Instruction Set Architecture (ISA): Sample Problems and Solutions
1) Consider the following block of SPIM code. The text segment starts at 0x00400000 and the data segment starts at 0x10010000.

.data

label: .word 8, 16, 32, 64
L1: .byte 64, 32
.text
global main
main: la $a0, label # pass array base address
    li $a1, 16 # pass array count
    jal func # function call

    li $v0, 10 # terminate program
    syscall
func: move $v0, $a0 # get array base address
    move $v1, $a1 # get array count
    add $v1, $v1, $v0 # array ending address
    move $t1, $0 # initialize loop sum
loop: lw $s6, 0($v0) # load first value
    add $t1, $t1, $s6 # update sum
    addi $v0, $v0, 4 # update pointer
    slt $t0, $v0, $v1 # check for termination
    bne $t0, $0, loop # onto next element
    move $v0, $t1 # pass sum back to main
    jr $ra

(a) What are the values of main, func, and loop? Clearly state which instructions, if any, have to be translated into more than one native instruction.

i. main _______0x00400000__________

ii. func _______0x00400014__________

iii. loop _______0x00400024__________

These answers recognizes that the la instruction can be (in this instance) translated into one native instruction. If you assume it is translated into two native instructions, then ii) and iii) are increased by 0x4.
(b) The second time through the loop, what are the values in registers $ra$, and $t0$.

iv.  $ra$ ___________0x0040000c______

v.  $t0$ ___________0x00000001______

The answer to i) depends on the answer to part a).

(c) If this is a Big Endian machine what value is returned in register $t0$ for the following instruction.

vi.  lw $t0, L1($0) ___________0x40200000__

This answer assumes the remaining bytes in the word are initialized to 0x00. In general, you cannot really say.

(d) The following is a block of encoded SPIM code. Decode the sequence to produce the original MIPS instructions.

0043402a  _slt $8, $2, $3___

1500fffc  _bne $8, $0, -4___

For the second instruction you might choose to indicate the offset in bytes (-16). Although the encoding captures the offset in words (-4).

(e) Imagine you have doubled the number of general-purpose registers in the processor but still need to encode instructions in 32-bits. What is the impact on the encoding of the beq and bne instructions in terms of the range of the target addresses?

Since the number of registers has doubled you need one more bit to address each register – 6 bits instead of 5. The branch instructions have to specify two registers. Therefore, if we must remain committed to 32-bit encodings the immediate field must be reduced by 2 bits, leaving us with a 14 bit immediate field. The target range has now been reduced to (-2^{13} to +2^{13}-1) bytes offset from PC+4.
2) Consider the execution of the following block of SPIM code. The text segment starts at 0x00400000 and that the data segment starts at 0x10010000.

```
.data
    .word 0x32, 44, Top, 0x33
str:    .asciiz “Top”
    .align 8
end:    .space 16
.text
    .globl main
main:   li $t3, 4
        la $t0, start
        la $t1, end
Top:    lw $t5, 0($t0)
        sw $t5, 0($t1)
        addi $t0, $t0, 4
        addi $t1, $t1, 4
        addi $t3, $t3, -1
        bne $t3, $zero, Top
exit:   li $v0
        syscall
```

(a) Show the addresses and corresponding contents of the locations in the data segment that are loaded by the above SPIM code. Clearly associate addresses and corresponding contents.

<table>
<thead>
<tr>
<th>Data Segment Addresses</th>
<th>Data Segment Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10010000</td>
<td>0x00000032</td>
</tr>
<tr>
<td>0x10010004</td>
<td>0x0000002c</td>
</tr>
<tr>
<td>0x10010008</td>
<td>0x00400010</td>
</tr>
<tr>
<td>0x1001000c</td>
<td>0x00000033</td>
</tr>
<tr>
<td>0x10010010</td>
<td>0x00706f54</td>
</tr>
<tr>
<td>0x10010014</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x10010018</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x1001001c</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x10010020</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x10010024</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

The value in 0x10010008 depends on how the preceding instructions are encoded. “la $t1, end” is encoded into two instructions while “la $t0, start” is encoded into one instruction. As long as your assumptions are consistent you should not lose points. Everything else is 0x00000000 up to and beyond 0x10010100 (this is the point of the .align 8 directive)

(b) What are the values of the following.

- exit: 0x00400028
- Top: 0x00400010
- addi $t3, $t3, -1 (encoding): 0x216bffff
(c) Provide a set of data directives that will perform the memory allocation implied by the following high level language declarations.

- character My_Char;
  integer Limit;

  .data
  .byte 0x00
  .align 2
  .word 0x0

- integer array[10];
  .data
  .space 40

** Note there are many solutions *

(d) Given the following sequences, what is the range of feasible addresses for the label target on the MIPS architecture?

```
start:       beq$1, $2, target
            add$4, $5, $6

            ...

target:     sub$6, $4, $8
```

\[ \text{start + 8} \leq \text{target} =< \text{start+4 +}2^{15}-1 \text{ (words)} \]

(e) What is the difference between the \text{j} and \text{jal} instructions?

Both instructions work identically with regard to the program control flow, i.e., which instruction is executed next and how the address fields of the instruction are encoded. The \text{jal} instruction also causes the value of PC+4 to be stored in register $31.
3) Consider the following MIPs code taken from some program. We wish to turn this piece of code into a procedure. Note that no procedure call conventions have been followed and the procedure must be re-written to follow the MIPs procedure call conventions.

```
procA: lw $12, 0($8) sw $12, 0($15) addi $8, $8, 4 addi $15, $15, 4 addi $16, $16, -1 bne $16, $0, procA jr $31
```

a) What registers will have to be saved on the stack by the procedure if the MIPS register procedure call convention is to be followed and you do not modify the code shown above?

$s0, $fp (assuming we use the frame pointer). $ra does not have to saved since no other procedure/function is called.

b) Which registers are used to pass parameters to this procedure? How many registers do you think will be required for parameter passing?

The values of $8, $15, and $16 would be required in the procedure and are passed using registers $a0, $a1, and $a2.

c) If the procedure is stored in memory starting at location 0x00400020, what will be the encoding of the jal procA instruction that is in the calling program?

0x0c100008
4) Consider the execution of the following block of SPIM code on a multicycle datapath. The text segment starts at 0x00400000 and that the data segment starts at 0x10010000. Assume immediate instructions take 4 cycles.

```assembly
.data
start: .word 21, 22, 23, 24
str: .ascii "CmpE"
.align 4
.word 24, 0x77
.text
.globl main
main: li $t3, 4
    lui $t0, 0x1001
    lui $t1, 0x1002
move: lw $t5, 0($t0)
    sw $t5, 0($t1)
    addiu $t0, $t0, 4
    addiu $t1, $t1, 4
    addi $t3, $t3, -1
end: bne $t3, $zero, move
done:
```

a) Show the addresses and corresponding contents of the locations in the data segment that are loaded by the above SPIM code. Clearly associate addresses and corresponding contents.

<table>
<thead>
<tr>
<th>Data Segment Addresses</th>
<th>Data Segment Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10010000</td>
<td>0x00000015</td>
</tr>
<tr>
<td>0x10010004</td>
<td>0x00000016</td>
</tr>
<tr>
<td>0x10010008</td>
<td>0x00000017</td>
</tr>
<tr>
<td>0x1001000c</td>
<td>0x00000018</td>
</tr>
<tr>
<td>0x10010010</td>
<td>0x45706d43</td>
</tr>
<tr>
<td>0x10010014</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x10010018</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x1001001c</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x10010020</td>
<td>0x00000018</td>
</tr>
<tr>
<td>0x10010024</td>
<td>0x00000077</td>
</tr>
</tbody>
</table>
b) Show a sequence of SPIM instructions that will branch to the label loop if the contents of register $t0$ is 13.

```
addi $t1, $t0, -13
beq $t1, $zero, loop
```

c) What are the values of `end`, `done`, `start`, and `main`?

```
start: 0x10010000
main: 0x00400000
end: 0x00400020
done: 0x00400024
```

d) What does the following bit pattern represent?

10101101 00101101 00000000 00000000. Provide your answer in the form indicated.

A MIPS instruction (show in symbolic code) 

```
    _sw $t5, 0($t1)___________
```
5) Answer the following questions with respect to the MIPS program shown below. Assume that the data segment starts at 0x10010000 and the text segment starts at 0x00400000.

```
L1: .word 0x32, 104
.asciiz "Test 1"
.align 2
Blank: .word
L2: .space 64
.main: li $t0, 4
       move $t2, $zero
       li $a0, 1
       loop: jal solo
             addi $t0, $t0, -1
             addi $a0, $a0, 1
             add $t2, $t2, $v0
             slt $t1, $t0, $zero
             bne $t1, $zero, loop
             li $v0, 10
             syscall
```

a) How much space in bytes does the program occupy in the data segment and text segment?

   Data segment (21 words, 84 bytes) - remember must also count reserved space

   Text segment (11 words, 44 bytes)

b) With respect to the above program,
   • what are the values of the following?

   L2 ___________0x10010014___________

   loop ___________0x0040000c__________

   memory location 0x100100c 0x0003120
What information would be stored in the symbol table for the above program? Be specific about the above program. Do not provide a generic answer.

Upon completion of compilation, the label solo and its address (relative to the start of this module). This is information to be used by the linker to find and link the module corresponding to solo.

c) What is the encoding of the following instructions?

\[
\text{bne } \$t1, \$zero, \text{ loop} \\
\text{add } \$t0, \$t0, \$v0
\]

\[
\text{0x1520} \quad \text{ff} \quad \text{fa} \\
\text{0x01024020}
\]

d) Which of the instructions in the above program, if any, must be identified as requiring relocation information?

The jal instruction since this is the only instruction that relies on absolute addresses.

e) Suppose the procedure solo is independently compiled as a separate module. When it is linked with the main program the first instruction in solo is placed in memory at address 0x0040100c. Provide the hexadecimal encoding of the jal solo instruction.

\[0x0c100403\]
Answer the following questions with respect to the MIPS program shown below. For the purpose of this test, assume that each instruction can be stored in one word, i.e., do not worry about pseudo instructions! Further, assume the data segment starts at 0x1001000 and that the text segment starts are 0x04000000.

```mips
.data
.word 24, 0x16
L2: .byte, 77,66,55,44
.text
move $t0,$0
loop: mul $t8, $t0, 4
   add $t1, $a0, $t8
   sw $0, 0($t1)
   addi $t0, $t0, 1
   slt $t7, $t0, $a1
   bne $t7, $0, loop
```

a) What are the values of the labels L2 and Loop?

```plaintext
L2 __0x10010008________
```

```plaintext
loop __0x04000004________
```

b) Provide the hexadecimal encodings of the following instructions?

```plaintext
bne $t7, $0, loop __0x15e0fffa_______________
add $t1, $a0, $t8 __0x00984820________________
```

c) Show the contents of the data segment. Show both addresses and values at those addresses.

```plaintext
0x10010000 0x18
0x10010004 0x16
0x10010008 0x2e37424d
```

d) The instruction BLE (Branch on less than or equal too) is not a native instruction. Show the SPIM code that could be used to implement this test.

```plaintext
ble $t2, $t3, loop → slt $t1, $t3, $t2
                 beq $t1, $0 loop
```
a) Write the values of the words stored at the following memory locations. Provide your answer in hexadecimal notation.

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>_0x10010008</td>
<td>__78452040</td>
</tr>
<tr>
<td>_0x0040000c</td>
<td>__8fbe0000</td>
</tr>
<tr>
<td>_0x00400000</td>
<td>__0c100007</td>
</tr>
</tbody>
</table>

b) Consider the process of assembly of the above program.
   • What would the symbol table contain if the main program was independently compiled?

All of the labels that the linker needs to know about: push and pop and main (for programs where the loader creates code that jumps to main). We do not need to store labels from the data segment.
• Assuming that the memory map is fixed, what relocation information would be recorded if any?

The jal instructions generate absolute addresses and must have relocation information stored to permit placement of the program starting elsewhere in memory.

c) What are the values of the following labels?

```
ret1  ___0x00400018________
push ___0x0040001c________
```

d) Assume that the procedures push and pop were assembled in a distinct file and linked with the main program. Further assume that the procedures were placed in memory starting at location 0x00400040. Provide the encoding of the ‘jal pop’ instruction.

```
0x0c100010
```
8) Answer the following questions with respect to the MIPS program shown below. Assume that the data segment starts at 0x10010000 and the text segment starts at 0x00400000.

```
.data
.first: .word 0x21, 32
.byte 4, 3
.align 2
str: .asciiz "Test"

.text
main: li $4, 4
loop: jal func
      slt $7, $0, $4
      bne $7, $0, loop
      li $v0, 10
end: syscall
func: addi $4, $4, -1
      jr $31
```

a) What function does the last two instructions in the main program realize?

Program Exit. Ref SPIM documentation

b) With respect to the above program,
   • what are the values of the following symbols?

   end __0x00400014___________
   loop __0x04000004___________
   func __0x04000018___________
   str __0x1001000c___________

   • What is the total number of bytes required for this program, both data segment and text segment storage?
     data segment - 16
     text segment - 32
c) What is the encoding of the following instructions?
   bne $7, $0, loop  \_0x14e0fffd\_\_\_\_\_\_\_
   jal func  \_0x0c10006\_\_\_\_\_\_
   addi $4, $4, -1  \_0x2084\_\_\_\_\_\_

d) Using only native instructions, how can you implement the following test: branch-on-greater-than-or-equal-to-5?

   Example:
   \text{slt} \ $t2, \ $t0, \ $t1
   \text{bne} \ $t2, \ $0, \ False
   j True

   Where True and False are the labels of the instructions that correspond to the blocks of code to be executed depending on the outcome of the test.
9) Consider the execution of the following block of SPIM code on a multicycle datapath. Assume that the text segment starts at 0x00400000 and that the data segment starts at 0x10010000. Assume that registers $7, $8 and $9 have initial values 16, 0x10010020, and 0x10020020 respectively.

```plaintext
.data
str: .asciiz “Start”
.align 2
.word 24, 0x6666
.text
move: lw $12, 0($8)
   sw $12, 0($9)
   addiu $8, $8, 4
   addiu $9, $9, 4
   addi $7, $7, -1
end:  bne $7, $0, move
```

a) Show the word addresses and corresponding contents of the locations in the data segment that are loaded by the above SPIM code. Clearly associate addresses and corresponding contents.

<table>
<thead>
<tr>
<th>Data Segment Addresses</th>
<th>Data Segment Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10010000</td>
<td>0x72617453</td>
</tr>
<tr>
<td>0x10010004</td>
<td>0x00000074</td>
</tr>
<tr>
<td>0x10010008</td>
<td>0x00000018</td>
</tr>
<tr>
<td>0x1001000c</td>
<td>0x00006666</td>
</tr>
</tbody>
</table>

b) Assume the exception/interrupt handler is stored at location 0x80000020. Provide the SPIM instruction(s) for transferring program control to begin executing instructions at this address. Do not worry about setting the return address.

```plaintext
lui   $1, 0x8000
addi $1, $1, 0x0020
jr    $1
```
10) Consider the execution of the following block of SPIM code on a multicycle datapath. Assume that the text segment starts at 0x00400000 and that the data segment starts at 0x10010000. Assume that registers $7, $8 and $9 have initial values 16, 0x10010020, and 0x10020020 respectively.

```
.data
str: .asciiz "Exams"
.align 4
.word 24, 0x6666
.text
move: lw $12, 0($8)
      sw $12, 0($9)
      addiu $8, $8, 4
      addiu $9, $9, 4
      addi $7, $7, -1
end:  bne $7, $0, move
```

a) Show the addresses and corresponding contents of the locations in the data segment that are loaded by the above SPIM code. Clearly associate addresses and corresponding contents.

<table>
<thead>
<tr>
<th>Data Segment Addresses</th>
<th>Data Segment Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10010000</td>
<td>0x6d617845</td>
</tr>
<tr>
<td>0x10010004</td>
<td>0x00000073</td>
</tr>
<tr>
<td>0x10010008</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x1001000c</td>
<td>0x00000000</td>
</tr>
<tr>
<td>0x10010010</td>
<td>0x00000018</td>
</tr>
<tr>
<td>0x10010014</td>
<td>0x00006666</td>
</tr>
</tbody>
</table>

b) What is the difference between signed and unsigned instructions and what is the motivation for making this distinction?

Signed instructions interpret operands as 2’s complement numbers. They also will set conditions such as overflow, negative, etc. Unsigned numbers will be treated as n bit binary numbers. They will typically not set conditions such as overflow. Certain classes of numbers naturally involve unsigned arithmetic, e.g., memory address computations. In order to detect these conditions in some instances and ignore them in others, MIPS uses two different types of instructions.
c) What are the values of end and str?

str: 0x10010000
end: 0x00400014

d) What does the following bit pattern represent?
00101001 10111010 00000000 00000000. Provide your answer in the form indicated.

A MIPS instruction (show in symbolic code) _slti $26, $13, 0_________
11) Answer the following questions with respect to the MIPS program shown below. Note that this simple program does not use the register saving conventions followed in class. Assume that each instruction is a native instruction and can be stored in one word! Further, assume the data segment starts at 0x10001000 and that the text segment starts are 0x00400000.

```
.data
    label: .word 8,16,32,64
    .byte 64, 32
.text
.globl main
main:   la $4, label
    li $5, 16
    jal func
    done

func:   move $2, $4
    move $3, $5
    add $3, $3, $2
    move $9, $0
loop:   lw $22, 0($2)
    add $9, $9, $22
    addi $2, $2, 4
    slt $8, $2, $3
    bne $8, $0, loop
    move $2, $9
    jr $31
```

a) What does the above program do?

Sum the elements stored starting at address label

b) State the values of the labels loop and main?

```
loop 0x00400020
main 0x00400000
```

c) what are the hexadecimal encodings of the following instructions?

```
bne $8, $0, loop 0x1500fffb
add $9, $9, $22 0x01364820
```
d) Identify the local and global labels in the program?

- main is a global label
- all other labels are local labels
12) What is an unresolved reference? How and when does it become resolved?

A reference to a label that is undefined. It is resolved by the linker which has access to all of the labels accessed by the programs being linked.

13) Write the SPIM instruction sequence that you could use to implement the branch-on-greater-than-or-equal-to-test. For example, how would implement “bge $7, $6, loop”.

```assembly
slt $1, $7, $6
beq $1, $0, loop
```
14) Consider the following isolated block of code. Note that some initialization code is missing.

```
.text
move: lw $12, 0($8)
    sw $12, 0($9)
addiu $8, $8, 4
addiu $9, $9, 4
addi $7, $7, -1
end:   bne $7, $0, move
```

a) If the text segment starts at 0x04000000, what is the value of the label `end`?

0x04000014

b) Show the hexadecimal encoding of the last instruction.

0x14e0ffffff

c) What does the following bit pattern represent?

00101100 01110000 11000000 00000000

A MIPS instruction (symbolic code!) 

\_sltiu $16, $3, 0xc000
15) Consider the following isolated block of SPIM code. Ignore initialization code. The data segment starts at 0x1001000 and the text segment starts at 0x00400000. Note that this block of code does not use registers saving conventions on a procedure call.

| start: | addi $2, $0, 85  
|        | addi $3, $0, 1  
|        | _jal mystery_  
| end:   | _j exit_  
| mystery: | add $4, $0, $0  
| loop:  | andi $5, $2, 1  
|        | _beq $5, $0, skip_  
|        | add $4, $4, $3  
| skip:  | _srl $2, $2, 1_  
|        | _bne $2, $0, loop_  
| exit:  | _jr $31_  
|        | _li $v0, 10_  
|        | _syscall_  

a) What are the values of the following labels?.

loop  ____0x00400014_____

skip ____0x00400020_____

b) What are the encodings of the following instructions?

addi $3, $0, 1  ____0x20030001___________

beq $5, $2, skip  ____0x10a00001___________

c) Let us suppose the procedure mystery is independently compiled and linked when the program is run. After linking the program is stored in memory and mystery has a value of 0x00400010. What is the encoding of the jal mystery instruction.

0x0c100004
d) Which of the following are valid MIPS instructions? If they are invalid you must state the correct form or clearly state why it is incorrect.

- \texttt{lw \$t0, \$t1($t3)}

  The offset must be a label or numeric constant. Examples of the correct form are:
  \texttt{lw \$t0, 4($t3) or lw \$t0, label2($t3)}

- \texttt{sliu \$t1, \$t2, 0x44}

  This is correct. Immediates can be specified in hexadecimal notation or base 10 notations.

- \texttt{bne \$t1, label, loop}

  The arguments to the instruction must be registers, or one of the arguments can actually be an immediate. A correct form would be
  \texttt{bne \$t1, \$s3, loop}


e) Suppose I want to store the following information in the data segment in the order shown. Show a sequence of SPIM data directives that will correctly do so. The text string, each reserved word and the byte array should be labeled so that programs may reference them easily.

- the text string “Enter a SPIM instruction”
- reserve space for 4 words
- store an array of 16 bytes (pick your own values)

```
.data
str: .asciiz "Enter a SPIM instruction"
.align 2
L1: .space 4
L2: .space 4
L3: .space 4
L4: .space 4
array: .byte 1,2,3,4,5,6,7,8,9,11,22,33,44,55,66,77
```
f) In the preceding question (1(f)) what assembler directive would cause the byte array to start on a 64 byte boundary?

```
.align 6
```

does allocation to the next $2^6$ byte boundary

g) What does the program in question 1 do? You have to specify and state the function or operation performed.

Counts the number of bit set to 1 in $2$. 
16) (a) What is the difference between native instructions and pseudoinstructions?

Native instructions are implemented by the underlying hardware. Pseudoinstructions are translated into native instructions by the assembler. Pseudoinstructions provide more powerful instructions to the programmer without affecting the complexity of the hardware since they are not interpreted by the hardware.
17) Consider the function `func` from the SPIM program shown in question 1).

```
func:
    move $v0, $a0
    move $v1, $a1
    add $v1, $v1, $v0
    move $t1, $0

loop:    lw $s6, 0($v0)
    add $t1, $t1, $s6
    addi $v0, $v0, 4
    slt $t0, $v0, $v1
    bne $t0, $0, loop
    move $v0, $t1
    jr $ra
```

(a) Assume that `func` was independently compiled assuming a starting instruction address of 0x00000000. It is actually loaded into memory starting at location 0x00402000.

i. Which instructions in `func`, if any, have to be corrected? Explain.

None. There are no instructions, which employ absolute addresses.

ii. What would the symbol table contain after assembly is complete?

As written above, there will be no entries in the symbol table although during assembly, `loop` will be stored to resolve any references to it. In general, you might have expected that this function would be prefaced with a `.globl func` directive to export the label `func` so other programs can link with it. However, that is not the case with the code above.
18) Consider the following block of SPIM code. The text segment starts at 0x00400000 and the data segment starts at 0x10010000.

```
.data
first:    .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
          .word 1,0,1,0,1,0,1,0
last:     .word 0

.text
.globl main

main:    la $t0, first    #load start address of array
         la $t1, last     #load end address of the array
         addi $t1, $t1, 4 #point to first word after the array
         li $t2, 0        #initialize count using immediate
         add $t3, $zero, $zero #initialize sum using another approach
loop:   lw $t4, 0($t0)    #fetch array element
         add $t3, $t3, $t4 #update sum
         add $t0, $t0, 4 #point to next word
         addi $t2, $t2, 1 #increment count
         bne $t1, $t0, loop #if not done, start next iteration
         li $v0, 10
         syscall

a) Provide the hexadecimal encodings of the following

lw $t4, 0($t0)                     8d0c0000
bne $t1, $t0, loop                 1528ffffc
```
b. Is the above code relocatable? Explain.

Yes. None of the instructions depend on the absolute address of another instruction.

c. The following is the binary representation of a block of assembled SPIM code. Disassemble the program producing the original SPIM instructions. Use the opcode map at the end of this exam.

<table>
<thead>
<tr>
<th>Assembled Binary</th>
<th>MIPS Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x21080004</td>
<td>addi $8, $8, 4</td>
</tr>
<tr>
<td>0x2129ffff</td>
<td>addi $9, $9, -1</td>
</tr>
<tr>
<td>0x1520fffc</td>
<td>bne $9, $0, -4</td>
</tr>
</tbody>
</table>

d. Consider a jal instruction stored at address 0x00400028. Determine whether each of the following addresses can be a target of this instruction, i.e., the starting location of the procedure. You must justify your answer to receive credit.

i. 0x20020040
   The jal instruction provides the lower 28 bits of the target byte address. The upper 4 bits come from the PC to produce a full 32-bit address. For this instruction the upper four bits of its PC are 0x0. Therefore, the target produced from the jal instruction cannot have the value 0x2 in the upper four bits. The target is not in the same 256Mbyte segment as the jal instruction. Hence, this address cannot be a target of the jal instruction.

ii. 0x04040044
    Following the reasoning cited above, this address can be a target of the jal instruction.

e. What will be contained in the symbol table after assembly?

The symbol main which is declared as globl.
19) Consider the following piece of SPIM code. The text segment starts at 0x00400000 and the data segment starts at 0x10010000. Assume all registers and memory locations are initialized to 0x00000000.

```
.data
start: .word 0x21, 22,
str: .asciiz "CompE"
.align 3
.word 24, 0x77

.text
.globl main

main: li $t3, 4
     lui $t0, 0x1001
     lui $t1, 0x1002
move: lw $t5, 0($t0)
     sw $t5, 0($t1)
     addi $t0, $t0, 4
     addi $t1, $t1, 4
     addi $t3, $t3, -1
end:  bne $t3, $zero, move
done:
```

(a) How many total bytes of storage are taken up by this program. If you need to make any assumptions about program assembly, state them explicitly.

Number of Bytes = _____60____

The .align 3 moves the loading pointer to 0x10010010. Therefore the data segment takes 24 bytes (6 words). Each instruction can be encoded in one instruction and therefore the 9 instructions take 36 bytes.

(b) Show the addresses and corresponding contents of the first four word locations in the data segment that are loaded by the above SPIM code.

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10010000</td>
<td>0x00000021</td>
</tr>
<tr>
<td>0x10010004</td>
<td>0x00000016</td>
</tr>
<tr>
<td>0x10010008</td>
<td>0x45706d43</td>
</tr>
<tr>
<td>0x1001000c</td>
<td>0x000000</td>
</tr>
</tbody>
</table>
(c) What would the contents of register $t0 (in hexadecimal notation), after the
execution of the following instruction, assuming Big Endian storage format?

\[ \text{lw $t0, str($0)} \quad \_0x436d7045\quad \]

The byte value at the word boundary is the most significant byte of the 32-bit word

(d) Show the hexadecimal encoding of the following instruction

i. \text{bne $3, $zero, move } \quad \_0x1460fffb \quad \text{(offset from PC)}

ii. \text{lw $t5, 0($t0)} \quad \_0x8d0d0000

(e) The following is the binary representation of a block of assembled SPIM code. Disassemble the program producing the original SPIM instructions. Use the opcode map at the end of this exam.

<table>
<thead>
<tr>
<th>Assembled Binary</th>
<th>MIPS Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xafbe003c</td>
<td>sw $30, 60($29)</td>
</tr>
<tr>
<td>0x00082821</td>
<td>addu $5, $0, $8</td>
</tr>
</tbody>
</table>

(f) Consider a program that references independently compiled procedures. What is the difference between static and dynamic linking? Be specific.

Static Linking: All program modules are linked at link/load time. All unresolved references are resolved and all non-relocatable instructions are patched. The size of the executable is large since all references must be resolved, i.e., all potential callers must be linked even of they will not be called during execution.

Dynamic Linking: Unresolved references are resolved at runtime. At compile/assembly time stub code is inserted that effects an indirect jump to system software that loads the requested module and patches the indirect jump to the entry address of the module. The next time this label is referenced, the indirect jump will directly take place to the named module.