1. [8 points; 2 points for each match] Match the following. (Write the numbers (3) corresponding to column1 in the blanks provided)

Column 1	**Column 2**
1. Instruction set architecture	**4** Percentage of good dies from the total number of dies on the wafer.
2. datapath	**3** Program that translates from a higher level notation to assembly language.
3. compiler	**2** Component of the processor that performs arithmetic operations.
4. yield	**1** Specific abstraction that the hardware provides the low-level software

2. [4 points] Cost per die is inversely related to **C**.
(4) A. Cost per wafer
B. Die area
C. Dies per wafer
D. Defect per area

3. [4 points] Given the following MIPS code,
(5)

```
Funct:          addi $s0, $s1, 12
                bne $s0, $t1, Label1
                j EndFunct
Label1:         subi $s2, $s1, 1
EndFunct:       jr $ra
```

Is the procedure `Funct` a leaf or non-leaf procedure? **leaf**

4. (1) [4 points] Consider a program that takes 100 million instructions to run and has an average CPI of 1.6. Write the **expression** for the execution time for this program when run on a 400MHz machine. Do **not** calculate it out.

$$\text{Execution Time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}} = \frac{100 \times 10^6 \times 1.6}{400 \times 10^6}$$

5. (2) [12 points; 2 points each] Consider the following incomplete code. Check all the registers that need to be saved just before the call to the `compute` procedure. Write a brief explanation why each particular register is saved or not saved, in the space provided.

```
main:
     ...
     addi $t0,$0,45
     addi $s0,$0,28
     ...                    # save some registers here
     addi $a0,$0,12
     jal  compute
     ...                    # restore some registers here
     add  $s0,$t0,$s0
     li   $v0,4
     ...
     jr   $ra
```

Registers:

x `$t0` : We have set the value of `$t0` before the function call to `compute`, and we use it afterwards. Therefore, we must save it, since registers starting with `t` are "caller-saved" registers.

__ `$s0` : Although we do set `$s0` before the call to compute and use it afterwards, we need not save this register, since it is a "callee-saved" register. `compute` will have left its values alone.

__ `$a0` : We don't use this register's values later on in the code, so we don't need to save its value.

__ `$v0` : Same as above.

x `$ra` : This must be saved for us to return properly later on. The original value in `$ra` will get overwritten when we execute `jal compute`.

__ `$0` : The value of this register is always 0, so no need for saving.

6. [12 points ; 3 points each] In the following situations, compare the performance of the machines given. Ignore cache, memory, disk, and other such effects. Assume the instructions of all the chips are of the same "power" -- that no chip's instructions do more than any other's. You will receive **no credit** for wrong answers with **no work**.

(6)

You have measured the following information about processors:

CPI(PPC 604e) = 1
CPI(Uberchip) = 30
CPI(TurboSparc) = 1
CPI(UltraSparc II) = 2.5

Machine A **vs** Machine B

1) 400Mhz PPC 604e **vs** 200Mhz PPC 604e

a) <u>A is faster than B</u>
b) B is faster than A
c) They are both the same speed
d) Cannot be determined

> All other things being equal, the one with the faster clock has better performance.

2) 300Mhz PPC 604e **vs** 300Mhz Pentium-II

a) A is faster than B
b) B is faster than A
c) They are both the same speed
d) <u>Cannot be determined</u>

> There is no data given for the Pentium-II

3) 10Ghz Uberchip **vs** 300Mhz PPC 604e

a) <u>A is faster than B</u>
b) B is faster than A
c) They are both the same speed
d) Cannot be determined

$$\frac{30}{10{,}000\times10^6} \text{ vs } \frac{1}{300\times10^6}$$

$$9{,}000\times10^6 \text{ vs } 10{,}000\times10^6$$

> Since they use the same instructions, we can just use the CPI and clock rate for comparison, as in the equations above. We multiply both sides by the denominators to get the second line. We do the same thing for the next problem below.

4) 150Mhz TurboSparc **vs** 375Mhz UltraSparc II

a) A is faster than B
b) B is faster than A
c) <u>They are both the same speed</u>
d) Cannot be determined

$$\frac{1}{150\times10^6} \text{ vs } \frac{2.5}{375\times10^6}$$

$$375\times10^6 \text{ vs } 375\times10^6$$

7. [20 points] Consider a system that has a memory of size 1MB (2^{20} bytes). Data is stored in the form of blocks of size 1768 bytes. There is a resident operating system, which occupies 3 blocks of the memory. Write a MIPS program to calculate the amount available for other programs and store the value in `$s0`. You are given a procedure, `divide` that receives two argument in `$a0` and `$a1`, and returns the quotient of `$a0` ÷ `$a1`. Do not write the definition for the procedure `divide`, just include a call to it in your code. **Note:** Do **not** use the `li` instruction.

(9)

```
                              # put the value, 1MB(2^20) into $a0
lui      $a0, 32              # constant may be 100000bin, or 20hex,
                              #   or 0x0020
ori      $a0, 0               # this may be omitted since lui resets
                              #   the lower bits

addi     $a1, $zero, 1768     # put the blocksize value 1768 into $a1

jal divide                    # do the division to get the total number
                              #   of blocks

addi     $s0, $v0, -3         # subtract the space occupied by the
                              #   operating system to get the total
                              #   amount of blocks available
```

8. [5 points] The 32-bit ALU in the textbook does not need a separate subtraction unit because… (10)

a) subtraction is handled in the software level using negation.
b) subtraction is not needed due to our number format.
c) <u>subtraction is handled by the ALU using bitwise inversion, the carry bit, and addition.</u>

9. [15 points ; 5 points each] Perform the following conversions to the two's complement number 15_{hex}. (11)

1. What is the value of the number in decimal notation?

$$15_{hex} = 1 \times 16_{ten} + 5 \times 1_{ten} = 21_{ten}$$

2. Write the **<u>negative</u>** of the number in 16-bit two's complement form **<u>in binary format</u>**:

The original value in binary is:

$$0000\ 0000\ 0001\ 0101_{bin}$$

(since the binary equivalents of 1_{hex} and 5_{hex} are 0001_{bin} and 0101_{bin}, respectively) To change this to a negative value, we negate all of the bits to get:

$$1111\ 1111\ 1110\ 1010_{bin}$$

and then add 1 to the result to get

$$1111\ 1111\ 1110\ 1011_{bin}$$

3. Write the **<u>negative</u>** of the number (i.e. same value as above) in 32-bit two's complement form **<u>in hexadecimal format</u>**:

Since it's a 32-bit two's complement form, we need to sign extend the previous bit sequence we obtained above, to get:

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 1011_{bin}$$

So we get:

$$FFFF\ FFEB_{hex}$$

10. [12 points ; 2 points each] Select the most appropriate format or addressing mode for the following instructions:

(7)

a. `addi` is a(n)
   i. R-format instruction
   ii. <u>I-format instruction</u>
   iii. J-format instruction

b. `jal` is a(n)
   i. R-format instruction
   ii. I-format instruction
   iii. <u>J-format instruction</u>

c. `lb` is a(n)
   i. R-format instruction
   ii. <u>I-format instruction</u>
   iii. J-format instruction

d. `slt` is a(n)
   i. <u>Register addressing</u>
   ii. Base or displacement addressing
   iii. Immediate addressing
   iv. PC-relative addressing
   v. Pseudodirect addressing

e. `addi` is a(n)
   i. Register addressing
   ii. Base or displacement addressing
   iii. <u>Immediate addressing</u>
   iv. PC-relative addressing
   v. Pseudodirect addressing

f. `jal` is a(n)
   i. Register addressing
   ii. Base or displacement addressing
   iii. Immediate addressing
   iv. PC-relative addressing
   v. <u>Pseudodirect addressing</u>

11. [4 points] What is the difference between the two MIPS instructions, `j` and `jal`.

(8)

`jal` will save the current PC into register `$ra` before jumping to the new address, whereas `j` will not save anything.