ECE 3056: Architecture, Concurrency and Energy of Computation

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 \text{8,16,32,64}
L1: .byte 64, 32
.text
.globl 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 \text{main}, \text{func}, and \text{loop}? Clearly state which instructions, if any, have to be translated into more than one native instruction.

i. \text{main} _______0x00400000__________

ii. \text{func} _______0x00400014__________

iii. \text{loop} _______0x00400024__________

These answers recognizes that the \text{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 \quad _______0x004000c_____

v. $t0 \quad _______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) \quad ___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.

\[
\begin{align*}
0043402a \quad \textit{slt} \; &\; 8, \; 2, \; 3 \\ 
1500ffffff \quad \textit{bne} \; &\; 8, \; 0, \; -4 \\
\end{align*}
\]

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 \texttt{beq} and \texttt{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 $(-2^{13} \text{ to } +2^{13}-1)$ 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
  start: .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 upto and beyond 0x10010100 (this is the point of the .align 8 directive).

(b) What are the values of the following.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>exit</td>
<td>0x00400028</td>
</tr>
<tr>
<td>Top</td>
<td>0x00400010</td>
</tr>
<tr>
<td>addi $t3, $t3, -1 (encoding)</td>
<td>0x216bffff</td>
</tr>
</tbody>
</table>
(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:

   target:  sub$6, $4, $8

   start + 8 <= target =< start + 4 + 2^{15} - 1 (words)

(e) What is the difference between the j and 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 jal instruction also causes the value of PC+4 to be stored in register $31.
3) The following is block of SPIM code to compute the sum of the elements of a diagonal in a matrix stored as shown. For the matrix shown below the sum in $t2 should end up being 
$1+12+23+34 = 70$. Fill in any missing pieces of code to complete the program. Use any 
additional registers as necessary to keep track of the number of loop iterations and any 
other information you require. You may complete your code on the next page as necessary. 
If does not have to fit on the spaces shown. You cannot change the loop body. Remember 	here are many distinct, correct, feasible solutions!

```spim
.data
row1: .word 1,2,3,4
row1: .word 11,12,13,14
row1: .word 21,22,23,24
row1: .word 31,32,33,34

.text
la $t0, row1
li $t6, 20  # load offset to next diagonal element (16+4)
li $t7, 4  # load counter for number of elements
loop:
    lw $t1, 0($t0)  # loop body that performs the sum
    add $t2, $t2, $t1
    add $t0, $t0, $t6
    addi $t7, $t7, -1
    bne $t7, $0, loop  # check if all elements have been accessed
li $v0, 10  # exit the program
syscall
```
4) 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
5) 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.

```
.data
    .word 21, 22, 23, 24
str: .asciiz “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>0x00000000</td>
</tr>
<tr>
<td>0x10010024</td>
<td>0x00000000</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 assuming that it is one of the following.

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

A single precision IEEE 754 floating point number (show actual value in scientific notation)

```
-1.0101101 x 2^-37
```

A MIPS instruction (show in symbolic code)

```
_sw $t5, 0($t1)
```
6) 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.

```
.data
start: .word 21, 22, 23, 24
str: .asciiz “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 a sequence of SPIM instructions to implement the following: \( x = a[i*4] \), where \( a[i] \) is the access of element \( i \) from array \( a[] \). Use whatever register allocation you need.

```
.text
add $t5, $t0, $t0 # $t0 contains the value of i
add $t5, $t5, $t5 # two additions to compute i*4
add $t5, $t5, $t5 # two additions to compute i*8
add $t5, $t5, $t5 # two additions to compute i*16
add $t1, $t5, $t3 # base address contained in $t3
lw $t2, 0($t1) # value of x stored in $t2
```
b) What does the following bit pattern represent assuming that it is one of the following.
00111100 00001101 00010000 00000000. Provide your answer in the form indicated.

A single precision IEEE 754 floating point number
(show actual value in scientific notation) 1.0001101001 x 2^{-7}

A MIPS instruction (show in symbolic code) _lui $t5, 0x1000
7) 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 0x1001000c _______0x00003120________
• 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?

```
  bne $t1, $zero, loop   ___0x1520fffa_________
  add $t0, $t0, $v0      ___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
```
8) Consider the program shown on the previous question. You have probably noticed that it does not follow the MIPS register saving convention. Rewrite the code showing any changes that would be required by the MIPS register saving conventions.

```
.data
L1:
    .word 0x32, 104
    .asciiz "Test 1"
    .align 2
Blank:
L2:
    .word
.text
main:
    li $t0, 4
    move $t2, $zero
    li $a0, 1
loop:
    subu $sp, $sp, 32
    sw $t0, 0($sp)
    sw $t2, 8($sp)
    sw $a0, 12($sp)
    sw $t1, 16($sp)
    jal solo
    lw $t0, 0($sp)
    lw $t2, 8($sp)
    lw $t1, 16($sp)
    lw $a0, 12($sp)
    addu $sp, $sp, 32
    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) What is the difference between the jal and j instructions?

The execution of the jal instruction also causes the return address to be saved in $ra. The target address is computed in the same manner for both instructions.
9) 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.

```
.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?

L2 __0x10010008________
loop __0x04000004________

b) Provide the hexadecimal encodings of the following instructions?

```
bne $t7, $0, loop  __0x15e0ffe8________________
add $t1, $a0, $t8  __0x00984820________________
```

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

```
0x10010000  0x18
0x10010004  0x16
0x10010008  0x2e3742d
```

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.

```
ble $t2, $t3, loop →
    slt $t1, $t3, $t2
    beq $t1, $0 loop
```
10) This question deals with procedure calls. Consider the following SPIM program segment. 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 at 0x04000000.

```asm
.data
first: .word 0x33, 0x2
.byte 4, 3, 0, 0
str: .ascii"Final"

.text
main: li $t0, 4
      move $a0, $t0
loop jal func
      move $a0, $v0
      slt $a0, $0, $t9
      bne $t0, $0, loop
      li $v0, 10
      syscall
func: addi $a0, $a0, -1
      move $v0, $a0
      jr $ra
```

a) Consider the first time the procedure func is called. What is the contents of $ra when the program is executing func.

0x0400000c

b) Based on the MIPS register saving conventions which registers should be stored on the stack in the calling stack frame when `func` is called.

$\text{st0, st9, sa0}$

Note in this case $\text{sa0}$ is stored as a matter of convention since the calling procedure is not relying on the availability of the value that $\text{sa0}$ contained before the call.
11) Answer the following questions with respect to the MIPS program shown below. Assume that the data segment starts at 0x10010000 and that the text segment starts at 0x00400000. Assume that the instruction done is encoded in one word.

```
label: .word 24, 28
.byte 64, 32
.asciiz “Example Program”

main:    jal push
         jal pop
         done

pop:     lw $fp, 0($sp) lw $ra, 4($sp) addiu $sp, $sp, 32
ret1:    jr $ra
push:    subiu $sp, $sp, 32
         sw $fp, 0($sp)
         sw $ra, 4($sp)
ret2:    jr $ra
```

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
```
12) Provide the MIPS code to do the following.

• implement the greater-than-or-equal-to-zero test using only native instructions.

Example for a general test of $t0 greater-than-or-equal-to $t1

```mips
slt $t2, $t0, $t1
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.

• initialize a register to the value 0x33334444.

```mips
move $t0, $zero
lui $t0 0x3333
ori $t0, $t0, 0x4444
```
13) 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: .ascii "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` 
   - `jal func` 
   - `addi $4, $4, -1`

   
   __0x14e0fffd__________
   __0x0c100006__________
   __0x2084ffff__________


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

   Example:
   ```
   slt $t2, $t0, $t1
   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.
14) Consider the case where Procedure A calls procedure B, and procedure B calls procedure C. Procedures A and B both use registers $15, $16, $17 and $24, and procedure C uses registers $9, $10, and $11. The top of the stack initially points to the top of an active frame.

a) Show the contents of the stack when Procedure C is executing. Assume the MIPS register saving conventions as defined in the text. State, or clearly indicate the number of stack frames and which registers are saved by each procedure.

Note: I have not shown the procedures saving $ra and $fp which should be illustrated. Procedure C does not use any of the $sx registers and does not call other procedures. Therefore C does not need to save any of the registers.

b) Would callee save have been more efficient? Justify your answer.

No. Each procedure would have to save all the registers used in the procedure. This is clearly less efficient, e.g., look at procedure C.
15) 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 "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 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.

```
lui $1, 0x8000
addi $1, $1, 0x0020
jr   $1
```
c) If the above code was implemented as a procedure, following the MIPS register saving conventions, which registers if any would be saved by the procedure.

All of the registers which are used are temp registers and therefore are not saved by the procedure. Since no other procedure is called, there is no need to save the return address registers, $ra. Using the MIPS convention only the frame pointer would be saved in this example. There are sufficient registers.
16) 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.

```assembly
.data
  str: .ascii "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 assuming that it is one of the following. 00101001 10111010 00000000 00000000. Provide your answer in the form indicated.

A single precision IEEE 754 floating point number  
(show in scientific notation) 1.011101 x 2^{-44} 
A MIPS instruction (show in symbolic code) _slti $26, $13, 0
17) 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 0x1500fff8
add $9, $9, $22 0x01364820
```
d) The above program does not implement any register saving conventions on a procedure call. Assume we will use the MIPS register saving convention i.e., Caller/Callee Save. Ignore the storage of local variables on the stack. What will be the contents of the top frame on the stack when the procedure func is executing?

\$fp, \$22, since no other procedure is called \$ra is not stored

e) Identify the local and global labels in the program?

main is a global label
all other labels are local labels
18) 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.

19) 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”.

   slt $1, $7, $6
   beq $1, $0, loop
20) 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) Let us assume we wish to implement the above function as a procedure.
   • How many arguments would the procedure have and how would they be passed to this procedure from a calling program?
      Three passed in argument registers $a0-$a2

   • name the registers that would be stored on the stack after this procedure is invoked
     $7, $8, $9, $12 would have to be stored by the caller, if at all.
     $fp is the only register that is stored following the steps in the text.

c) Show the hexadecimal encoding of the last instruction.

0x14e0ffffa

d) What does the following bit pattern represent assuming that it is one of the following.

00101100 01110000 11000000 00000000

A single precision IEEE 754 floating point number (in scientific notation)  
_1.111000011 \times 2^{39} 

A MIPS instruction (symbolic code!)  
\_sltiu $16, $3, 0xc000 

21) 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.

- `lw $t0, $t1($t3)`

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

- `sltiu $t1, $t2, 0x44`

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

- `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
  - `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: .ascii “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
```

this moves allocation to the next $2^6$ byte boundary

g) What does the program in question 1 do? You have to specific and state the function or operation performed. No partial credit.

Counts the number of bit set to 1 in $2$. 
22) (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.

22 (b) Let us consider the program shown in the previous question. The procedure mystery does not appear to follow any of the SPIM procedure call conventions. If you were to rewrite mystery to follow the MIPS caller/callee saving convention, what registers would you expect to find in the stack frame for mystery while it is executing?

Based on our class discussions only $fp.

22 (c) Suppose that a 32-bit integer A is stored in memory, starting at the address given in register $t1, and that another 32-bit integer B is stored in the next sequential word after A. Write a MIPS procedure Swap that exchanges the values A and B in memory if A > B. The procedure should return to its caller when it is finished. Assume a purely caller-save convention for preserving registers is being used. Full credit is the program fits in the table below. Provide comments.

<table>
<thead>
<tr>
<th>label</th>
<th>instruction</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap:</td>
<td>lw $t0, 0($t1)</td>
<td>Load A</td>
</tr>
<tr>
<td></td>
<td>lw $t2, 4($t1)</td>
<td>Load B</td>
</tr>
<tr>
<td></td>
<td>slt $t3, $t2, $t0</td>
<td>Check if B&lt;A</td>
</tr>
<tr>
<td></td>
<td>beq $t3, $0, exit</td>
<td>If not true (result=0)exit</td>
</tr>
<tr>
<td></td>
<td>sw $t0, 4($t1)</td>
<td>Swap A and B</td>
</tr>
<tr>
<td></td>
<td>sw $t2, 0($t1)</td>
<td></td>
</tr>
<tr>
<td>exit:</td>
<td>jr $31</td>
<td>Return</td>
</tr>
</tbody>
</table>

Another solution would expect to find the address of A in $a0 and use this register instead of $t1. Since we are using caller-save and this procedure does not call anyone else, the return address does not need to be saved. With no saving requirements there is no stack frame that needs to be allocated/deallocated.
23) Consider the function `func` from the SPIM program shown in question 1).

(a) Show modifications just to the function `func`, if any, to conform to the MIPS procedure call convention. If none, state why.

```assembly
func:
  addi $sp, $sp, -4
  sw $s6, 0($sp)

  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

  lw $s6, 0($sp)
  addi $sp, $sp, -4
  jr $ra
```

This is a function/procedure so you should consider the callee save convention which requires that the $s registers be saved. As the writer of this procedure, you know that no other functions/procedures are being called and therefore you do not need to save $ra or any other registers (as a caller for example). Only $s6 needs to be saved and restored as per the MIPS calling convention.

(b) 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.
24) 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
  .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
    addi $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 1528fff8
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 (words)</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 comes 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.
25) 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.

```assembly
.data
  start: .word 0x21, 22,
  str: .asciiz "CmpE"
  .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?

\[ \text{lw} \; \text{t0}, \; \text{str}(\text{t0}) \quad _{0x436d7045} \]

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. \[ \text{bne} \; 3, \; \text{zero}, \; \text{move} \quad _{0x1460fff} \] (offset from PC)

ii. \[ \text{lw} \; \text{t5}, \; 0(\text{t0}) \quad _{0x8d0d0000} \]

(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 if 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.
Consider the 32-bit ALU design discussed in class and shown below. Note that it supports the \texttt{beq} instruction. Now suppose that we wished to similarly add hardware support for \texttt{ble} \texttt{$t0$, $t1$, loop} – the \textit{branch-on-less-than-or-equal-to} instruction. \textbf{Clearly} show i) the necessary changes to the ALU hardware, ii) list the corresponding values of the ALU control signals for this instruction, and iii) describe briefly (in a sentence or two) how the single cycle datapath would operate with these changes.

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
Instruction & Binvert & Operation \\ 
\hline
\texttt{ble} & 1 & 10 \\ 
\hline
\end{tabular}
\end{center}

The ALU is set to perform \texttt{$t0$-$t1$ (110)}. Add a 2-input OR gate to form the logical OR of the \texttt{Zero} signal and the MSB to produce a new \texttt{ble} signal.

At the datapath level, the branch address multiplexor is now driven by a logical OR condition. The branch is taken if i) it is a \texttt{beq} instruction and the \texttt{zero} signal = 1, or ii) it is a \texttt{ble} instruction and the \texttt{ble} signal = 1. The \texttt{ble} instruction can use the same opcode at \texttt{beq}.
Consider the 32-bit ALU design discussed in class and shown below. Now suppose that we wished to add hardware support for a new instruction - \texttt{bre} $\texttt{t0}$, $\texttt{t1}$ – the \textit{bit reverse} instruction. The contents of register $\texttt{t0}$ is obtained by reversing the order of bits in register $\texttt{t1}$. For example, the 4-bit number 1101 will become 1011 and the number 1100 would become 0011. Clearly i) describe the necessary changes to the ALU hardware showing the changes to the $i^{th}$ ALU bit position by modifying the figure below, and ii) list the corresponding values of the ALU control signals for this instruction. Provide a brief explanation.

![ALU Diagram]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Binvert</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{bre}</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

- Add an additional input to each multiplexor
- The new input of the multiplexor at bit position $i$ is connected to the $(31-i)^{th}$ bit of the input corresponding to $\texttt{t1}$, e.g., input $a$
- When ALU control is set to 100, the output will be the bit reversal of the input
- The value of \texttt{binvert} does not matter.
- As for the implementation, one option is to use the I-format with the destination register being $\texttt{rt}$.
- There are other solutions.