



## Developing a Program in SPIM

- By now, if you have been doing all the homework and class examples, you have begun to feel a lot more at home with the programming process in SPIM .
- However, the primary focus of ALL programming is loops, since loops do all the heavy work in a program. This is where you need additional experience.
- Once again, the only way to learn programming is to program. That includes programming loops.
- As a way to get additional experience, we will do this program together. At this time, turn on your laptops.



## Programming Problem

- Our joint program will take keyboard input for a decimal number up to eight digits (99,999,999) and convert the decimal number to hexadecimal.
- It will then print out the hex result on the SPIM simulated console.
- The program will run continually and convert multiple numbers, if desired.
- We will construct this program together, so that you can get some more practice in live coding.
- The program should be completed off-line if you do not finish by the end of class (and also to make an addition to the program discussed at the end of class).



## Thoughts on the Program

- **What does our decimal-to-hex program need to do?**
  1. **Input the number from the keyboard.**
  2. **Convert the decimal number to a hexadecimal number.**
  3. **Output the number in hex.**
  4. **Key consideration: SPIM does not have a way to output hex numbers, since any number we output with syscall 1 is automatically converted to decimal.**
  5. **Therefore we must convert each hex digit of the number to the corresponding ASCII character, since we can output ASCII characters in two ways:**
    - **Convert each hex digit to ASCII, and print out using syscall 11.**
    - **Convert all the digits, store in a string, and output using syscall 4.**



## Thoughts (2)

- **Some good news:**
  - All numbers in the computer are binary.
  - **Furthermore, these binary numbers are immediately partitionable into hex (as in MIPS simulated console display).**
  - **Then any number input into the MIPS computer can be easily separated into 4-bit nibbles, that represent hexadecimal digits.**
- **Our task is then reduced to:**
  - **Isolating each 4-bit nibble separately (as a single hex digit).**
  - **Converting each nibble to the equivalent ASCII character and printing the character.**



## Thoughts (3)

- **Other considerations:**
  - For simplicity, we will only use fixed-point, integral numbers, (we have not covered floating-point programming).
  - Remember: System call 5 will take any number of digits in an input. **However, it truncates after the decimal number input reaches the size of a 32-bit fixed-point representation (that is 0x ffff ffff, or about  $\pm 2,150,000,000$ ).**
  - **Worse yet, it truncates the most significant bits.**
  - **In order to forestall this problem we will limit inputs to eight (8) decimal digits. The biggest 8-digit decimal number is  $99,999,999 = 0x05f5e0ff$ , so we will always be able to represent the decimal number properly in the conversion.**



## Thoughts (4)

- **Our decimal-hex conversion routine is then:**
  1. **After keyboard input, store contents of \$v0 in another register (must reserve \$v0 for system calls). The number will already be the binary equivalent of the decimal number.**
  2. **One binary nibble (i.e., 4 bits) is equivalent to one hex digit. We therefore analyze the number nibble by nibble as a hex digit string.**
  3. **We convert each nibble to the equivalent ASCII character for that hex number, storing the ASCII character in a reserved byte string location, or printing character-by-character.**
  4. **When all eight nibbles (hex digits) of the 32-bit MIPS input data word are analyzed, the conversion is complete.**



## Thoughts (5)

- Since we want to output the converted hex number correctly, that means that we must print out the most significant digit first, proceeding to less significant digits in descending order (we read numbers left-to-right starting with the most significant digit).
- **This suggests that we analyze the hex number starting with the most significant digit, then the next-most-significant, and so on, so that the printing order will be correct.**
- **Due to the restriction of 8 decimal digits (99,999,999 max. size), the left-most digit will always be 0. If the input number is much smaller, more digits of the result will be 0.**
- **Leading 0's can easily be discarded by a preliminary testing loop until a non-zero number is found, then starting the analysis (we will want to keep 0's interior to the number).**



## Thoughts (6)


- How can we analyze the eight-digit hex number one digit at a time?
  - Use logical **AND** instruction to “mask off” or isolate each 4-bit hex number.
  - The “masked” result will be a 32-bit number with all 0’s except where the 4-bit mask preserved 4 bits.
  - To simplify the loop, the “mask” should always be in the same place.
  - We can use a shift or rotate instruction to get each succeeding 4-bit nibble positioned under our mask.



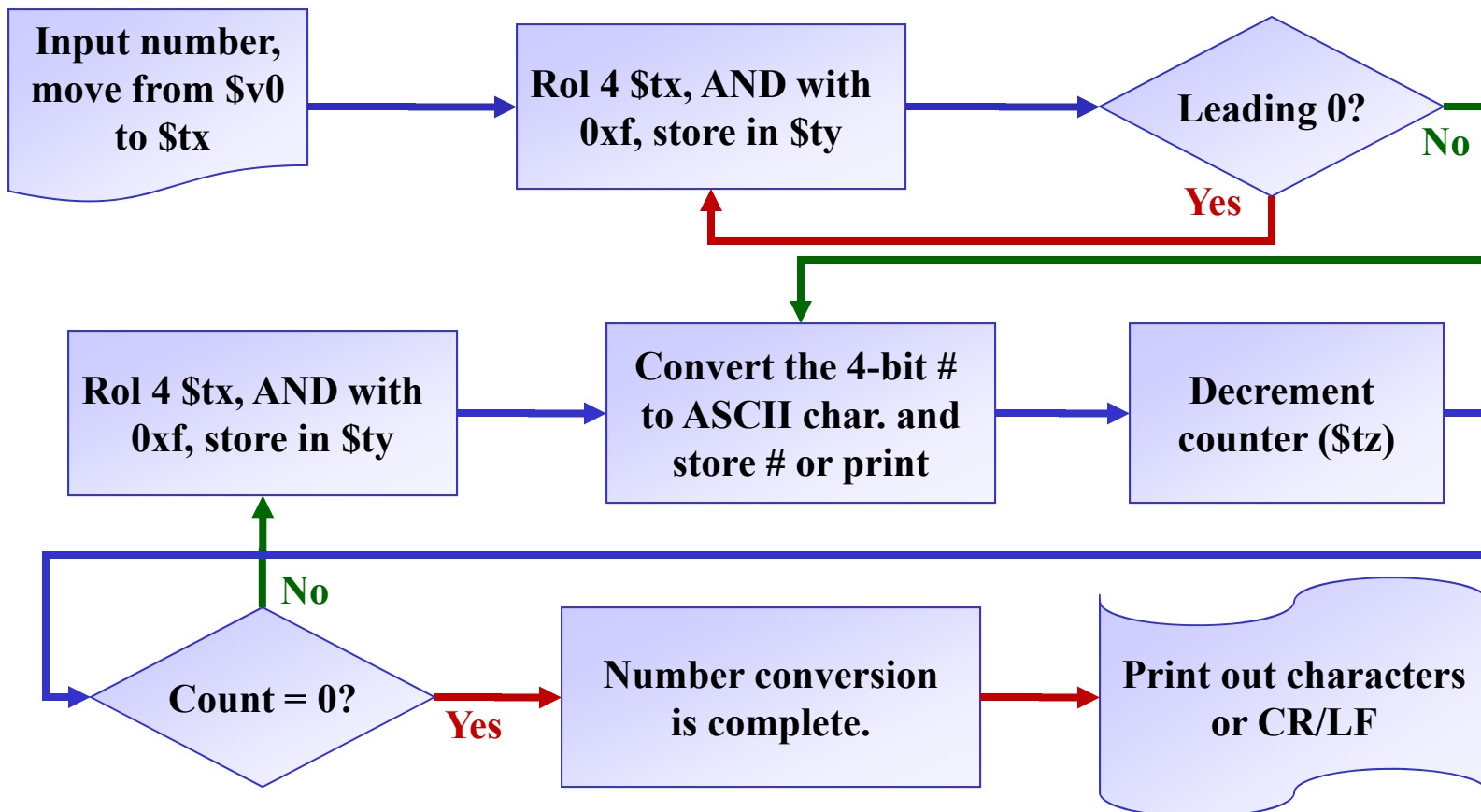
## Thoughts (7)

- Doing a rol 4 then an andi using a **0x 0000 000f** mask will result in a hex number (nibble) in the four LSB positions, and also allows analysis of the most-significant hex digit first.
- We put the number being analyzed in the four least significant bit locations. Then the four bits can be analyzed as the single hex digit that they represent (the other 28 bit positions = 0).
  - 0x 0000 d000** ← This is “d thousand,” NOT hex “d.”
  - 0x 0000 000d** ← This number is a hex “d.”
- After converting the digit to the equivalent ASCII character, we can either output using syscall 11 or store the character for later output using syscall 4. By analyzing the most-significant-digit first, our print routine will print out digits in the correct numeric order.

## Program Sequence

- **Now we can write down a the basic task sequence:**
    - **Input number from keyboard (immediately → binary [hex] form).**
    - **Move number to another register. Set up loop parameters.**
  - **Loop sequence:**
    - **Rotate 4 most significant (leftmost) bits to right-most 4 bit position.**
    - **Mask off those 4 bits and store in another register.**
    - **(Result is one hex digit [4-bits] in right 4 bit positions.)**
    - **Test to see if nibble = 0. Discard 0's until non-0 character found.**
    - **Convert binary nibbles to equivalent hex ASCII character.**
    - **Output using syscall 11 or store characters for later print-out.**
    - **Repeat for each 4-bit number until remaining digits are converted.**
    - **Print out digits (if not printed “on the fly”).**
- 

## Basic Flow Chart for Hex Digit



## Loop Mechanics

- With the basic program flow laid out, the “hex digit analysis loop” must be designed.
  - Since each 32-bit binary MIPS number is equivalently 8 hex digits, we must go through the loop 8 times (remembering that at least the first digit will always be a leading zero).
  - This means that we require a counter. This can be a register that we initially set to 0.
  - Each time we traverse the loop and analyze one ASCII character, we will add one to the register.
  - When the number in the register = 8, we will have gone through the loop 8 times, and the analysis will be complete.



## Loop Mechanics (2)

- The central work of the loop is simple:
  - Rotate the input number (being held in some register) four bits left. This transfers one digit to the LS position in the number.
  - “Mask” off by ANDing the number with 0xf and store it in another register. This represents one hex digit. Since we rotate left, we analyze the biggest hex digit first and subsequent digits in descending order (the order we need to print!).
  - Convert this digit to the ASCII code for the digit.
  - We can either store this digit for output later or output immediately with syscall 11.
  - As we are in the loop design phase, we have to make that choice NOW. I will make it by deciding that we will output each digit on-the-fly using syscall 11!



## Writing the Program

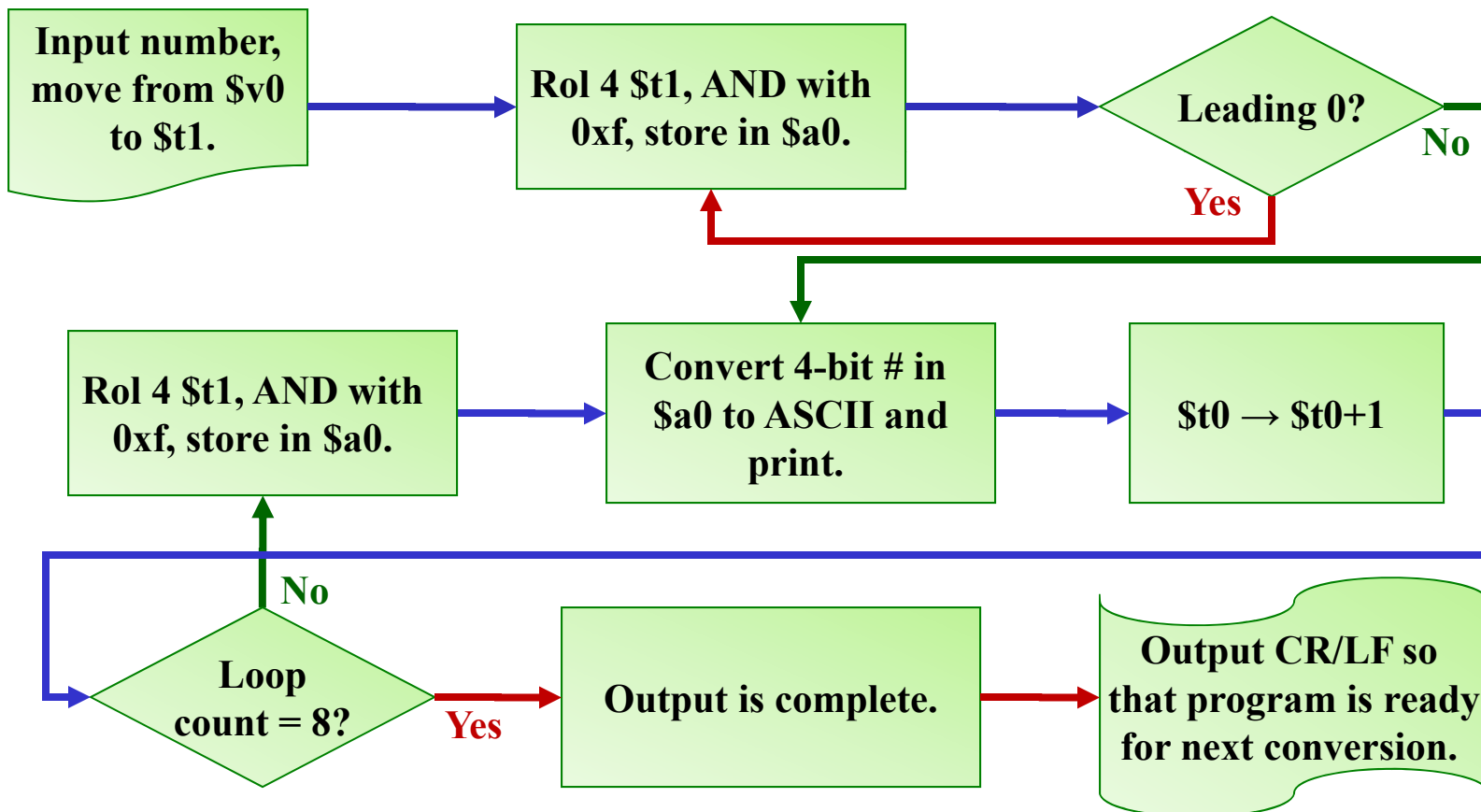
- We can start writing the program now, although we still have some decisions to make. Consider our data declarations:
  - Prompt command, to tell the user to input a number.
  - An answer announcement, preceding the numerical answer.
  - A request if another number conversion is desired.
- Data declarations:
  - prompt: `.ascii "Enter decimal number (8 digits, maximum): "`
  - answer: `.ascii "Hexadecimal number is: 0x"`
  - comp: `.ascii "Input another number (y/n)?"`



## Register Assignments

- To keep it simple, let's assign registers now, so we will be programming using a common set of designations.
- We can use mainly the t-registers with one exception.  
My suggestion:
  - Counter – Use \$t0 – This will keep track of “times through the loop.”
  - Register to hold hex digit (nibble) being analyzed – Use \$a0 (Why use this register?)
  - Register to hold original number input (transfer from \$v0) – Use \$t1
- Isn't it surprising how few registers we need?

## Updating the Flow Chart







## Doing the Programming

- **We can write several early sections of the program before we get to the data conversion algorithm, which we have yet to deal with:**
  - **The program header (basically a big comment).**
  - **The data declaration (which we have already looked at).**
  - **The pre-loop code.**
  - **The leading 0 elimination loop.**
- **Then we will consider the data conversion loop, and how it must really operate.**

## Program “Prolog”

```
# Lecture 16 Demo Program 1, Hex Number Conversion  
# Converts a Decimal Number to Its equivalent hex value and  
# prints out hexadecimal number to console.
```

```
#
```

```
# $t0 - Loop counter
```


```
# $a0 - Holds each hex digit as analyzed
```


```
# $t1 - Holds number input from keyboard
```


Program name  
and description


Definition of  
register roles


## Data Declarations


**Initial prompt to user** 

**.data**  **Data directive**

prompt: .ascii "Enter decimal number (8 digits, maximum): " 

answer: .ascii "Hexadecimal number is 0x"  **Answer leader**

comp: .ascii "Input another number(y/n)?"  **Program will do multiple conversions**

# End of file Lecture 16 Demo Program 1  **Final comment**

## Doing Pre-Loop Software

- What do we need to do before entering the loop that eliminates leading 0's?
  - Text declaration (“`.text`”)
  - Prompt to enter a number  $< 99,999,999$
  - Number input syscall
  - Transfer number out of `$v0`
  - Since we are outputting the answer digit by digit, here we should output the answer header (“Hexadecimal number is”)
  - Clear the counter (since we plan on program being able to do multiple decimal-hex conversions)



## Pre-Loop Software

```
main:  .text          # Text declaration
       la $a0,prompt  # Load print out of prompt sequence
       li $v0,4        # Output prompt to enter decimal number
       syscall
       li $v0,5        # Read integer to convert to hex
       syscall
       move $t1,$v0    # Get number out of $v0
       la $a0,answer   # Output answer header
       li $v0,4
       syscall
       move $t0,$0     # Clear counter (can do multiple conversions)
```



## Eliminating Leading Zeroes

- Remember, in evaluating digits, we start at the most significant digit first (do an rol 4 for each digit to get it to least significant position).
- The first digit will ALWAYS be 0, and possibly one or more additional digits, depending on the size of the input decimal number.
- Therefore, our first job, after inputting the number, is to test it and “throw it away” if it is a zero.
- Once we find a non-zero number, we go into the decimal-to-hex conversion routine.

## Eliminating Leading Zeroes (2)

- Things to be done:
  - Put 11 in \$v0 (will be doing several syscall 11's).
  - Label the start of the 0-elimination loop.
  - Do rol 4 on \$t1 and return to \$t1.
  - Mask off least significant 4 bits in \$t1 and → \$a0 (why put in \$a0?).
  - Test for 0 value.
  - If 0, get next digit as above.
  - What if some smart Alec inputs a “0” as the number?! Need to test for 8 successive 0 digits!
  - If digit ≠ 0, go to decimal → hex conversion routine.



## Zero Elimination Software

```
elim:  li $v0,11      # Load number of syscall (11) into $v0
      rol $t1,$t1,4   # Left rotate digit to right-most
                        # position
      and $a0,$t1,0xf # "Mask off" left-most digit
      bgtz $a0,num    # If a non-zero go to character
                        # conversion routine
      addi $t0,$t0,1  # Since character = 0, increment counter
      beq $t0,8,zero  # If 8 zeroes, loop done; go to print
      j elim          # Get next character
```



## Converting to Hex

- The big question: **How to convert a hex digit to the ASCII code for the character representing that digit?**
- Consider hexadecimal numbers 0-9 (0000-1001):
  - The ASCII code for 0 is 0x 30 (decimal 48).
  - The ASCII code for 1 is 0x 31 (decimal 49).
  - Etc., to 9, whose ASCII code is 0x 39 (decimal 57).
  - The ASCII code for hex numbers 0 to 9 (binary 0000 to 1001) is exactly decimal 48 (hex 30) greater than the number.
  - Therefore we can define the formula:  $n_{\text{ASCII}} = n + 48$  for a decimal number  $n$ .
- Any hex number 0 to 9 can be converted to the ASCII code for that number by adding 48.



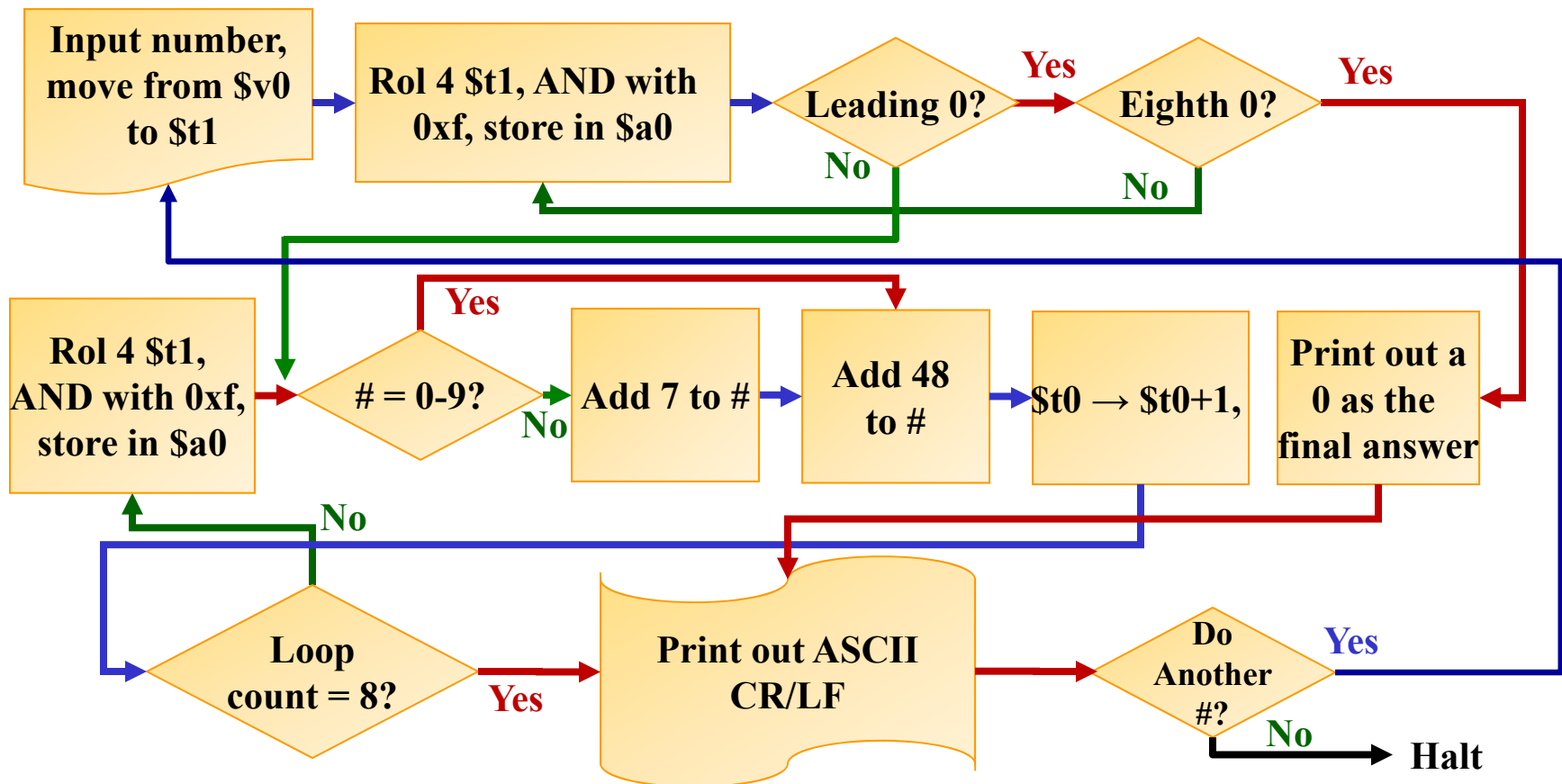
## Converting to Hex (2)

- What about hex numbers a-f?
  - Clearly, there must be some value to add to numbers a-f that results in the ASCII code for those numbers.
  - In fact, the ASCII code for A is 7 greater than the ASCII code for 9.
  - Since A-F are closer numerically to 0-9, I choose to use capital letters represent hex A-F.
  - The loop tests each hex number to see if it is 9 or less.
  - If 0x9 or less add 48 to get the ASCII code.
  - If hex A-F, add (48+7) to get the ASCII code.

## Final Version of Flow Chart

- Now we update the flow chart with the fine points of our decimal-to-hex routine.
  - Need to show zero-elimination loop in more detail.
  - Need to add test for a 0 being input to our flow chart.
  - Need to show the hex number-to-ASCII character in detail now that we understand the numeric conversion routine.
  - Need to add the routine that asks if another decimal-to-hex digit conversion is desired.

# Final Loop Flow Chart





## Elements of Data Conversion Loop

- We now have enough information to go ahead and write the data conversion loop.
- Once we get to a non-0 number, we convert to hex:
  - Left rotate the number in \$t0 4 bits, return to \$t0.
  - Mask off least 4 bits and store in \$a0 (why is that?).
  - Test to see if <10. (The first time through this loop we start here – why is that?)
  - If so, convert to number 0-9.
  - If  $\geq 10$ , convert to A-F.
  - Output the hex character using syscall (11 already in \$v0).
  - Increment and test counter.
  - Do loop again or stop if counter = 8.

## Loop Software (2)

```
loop:  rol $t1,$t1,4    # Left rotate left digit to right-most position
      and $a0,$t1,0xf  # "Mask off" left-most digit
num:   ble $a0,9,conv   # Go to conv routine directly if hex # 0-9
      add $a0,$a0,7     # Add 7 because hex number is a-f
conv:  add $a0,$a0,48    # Convert number to ASCII
      syscall          # Output ASCII representation of hex number
      addi $t0,$t0,1    # Increment counter
      blt $t0,8,loop    # If analysis not complete, do loop again
      j next           # Analysis complete; go to end routine
```

## Final Software Pieces

- **Finally, we need to do several miscellaneous things:**
  - **Output CR/LF (to create a new line).**
  - **Output question about doing another number conversion**
  - **Another CR/LF (new line in case we start over).**
  - **Input an ASCII character (syscall 12) to get user's answer to “do another?”**
  - **Go to start if do another number.**
  - **Halt if user input not = y.**



## Final Software Pieces (2)

```
next:  li $a0,0x0a      # Print out carriage return for neatness
        syscall
        la $a0,comp     # Convert another number to hex?
        li $v0,4
        syscall
        li $v0,11        #
        li $a0,0x0a      # Output CR/LF
        syscall         #
        li $v0,12        # Input answer (y = "Yes")
        syscall
        beq $v0,0x79,main # If yes, go back to start of program
        li $v0,10        # End program
        syscall
```



```

#   Lecture 16 Demo Program 1, Hex Number Conversion
#   Converts a Decimal Number to Its Equivalent Hex Value and
#   prints out hexadecimal number to console.
#
#   $t0 - Loop counter
#   $a0 - Holds each hex digit as analyzed
#   $t2 - Holds number input from keyboard

        .text
main:    la $a0,prompt # Load print out of prompt sequence
        li $v0,4          # Output prompt to enter decimal number
        syscall
        li $v0,5          # Read integer to convert to hex
        syscall
        move $t1,$v0
        la $a0,answer     # Output answer header
        li $v0,4
        syscall
        move $t0,$0       # Clear counter

elim:    li $v0,11         # Load number of syscall (11) into $v0
        rol $t1,$t1,4     # Left rotate left digit to right-most position

        and $a0,$t1,0xf   # "Mask off" left-most digit
        bgtz $a0,num      # If a non-zero go to character conversion routine
        addi $t0,$t0,1    # Since character = 0, increment counter
        beq $t0,8,zero    # If 8 zeroes, loop done; go to print
        j elim            # Get next character

loop:    rol $t1,$t1,4     # Left rotate left digit to right-most position
        and $a0,$t1,0xf   # "Mask off" left-most digit

num:     ble $a0,9,conv    # Go to conv routine directly if hex # 0-9
        add $a0,$a0,7     # Add 7 because hex number is a-f

```

```

conv:    add $a0,$a0,48      # Convert number to ASCII
        syscall            # Output ASCII representation of hex number
        addi $t0,$t0,1      # Increment counter
        blt $t0,8,loop     # If analysis not complete, do loop again
        j next             # Analysis complete; go to print routine

zero:    li $a0,0x30        # If number was 0, put 0 in print buffer
        syscall

next:    li $a0,0x0a        # Print out carriage return for neatness
        syscall
        la $a0,comp        # Convert another number to hex?
        li $v0,4
        syscall
        li $v0,11          #
        li $a0,0x0a        # Output CR/LF
        syscall           #
        li $v0,12          # Input answer (y = "Yes")
        syscall
        beq $v0,0x79,main  # If yes, go back to start of program
        li $v0,10          # End program
        syscall

        .data
prompt:  .asciiz "Enter decimal number (8 digits, maximum): "
answer:  .asciiz "Hexadecimal number is 0x"
comp:    .asciiz "Input another number(y/n)?"

```

**# End of file Lecture 16 Demo Program 1**



## Summary

- We have now defined, flow-charted, coded, and checked out another program together.
- In order to properly learn SPIM, you **MUST** do this process a number of times.
- As I am sure you have learned by now, the only way to improve your programming skill is to do more programming.
- My experience: Unless you design 10-15 programs, as you are required to do in your homework and labs and class exercises, you reduce your chances of passing EE 2310. And that means on your own.



## Program 2

- Program 2 is our programming exercise for this lecture.
  1. Load 0x84c732d6 into \$t0, using li.
  2. Construct a program to output each byte as a signed number. Text and data declarations are shown.
  3. Hints:
    - A byte loaded from memory is automatically sign-extended.
    - Rotate can be useful.
    - Use \$t1 as a counter.
  4. Output a CR/LF (0x0a) between each number.
  5. List the numbers that are output.

.text

main:

.data

data1: .space 1



## Practice Assignment

- **Do the following: Add lines of code to the class programming exercise program to (1) eliminate negative numbers, and (2) eliminate all positive numbers greater than 99,999,999.**