University of Delaware
Computer and Information Science
CISC 260 – A Sample Final Exam

1. This is an open notes exam. You are not allowed to use any electronic devices except standard calculators.
2. Check that you have all pages. There are 4 problems for a total of 100 points.
3. Write you name on every page in the space provided
4. If you need more space, use the back of the same page. But do not feel obligated to "fill up" all the space that is provided.
5. Give clear, concise answers to each question.

Problem	Grade
1	
2	
3	
4	
Total	

1. [30 points] Short Answers

a. Write down the IEEE 754 binary representation of the number -0.75 in single precision.
Answer:

1 01111110 10000000000000000000000

b. What decimal number does the bit pattern 1010 1101 0001 0000 0000 0000 0000 0000 represent if it is a floating point number in IEEE 754 standard?

Answer:
$-1.125_{ten} \times 2^{-37}$

c. Is there an overflow for an 8-bit machine when adding a two's complement integer x to a two's complement integer y as given below? Show your work. x = 0100 1011 and y = 0111 0100

Answer: Yes, there is an overflow, because of the negative value for sum of two positive integers.

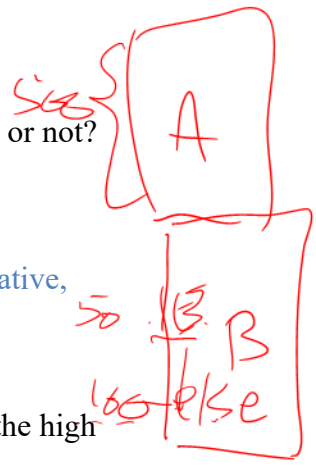
```
0111 0100
0100 1011     (+
-------------------
1011 1111
```

Else

d. In linking stage, does the label "else" in the following code require relocation or not?

```
BEQ  else
```

Solution: No relocation is required, because reference to label "else" is pc-relative, i.e., does not need the absolution address.



e. Is the recursion tail recursive? If not, convert it to a tailed recursion (only at the high level code, not the assembly code.)

```
int SquareSum(int n) {
     if (n == 1) return 1;
     return n*n + SquareSum(n - 1);
}
```

Solution: No, because after recursive call there is extra computation to do.

```
int SquareSum1(int n, int q) {
     if (n == 1) {
               return q +1n;
     }
     return SquareSum1(n – 1, n*n + q);
}
```

Call SquareSum1 with q=0

2. [25 points] Below is a function that inserts a node into an linked list.

```
void insert(node** root, node* a_node) {

  if((a_node)->data <= (*root)->data) {
        (a_node)->next = *root;
     *root = a_node;
  } else if((*root)->next != NULL) {
    if ((a_node)->data <= ((*root)->next)->data){
      (a_node)->next = (*root)->next;
      (*root)->next = a_node;
    } else {
      insert(&(*root)->next, a_node);
    }
  } else {
     (*root)->next = a_node;
  }

}
```

Translate the function into ARM assembly code. You can assume that the two arguments are put in r0 and r1 respectively by the caller.

A node in the linked list is structured as follows

data	Pointer to the next node

```
Insert: @ insert a new link (pointer in r1) to root (pointer to pointer in r0)
        sub sp, sp, #8
        str lr, [sp]
        ldr r2, [r0]        @ r2 has pointer to the root
        ldr r3, [r2]        @ r3 has data in root node
        ldr r4, [r1]        @ r4 has data in new node
        cmp r4, r3
        bgt Moveback
        str r2, [r1, #4]    @ new_node -> root
        str r1, [r0]        @ update new node as root
        add sp, sp, #8
        mov pc, lr
    Moveback:
        @ check if root is the last node
        ldr r3, [r2, #4]  @ r3 has pointer to node after root
        cmp r3, #0          @ check to see if r3 is null
        bne hasMore1
        str r1, [r2, #4]  @ root -> new_node
        add sp, sp, #8
        mov pc, lr
    hasMore1:
```

```
        ldr r5, [r3] @ r5 has the data of the node after the root
        cmp r4, r5
        bgt hasMore2
        str r3, [r1, #4]
        str r1, [r2, #4]
        add sp, sp, #8
        mov pc, lr
hasMore2:
        str r2, [sp, #4]  @ save the pointer to root
        str r3, [r0]          @ put pointer to node after root into [r0]
        BL Insert
        ldr lr, [sp]
        ldr r2, [sp, #4] @ retrieve the saved pointer to root
        str r2, [r0]         @ put it in [r0]
        add sp, sp, #8
        mov pc, lr
```

3. [20 points]

```
address                .text
0x0004 0000    main:ldr  r0,  =x
0x0004 0004         bl   foo
0x0004 0008         mov  pc, lr
0x0004 000c    foo: mov  r5, #0
0x0004 0010         cmp  r0, #0
0x0004 0014         blt  L1
0x0004 0018         add  r5, r5, #1
0x0004 001c         sub  r0, r0, #1
0x0004 0020         b    foo
0x0004 0024    L1:  mov  r0, r5
0x0004 0028         mov  pc, lr

                   .data
0x1000 0000                x: .word 10
```

For the above ARM assembly program, answer the following questions.

a) Show the address next to each instruction according to the ARM conventions for memory allocation.
Answer: see above.

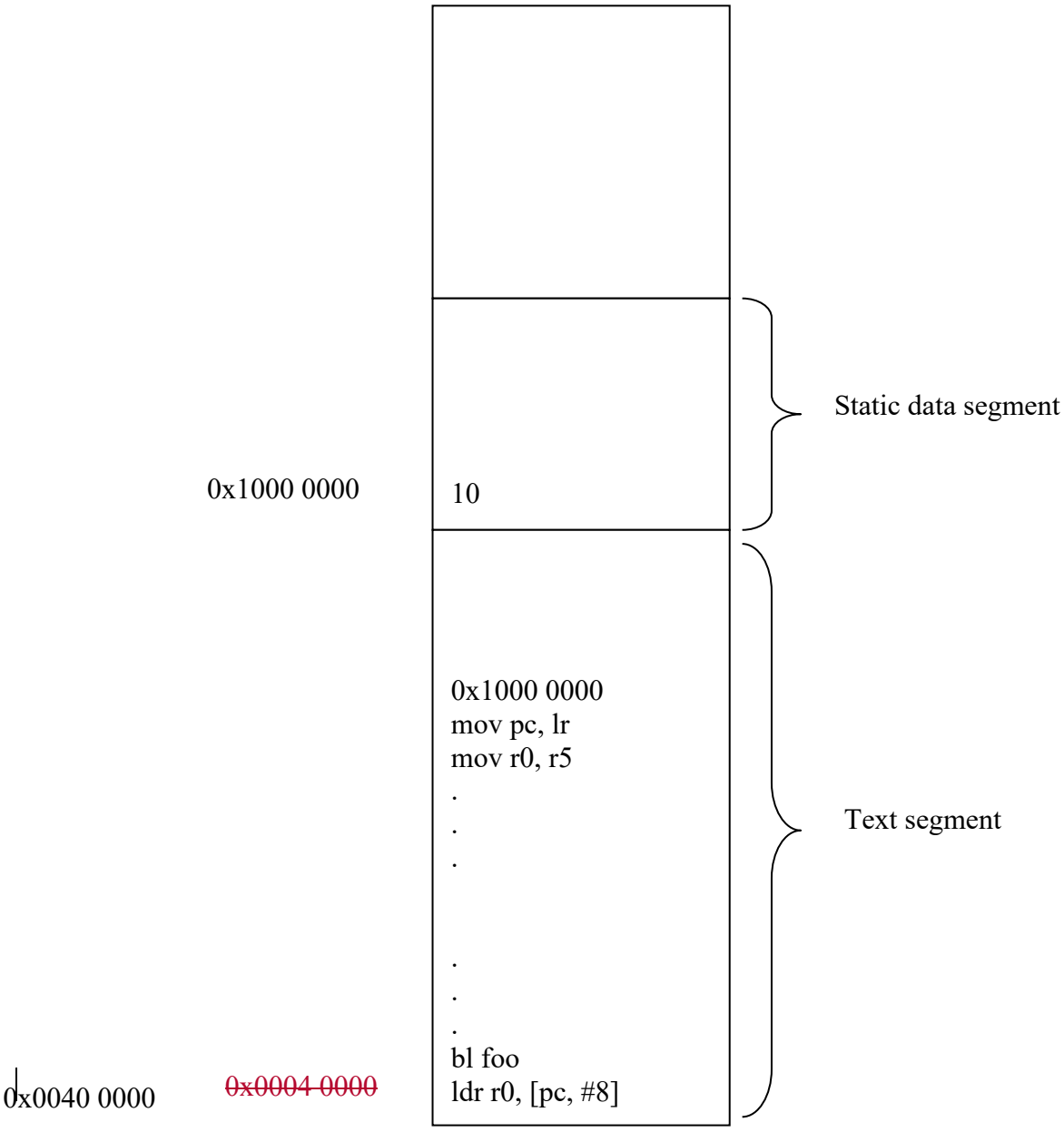
b) Draw the symbol table listing the labels and their addresses
Answer:

Symbol	address
main	~~0x00040000~~ 0x00400000
foo	~~0x0004000c~~ 0x0040000C
L1	~~0x00040024~~ 0x00400024
x	0x10000000

c) Convert "blt L1" into machine code (written in hex)

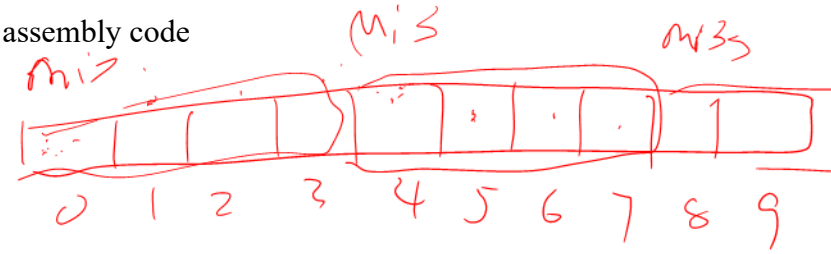
Answer: machine code: BA000002
See slides 6-8 in lecture notes "assembler_linker_loader.pdf"

d) Sketch a memory map showing where data and instructions are stored.


0x1000 0000
10
Static data segment
0x1000 0000
mov pc, lr
mov r0, r5
.
.
.
.
.
.
bl foo
ldr r0, [pc, #8]
Text segment
0x0040 0000
0x0004 0000

4. [25 points] Consider the following ARM assembly code

```
        mov   r3, #0
        mov   r7, #10
Loop:   ldr   r1, [r2, #0]
        mul   r1, r1, r4
        add   r2, r2, #4
        add   r3, r3, #1
        cmp   r7, r3
        bne   Loop
```



a) Compute the number of cycles needed for each loop iteration. Assume CPI is 1 for data processing, 5 for data transfer, and 2 for branching.

Answer: 5+1+1+1+1+2 = 11

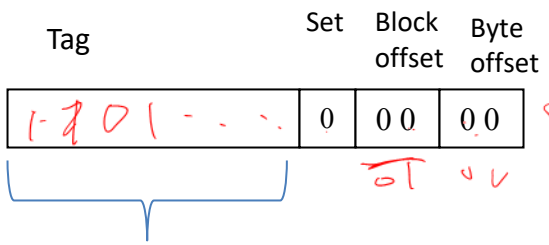
b) What is the average CPI for the above code?

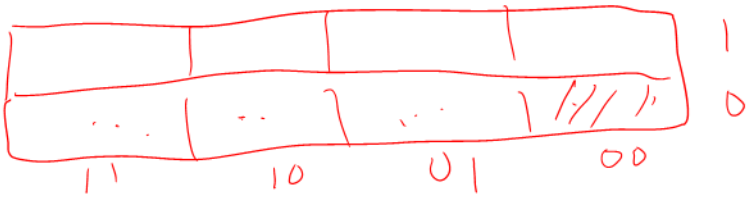
Answer: total instruction count: 2 + 6 x 10 = 62
Total number of cycles = 2 + 11 x 10 = 112
Average CPI = 112/62 = 1.81

c) Assume the clock cycle is 200ps, what is the total CPU time of running the above code.

Answer: CPU time = IC x CPI x CC = 62 x 1.81 x 200ps = 22,444ps

d) What type of locality does this code have for accessing the data in memory? If the instruction **ldr** can load four adjacent words into the cache, what is the miss rate? Note that the cache has two sets, each set has a block of 4 words, with memory mapping as shown below, where the values are only placeholder to indicate the numbers of bits in each field.





Answer: spatial locality.
Miss rate = 3/10