CSCI 401 PRACTICE TEST 1

K.E. SCHUBERT

(1) You are to select a compiler to develop applications for a company with two types of computers. The company wants the best average performance with both machines. Assume all the machines are 1GHz machines.

Type	CPI 1	CPI 2	Compiler 1	Compiler 2
Arithmetic	1	1	35%	30%
Branch	6	3	25%	20%
Memory	3	5	40%	50%

If the code is 10000 lines (for either compiler) when assembled how long does it take to run on each machine?

	Compiler 1	Compiler 2
Machine 1	$1 \times .35 + 6 \times .25 + 3 \times .4 = 3.05$	$1 \times .3 + 6 \times .2 + 3 \times .5 = 3$
Machine 2	$1 \times .35 + 3 \times .25 + 5 \times .4 = 3.1$	$1 \times .3 + 3 \times .2 + 5 \times .5 = 3.4$
Machine 2	3.075	3.2

Since time is the inverse of performance, we want the lowest average and ergo pick compiler 1. If each command runs only once (a bad assumption in reality but we will use it for now), the code will run in:

machine 1: $\frac{10000 \times 3.05}{10^9} = 3.05 \times 10^{-4}$ seconds.

machine 2: $\frac{10000 \times 3.1}{10^9} = 3.1 \times 10^{-4}$ seconds.

(2) Write the MIPS assembly code for the following function. Assume the array a has been defined as size n (i.e. elements numbered from 0 to n-1). You do not need to write the code to call the function but you need to state where you assume the parameters and return address will be.

```
int poly_eval(int* a, int n, int x){
  y=a[n-1];
  for(i=n-2;i>=0;i--){
    y=y*x+a[i];
  }
  return y;
}
```

```
##############################################
# poly_eval
# leaf procedure to evaluate polynomials
# parameters:
# a0 : pointer to array of coefficients
# a1 : number of elements in array
#    : reused as current element in array
# a2 : x value
# return value:
# v0 : value of polynomial
# temporary values:
# t0 : value of a[i]
poly_eval:  sll  $a1, $a1, 2          # 4*n
            addi $a1, $a1, -4         # 4*(n-1)
            add  $a1, $a1, $a0        # address of last element in array (a[n-1])
            add  $v0, $zero, $zero    # initialize the answer
            blt  $a1,$a0, poly_done   # if no element then done
            lw   $v0, 0($a1)          # load a[n-1] to y
            beq  $a1,$a0, poly_done   # if 1 element then done
poly_do:    mult $v0, $a3             # y=y*x
            mflo $v0                  # we are assuming that the answer is small enough t
            addi $a1, $a1, -4         # next coefficient is four bytes down
            lw   $t0, 0($a1)          # next coefficient
            add  $v0, $v0, $t0        # add next coefficient
            bne  $a1, $a0, poly_do    # more coefficients left
poly_done:  jr   $ra                  # return
```

(3) Perform the indicated calculations by the algorithm requested showing all steps. Show how you get the number.

(a) -3×-6 by booth's.

$x = -3 = 1101$, $y = -6 = 1010$ and $-y = 6 = 0110$.

U	V	X	X_{-1}
0000	0000	1101	0
0110	0000	1101	0
0011	0000	1110	1
1101	0000	1110	1
1110	1000	0111	0
0100	1000	0111	0
0010	0100	1011	1
0001	0010	1101	1

$00010010 = 18$

(b) $7 - 8$ by conditional sum.

$7 = 0111$ and $-8 = 1000$

0	1	1	1
+1	0	0	0
(10,01)	(10,01)	(10,01)	(10,01)
	(100,011)	(100,011)	
	(10000,01111)		

Since this was done as addition no carry-in was set so the solution is $\begin{array}{c|c} 0 & 1111 \end{array}$ or -1 in signed base ten.

(c) 3.75×29.625 in floating point.

Convert:

$3.75 = 11.11 = 1.111 \times 2^1$

$29.625 = 11101.101 = 1.1101101 \times 2^4$

Multiply Significants:

1.	1	1	0	1	1	0	1			
0.	1	1	1	0	1	1	0	1		
0.	0	1	1	1	0	1	1	0	1	
0.	0	0	1	1	1	0	1	1	0	1
1	1.	0	1	1	1	1	0	0	0	1	1

1.10111100011×2^1

Add exponents to normalization exponent and put in excess 127:

$1 + 4 + 1 + 127 = 133 = 10000101$

Write in single precision:

0	10000101	1011 1100 0110 0000 0000 000