you have to actually show the differences in the procedures, not just the signs.
Ans>

Iteration		Remainder	Divisor	Difference	Quotient
0	Initialize	000000 110101	001110 000000		000000
1	1: SHR, SHL, Difference	000000 110101	000111 000000	000000 110101 - 000111 000000 ⇓ 000000 110101 + 111001 000000 111001 110101	000000
	2: Diff<0 ⇒ Do Nothing				
2	1: SHR, SHL, Difference	000000 110101	000011 10000	000000 110101 - 000011 100000 ⇓ 000000 110101 + 111100 100000 111101 010101	000000
	2: Diff<0 ⇒ Do Nothing				
3	1: SHR, SHL, Difference	000000 110101	000001 110000	000000 110101 - 000001 110000 ⇓ 000000 110101 + 111110 010000 111111 000101	000000
	2: Diff<0 ⇒ Do Nothing				
4	1: SHR, SHL, Difference	000000 110101	000000 111000	000000 110101 - 000000 111000 ⇓ 000000 110101 + 111111 001000 111111 111101	000000
	2: Diff<0 ⇒ Do Nothing				
5	1: SHR, SHL, Difference	000000 110101	000000 011100	000000 110101 - 000000 011100 ⇓ 000000 110101 + 111111 100100 + 000000 011001	000000
	2: Diff≥0 ⇒ Rem=Diff, set lsb Quotient	000000 011001			000001
6	1: SHR, SHL, Difference	000000 011001	000000 001110	000000 011001 - 000000 001110 ⇓ 000000 011001 + 111111 110010 + 000000 001011	000010
	2: Diff≥0 ⇒ Rem=Diff, set lsb Quotient	000000 001011			000011