1. **[10 points]** We know that the execution time it takes to run a program equals IC×CPI×CT, where IC stands for instruction count, CPI stands for cycle-per-instruction, and CT stands for cycle time. A program is an implementation of an *algorithm* using some *programming language* and compiled by some *compiler* into machine instructions of some *ISA*. Please list the influences of the four factors: algorithm, programming language, compiler, and ISA, on the three peromance variables: IC, CPI, and CT of program execution.

   [Solution]

   CPU performance depends on the following factors:

   - Algorithm: affects IC, CPI
   - Programming language: affects IC, CPI
   - Compiler: affects IC, CPI
   - Instruction set architecture: affects IC, CPI, CT

   #

2. **[15 points]** Please answer the following questions about the calling convention used by MIPS GCC.

   (a) What is the usage differences between `$t0 ~ $t9` and `$s0 ~ $s7`.

   (b) When you write a function in C code, what is the requirements for the compiled instruction of that function to avoid using the stack area.

   (c) For 32-bit MIPS processors, if the return value of a function is larger than 64-bit, what would the compiler do?

   [Solution]

   (a) A function must save any of the `$s` registers onto the stack or into $t# registers before use and restore the used `$s` registers before return. It does not have to do so for the temporary registers `$t`.

   (b) The parameters of the function must be less than or equal to 4 words. The function shall not use the stack to store local variables and `$s` registers. The function must be a leaf function. The return value must be less than two words.

   (c) The compiler will allocate a buffer in the stack area of the function to store the returned data, and return a pointer to that buffer in $v0. #

3. **[10 points]** The register numbers of `$s2` and `$zero` are 17 and 0, respectively. Given the MIPS code sequence below, if we assume it starts at the address $(80004000)_{16}$, what is the machine code for the instruction `beq` in the sequence?

```
         add  $t0, $zero, $zero
loop:    beq  $s2, $zero, finish
         add  $t0, $t0, $s1
         sub  $s2, $s2, 1
         j    loop
finish: addi $t0, $t0, 100
         add  $v0, $t0, $zero
```

   [Solution]

   The machine code is $(12\ 20\ 00\ 03)_h$. #

4. **[15 points]** Assume you must add a new instruction `rpt` into the MIPS ISA that performs both the condition evaluation and index decrement of a for-loop in one instruction. For example, the instruction

`rpt $s0, loop` will do the following operations (in C pseudo code):

```
if ($s0 > 0) {
    $s0 = $s0 - 1;
    goto loop
}
```

(a) What is the most appropriate instruction format to use for `rpt`? Why?

(b) What is the shortest sequence of MIPS instructions that performs the same operation?

[Solution]

(a) The I-type instruction format is most appropriate because we must encode a register ID and a branch offset in the instruction.

(b) The shortest equivalent MIPS sequence is:

```
slt     $t0, $zero, $s0
sub     $s0, $s0, $t0
bne     $t0, $zero, loop
```

#

5. **[10 points]** For IEEE-754 16-bit half precision format, the exponent is 5 bits wide, bias is 15, and the mantissa is 10 bits long. Assume that the binary form of A = $(0\ 00001\ 1111111100)_2$. Assume that the rounding mode is truncation and we only consider the normalized numbers.

(a) What is the largest positive B that will not cause an overflow to A + B?

(b) Give an example of a positive B such that (A – B)/2 will cause an underflow.

Your answer should be in IEEE-754 binary form.

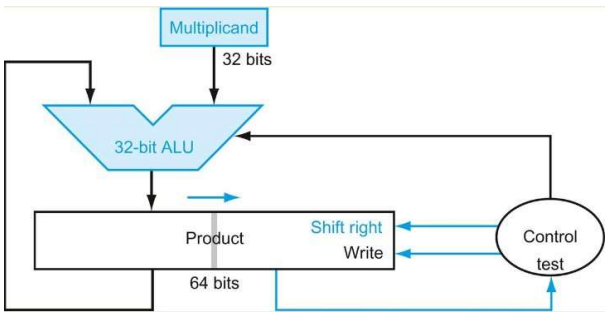
[Solution]

The exponent has 5 bits meas the dynamic range of the number is between $2^{-14} \sim 2^{15}$.

(a) This is a trick question. Overflow means the 5-bit exponent is too small to represent the result. Since A = $1.1111111100 \times 2^{-14}$ is a very small number, any large number added by A will simply cause total truncation of A, not overflow. The largest number B that can be represented in 16-bit IEEE 754 format is $1.1111111111 \times 2^{15} = (0\ 11110\ 1111111111)_2$

(b) Underflow means |A – B|/2 is less than the smallest representable number $1.0000000000 \times 2^{-14}$.
Let B = $1.1111111011 \times 2^{-14} = (0, 00001\ 1111111011)_2$, A–B = $0.0000000001 \times 2^{-14}$.
Therefore, |A – B|/2 cannot be represented using an IEEE 754 16-bit number, which results in an underflow. #

6. **[10 points]** Calculate the number of clock cycles required to perform a 32-bit × 32-bit multiplication using the following architecture to run the shift-add multiplier:



The 32-bit add operation and the 1-bit shift operation can be done in one clock cycle in parallel. Assume that the multiplicand and the product registers have already been initialized. In particular, the least significant 32 bits of the product register are initialized with the multiplier. Just count how many cycles it will take for the circuit to complete the multiplication and store the result in the product register.

[Solution]

It will take 32 cycles to perform a 32-bit × 32-bit multiplication.

7. **[15 points]** Write a MIPS assembly code to execute the following C statement without using any multiplication instructions:

```
B[65540] = A[3][i-j];
```

Assume that the variables `i` and `j` are assigned to registers `$s0` and `$s1`, respectively. Also assume that the base address of the arrays `A` and `B` are in registers `$s2` and `$s3`, respectively. The array `A` is declared as "`int A[5][15]`" and `B` is declared as "`int B[80000]`".

Your code must be as short as possible.

[Solution]

We must conpute $s2 + (3×15 + $s0 – $s1)×4 = $s2 + 180 + [($s0 – $s1)<<2] as the source address, and $s3 + (65536 + 4) × 4 = $s3 + (65536 << 2) + 16 as the target address.

```
    sub     $t0, $s0, $s1      # i - j
    sll     $t0, $t0, 2        # Adjust word addr to byte addr
    add     $t0, $t0, $s2      # &(A[0][i-j]) in $t0
    lw      $t1, 180($t0)      # A[3][i-j]in $t0

    addi    $t0, $zero, 1      #
    sll     $t0, $t0, 18       # 65536 << 2 in $t0
    add     $t0, $t0, $s3      #
    sw      $t1, 16($t0)       # B[65540] = A[3][i-j];
```

#

8. **[15 points]** Use less than ten MIPS assembly instructions to implement the following recursive C function:

```
int f(int n, int a) { return (n > 0)? f(n-1, a+n) : a; }
```

You must follow the MIPS GCC calling convention in your code.

[Solution]

```
f:  slti    $t0, $a0, 1        # test if n <= 0
    bne     $t0, $zero, _exit  # go to _exit if n <= 0
    add     $a1, $a1, $a0      # add n to a
    addi    $a0, $a0, -1       # subtract 1 from n
    j       f                  # go to f

_exit:
    add     $v0, $a1, $zero    # return value a in $v0
    jr      $ra                # return to caller
```

#