Question 1. (14 points) This question involves **8-bit signed, 2's complement** binary numbers.

(a) Give the 8-bit 2's complement binary and hexadecimal representations of the decimal number +76.

0100 1100

4C

(b) Give the binary and decimal values of the 8-bit 2's complement hexadecimal number 0xE8.

11101000 (also ok if you said –0001 1000; the question could have been a bit more precise)

-24

Powers of 2 and 16 for reference.

Number	Hex	Decimal
2^0	16^0	1
2^1		2
2^2		4
2^3		8
2^4	16^1	16
2^5		32
2^6		64
2^7		128
2^8	16^2	256
2^9		512
2^{10}		1024
2^{11}		2048
2^{12}	16^3	4096

Question 2. (15 points) For each of the following MIPS assembly language instructions:

i. Circle whether the instruction is a real machine instruction or an an assembly-language pseudo instruction, and

ii. If the instruction is a pseudo-instruction, show the actual MIPS instruction(s) that the assembler would generate for the pseudo instruction. You should give the actual machine instructions in assembly language; do not translate the instructions to binary or hex. If there is more than one possible machine instruction sequence that implements the pseudo-instruction, we will give credit for a good answer that produces the correct result even if it is not exactly the same as that which SPIM would produce.

(a) `li $s0, 17`

Circle: machine instruction (assembler pseudo-instruction)

Actual machine instruction(s) if this is a pseudo-instruction:

`addiu $s0, $zero, 17` (`addi`, `ori` or other equivalent answers also ok)

(b) `ori $t3, $a0, 0xFF`

Circle: (machine instruction) assembler pseudo-instruction

Actual machine instruction(s) if this is a pseudo-instruction:

(c) `bge $s0, $t0, there`

Circle: machine instruction (assembler pseudo-instruction)

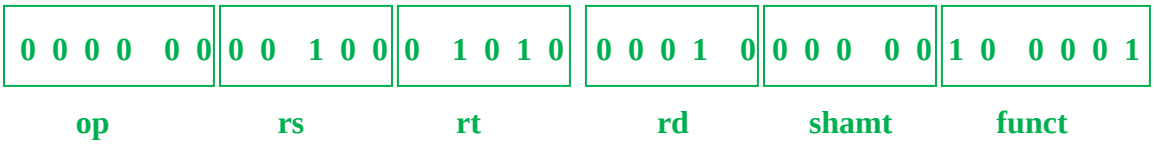
Actual machine instruction(s) if this is a pseudo-instruction:

```
slt   $at, $s0, $t0       # $at = 1 if <, or 0 if ≥
beq   $at, $zero, there
```

Equivalent instructions are fine, but using another register instead of `$at` is not, since that could clobber user data. There was also a deduction if there was more than one branch in the answer, since a real assembler would not generate that.

Question 3. (12 points) Suppose we have a 32-bit MIPS word containing the value `0x008A1021`. We would like to know what MIPS machine instruction this represents.

(a) Write this instruction word in binary. Leave enough spaces between the digits for part (c) of the question.



(b) What is the format of this instruction? (circle)

(R) I J

(c) In your answer to part (a), draw boxes around the bits that make up the different fields of the instruction and then label the instruction fields (opcode, rs, etc.)

(d) Translate this instruction to assembly language. Use symbolic register names like `$t8` instead of absolute register numbers like `$24`.

```
addu   $v0, $a0, $t2
```

Question 4. (12 points) Suppose we execute the following MIPS instructions

```
   li   $t0, 2
   li   $t1, 5
   slt  $t2, $t1, $t0
   beq  $t2, $zero, skip
   addi $t1, $t2, 3
skip:
   li   $v0, 42
```

In the following table, write down each of the registers changed during execution and their values after the code has executed. The first column is filled in for you.

Register	`$t0`	**`$t1`**	**`$t2`**	**`$v0`**			
Value	2	**5**	**0**	**42**			

Question 5. (12 points) In a processor implementation, a *data hazard* can slow down the pipeline.

What is a *data hazard*? Give a short example using MIPS code that illustrates the problem and give a brief explanation of what the problem is.

A data hazard occurs when an instruction needs data from a previous instruction before it would normally be available in the pipeline. In the example below, one of the inputs to the second `add` instruction, the value in `$t0`, is computed by the previous instruction. Without data forwarding or some other mechanism, the pipeline would have to stall long enough for the result of the first instruction to be stored in `$t0` before the second instruction could fetch its operands.

```
   add   $t0, $a0, $a1
   add   $t1, $t0, $t2
```

Question 6. (15 points) A little programming. Implement a function in MIPS assembly language to return the larger of its two integer arguments. In a C-like programming language, this function would be specified as follows:

```
/* return the larger value of a or b */
int max(int a, int b) { ... }
```

You should use the standard MIPS calling conventions for this function. However, since this is a leaf function, you do not need to allocate a stack frame if your solution does not need one. You can use regular assembly language pseudo-instructions in your solution if you wish. Don't worry if your solution is short – it might not take that much code to do the job. Comments are useful to help the grader understand your code.

```
# return the maximum value of the two integer arguments of
# this function.  If the arguments have the same value,
# return either one.

max:

# on entry: $a0 = a (first argument)
#           $a1 = b (second argument)
# result goes in $v0

   move  $v0, $a0         # answer = a (initially)
   bge   $a0, $a1, exit   # if a ≥ b then return
   move  $v0, $a1         # answer = b
exit:
   jr    $ra              # return with answer in $v0
```

There are obviously many different ways to answer this question. Some answers avoided pseudo-instructions, others returned the second argument value if the two arguments were equal, and there were other variations. All of these were fine if they returned the right answer and followed the standard argument conventions.

Question 7. (12 points) One of the terms we encountered in discussing memory hierarchies and caches was *spatial locality*. Give a brief explanation of what this term means and an short example that gives one illustration of this property.

Spatial locality is the observation that if a particular location in memory is referenced by a program, nearby locations are likely to be referenced soon. There are several good examples:

- **Instructions are normally fetched from sequential memory locations unless a branch is taken.**
- **A loop that processes an array will often reference neighboring elements in quick succession (for example copying a string, or zeroing out the contents of a data structure).**
- **If some field of an object, struct, or record is referenced, it is likely that nearby fields of the same object will be referenced also.**

Question 8. (8 points). Suppose we interpret the following word of memory as a sequence of ASCII characters. What does it say? (i.e., translate from hex to ASCII). Hint: The MIPS reference card (green card) is your friend.

`0x42796521`

Bye!