Homework Quiz (HW #2)  
SOLUTIONS  
September 19, 2001  
CS152 Computer Architecture and Engineering

This quiz covers one of the problems from homework #2.  
Good Luck!

<table>
<thead>
<tr>
<th>Your Name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SID Number:</td>
<td></td>
</tr>
<tr>
<td>Discussion Section:</td>
<td></td>
</tr>
</tbody>
</table>

Total: |                |
General Base Conversions

In homework problem 3.22, you were asked to write a MIPS assembly language program to convert an ASCII decimal string to 32-bit integer. For this quiz, construct a similar program that can convert a decimal string into an arbitrary-precision integer.

- The first argument (a0) contains a pointer to a null-terminated input string
- The Second argument (a1) contains a pointer to an array of words in memory for the result, least-significant word first.
- On exit, v0 will contain the number of 32-bit words in the result (at least 1), or an error.
- This procedure (call it “atoapi”) must adhere to all MIPS conventions

If any character in the string (before the final null character) is not a legal character for the specified base, then v0 should contain an error code of –1. Also, a zero-length string should return –1.

Examples:
If the input string is “0”, the output memory at [a1] will contain 1 word (0), and v0 = 1
If the input string is “452” the output memory at [a1] will contain 1 word (452), and v0 = 1

If the input string is “4294967298”, which is $2^{32} + 2$, then the output memory pointed at by a0 will contain a “2” for the first word (least-significant word) and a “1” for the second word. Further, v0 = 2.

If the input string is “6B7” the result register, should contain –1
If the input string is “” (i.e. first character is null), the result register should contain –1

The important portion of the ASCII character table is as follows (Note that values are in HEX notation):

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0”</td>
<td>0x30</td>
</tr>
<tr>
<td>“1”</td>
<td>0x31</td>
</tr>
<tr>
<td>“2”</td>
<td>0x32</td>
</tr>
<tr>
<td>“3”</td>
<td>0x33</td>
</tr>
<tr>
<td>“4”</td>
<td>0x34</td>
</tr>
<tr>
<td>“5”</td>
<td>0x35</td>
</tr>
<tr>
<td>“6”</td>
<td>0x36</td>
</tr>
<tr>
<td>“7”</td>
<td>0x37</td>
</tr>
<tr>
<td>“8”</td>
<td>0x38</td>
</tr>
<tr>
<td>“9”</td>
<td>0x39</td>
</tr>
</tbody>
</table>

The MIPS instruction set is included as an attachment. Write this routine in as few instructions as possible on the following page. Assume the virtual MIPS machine (no branch delay slots). Hint: start by writing a “times-ten” routing that takes the current number in your memory register (pointed at by a0 and of some length) and multiplies it by 10. Possibly extending the number of words as necessary.
This is a straightforward extension of the homework exercise. The trickiest part about this solution is the multi-precision arithmetic. The simplest solution is to create an inner loop (timestenplus) that handles the x = x*10+y, where x is the current result in memory and y is a new digit. We accumulate the length of the result in words in v0.

```assembly
atoapi: st $r0, 0($a1) ; Default: result = 0
        addi $v0, $r0, 1 ; one word result so far

ld $t7, 0($a0) ; Special check for empty string
beq $t7, $r0, error

nextdigit:
ld $t7, 0($a0) ; Get next digit
addi $a0, $a0, 1 ; Advance string pointer
bne $t7, $r0, cont ; Not end of string
jr $ra ; Finished – length in $v0

cont: addi $t7, $t7, -0x30 ; Convert from ASCII to digit
slti $t0, $t7, 0 ; Less than 0?
bne $t0, $r0, error ; Yup.  Bad digit
slti $t0, $t7, 10 ; Greater than 9?
beq $t0, $r0, timestenplus ; No – good digit in $t7

error: addi $v0, $r0, -1 ; Error return value
jr $ra

timestenplus:
add $t0, $r0, $r0 ; Current word count = 0
add $t1, $a1, $r0 ; Current word pointer = start
addi $t2, $r0, 10 ; Constant for multiplying

loop: lw $t3, 0($t1) ; Get next word from memory
mulu $t3, $t2 ; Multiply word by 10
mflo $t3 ; Low result of multiply
addu $t4, $t3, $t7 ; Add in new digit
sw $t4, 0($t1) ; Save result in memory

slt $t5, $t4, $t3 ; Check for carry from addu
mfhi $t3 ; High result of mult
add $t7, $t3, $t5 ; Add carry from addu

addi $t0, $t0, 1 ; Next word ID
addi $t1, $t1, 4 ; Next word pointer
bne $t0, $v0, loop ; Keep looping until through all digits

beq $t7, $r0, nextdigit ; Finished - no need to increase digits

st $t7, 0($t1) ; Add extra word
addi $v0, $v0, 1 ; Increase number of words by 1
j nextdigit
```