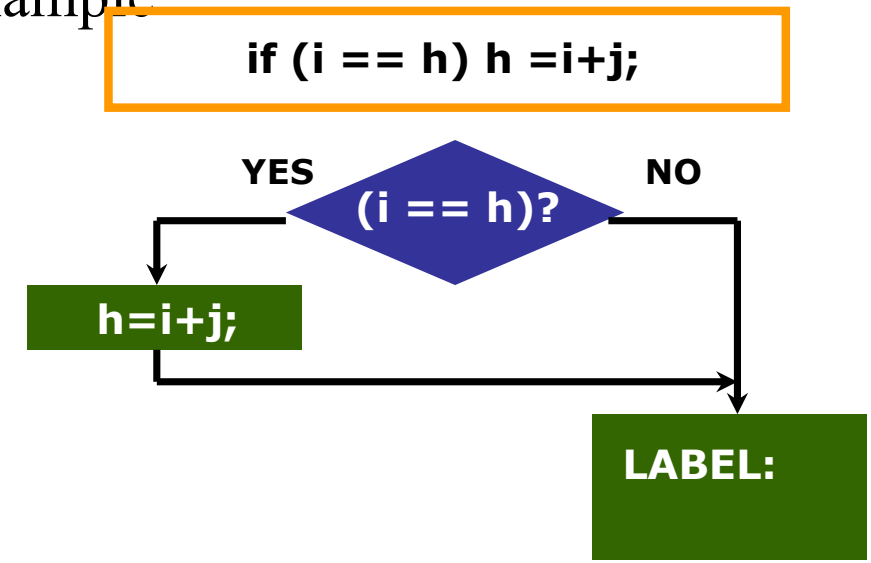
Control

- Instruction that potentially changes the flow of program execution
- MIPS conditional branch instructions
  - `bne $s4, $s3, LABEL` goto LABEL if $s4 != $s3
  - `beq $s4, $s3, LABEL` goto LABEL if $s4 == $s3
- Example

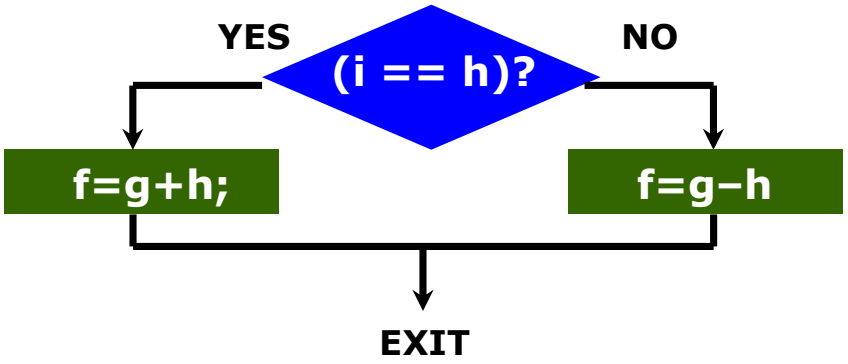



```
        bne $s5, $s6, LABEL
        add $s6, $s5, $s7
LABEL: ...
```

Control

- MIPS unconditional branch instruction (i.e., jump)
  - j LABEL
- Example
  - i, f, g, and h are in registers $s4, $s5, $s6, $s7

```
if (i == h) f=g+h;
else f=g–h;
```




```
    bne $s4, $s7, ELSE
    add $s5, $s6, $s7
    j EXIT
ELSE:
    sub $s5, $s6, $s7
EXIT:
    …
```

Control

- We have beq and bne; what about branch-if-less-than?
  - We have slt

```
if (s1<s2) t0=1;
else t0=0;
```

```
slt $t0, $s1, $s2
```

- Can you make a "psuedo" instruction "blt $s1, $s2, LABEL"?
- Assembler needs a temporary register to do this
  - $at is reserved for this purpose

```
slt     $at,$s1,$s2          # $at==1 when $s1<$s2
bne     $at,$0,LABEL         # $at!=0 implies $at==1
```

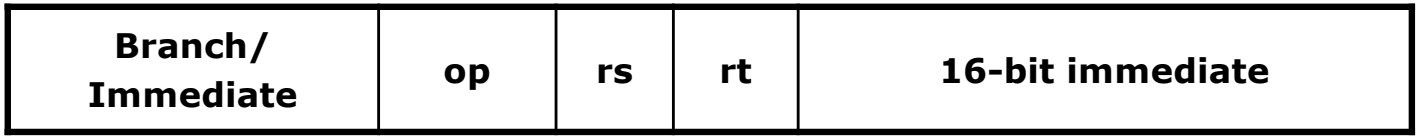
In-class exercise with control

- Let's write an assembly language program:
  - Print "Big endian" if machine is big endian, otherwise print "Little endian"

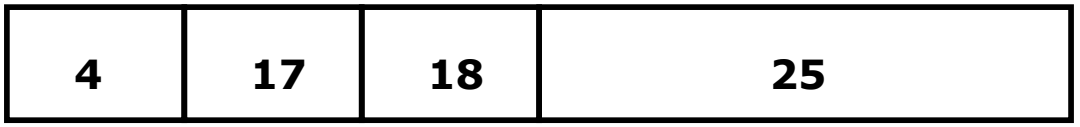
- We can do this by declaring some 4 bytes (word) in memory
- Then, load the word
- Check which which byte was put in the lowest position

- See inclass3.asm

Address in I-format



- Immediate address is not a 32-bit value – it's only 16-bit!
  - The 16-bit immediate value is signed (2's complement form)
- Addressing in branch instructions
  - The 16-bit number in the instruction specifies the number of "instructions" to be skipped
  - Memory address is obtained by adding this number to the PC
  - Next address = PC + 4 + sign-extend(16-bit immediate << 2)
  - B/C 16-bit immediate is signed: can go both FORWARD and BACKWARD
- Example
  - beq $s1, $s2, 100



Example of Addresses in Branches

- Branch address is "PC relative"
  - The target is RELATIVE to address of the branch instruction itself
  - Next PC = PC + 4 + sign-extend(16-bit immediate << 2)

```
0x00400000              beq     $s0,$s1,LABEL  # what's LABEL???
0x00400004              andi    $s0,$s0,0xFF
0x00400008              srl     $s0,$s0,2
0x0040000c              lui     $s1,0xFF00
0x00400010              ori     $s1,$s0,$s1
0x00400014      LABEL:  …
   PC when branch taken=0x00400000 + 4 + sign-extend(LABEL << 2)
   LABEL        = low16((0x00400014 - (0x00400000 + 4)) >> 2)
                = low16(0x10 >> 2)              (16/ 4)
                = 0x4                   (forward 4+1 instructions)
```

Example of Addresses in Branches

- Branch address is "PC relative"
  - The target is RELATIVE to address of the branch instruction itself
  - Next PC = PC + 4 + sign-extend(16-bit immediate << 2)

```
0x00400000      LABEL:  andi    $s0,$s0,0xFF
0x00400004              srl     $s0,$s0,2
0x00400008              lui     $s1,0xFF00
0x0040000c              ori     $s1,$s0,$s1
0x00400010              beq     $s0,$s1,LABEL  # what's LABEL???
0x00400014              …
   PC when branch taken=0x00400010 + 4 + sign-extend(LABEL << 2)
   LABEL        = low16((0x00400000 - (0x00400010 + 4))) >> 2)
                = low16((0xFFFFFFEC) >> 2)     (-20 / 4)
                = 0xFFFB                       (backward -4-1 instructions)
```

J-format

- The address of next instruction is obtained from PC and the immediate value
  - Next address = {PC[31:28],IMM[25:0],00}
  - Address boundaries of 256MB
- Example
  - j 10000

Control

- What about comparisons against constant?
  - Can we use "beqi"??? Why not?
- Comparison variants with an immediate
  - slti $s0,$s1,16
  - addi $s0,$s1,-20 ($s0==0 when $s1==20)

```
if      (a > 20) { a = a + 2; }

        slti    $at,$s0,21      # $at==1 when a<=20
        bne     $at,$0,LAB      # $at!=0 implies $at==1, skip
        addi    $s0,$s0,2
LAB:    …                       # target when a<=20
```

Control

- Loops
  - "While" loops
    - Example on page 74
  - "For" loops

While Loop

```
while (condition == true) { some work; }
```

use comparison, branch to exit, and jump to continue

generally written as:

```
loop:  check condition
       branch if condition is false to exit
       some work
       j      loop
exit:
```

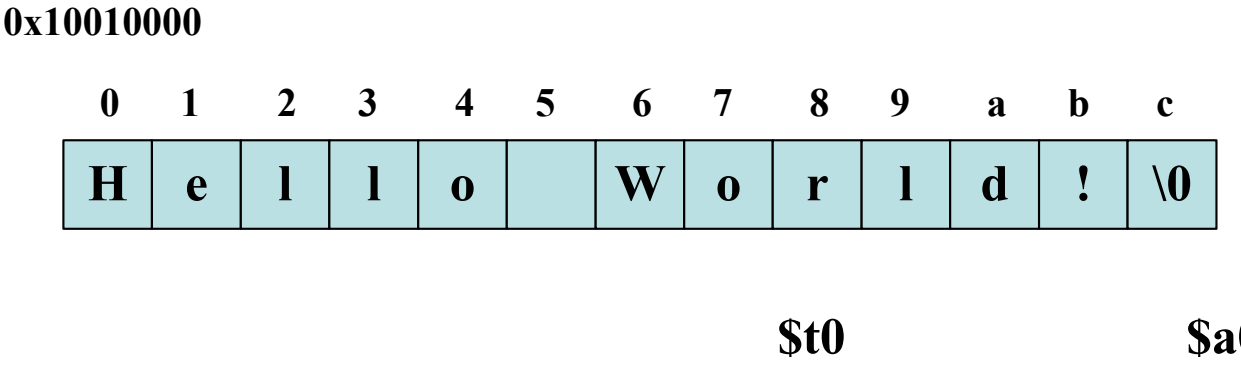
While Loop

```
char *str = "Hello World!";
char *s = str;
while (*s != '\0') { printf("%c", *s); s = s + 1; }
```

```
        .data
str:    .asciiz       "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

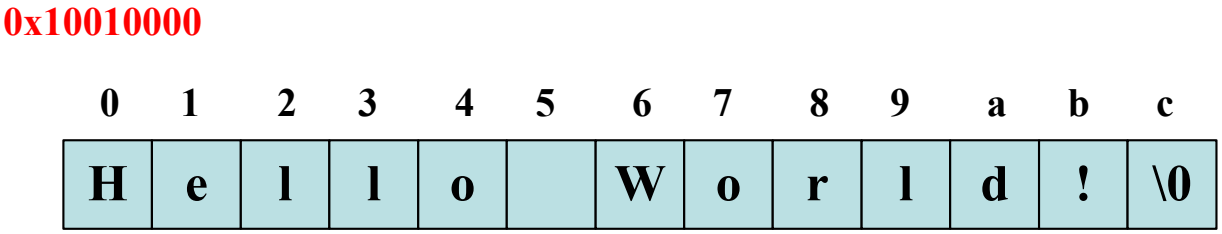
See mips12.asm

While Loop




```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

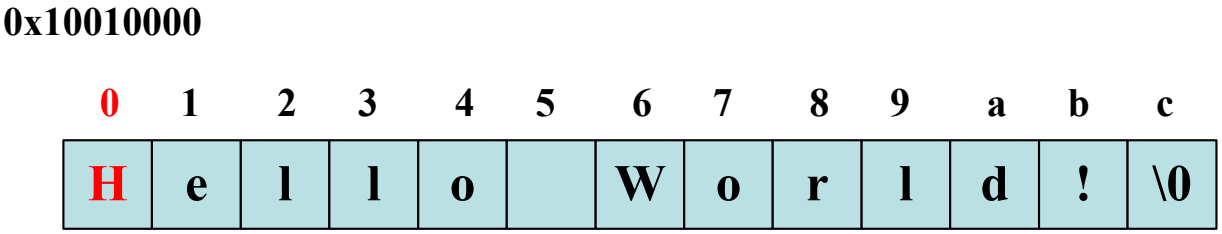
While Loop



$t0	$a0
10010000	0

```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

While Loop

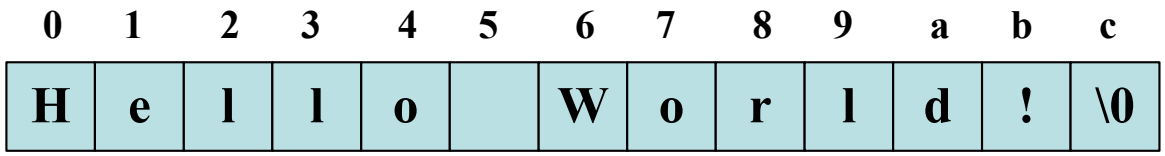


$t0	$a0
10010000	0
10010000	'H'

```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

While Loop

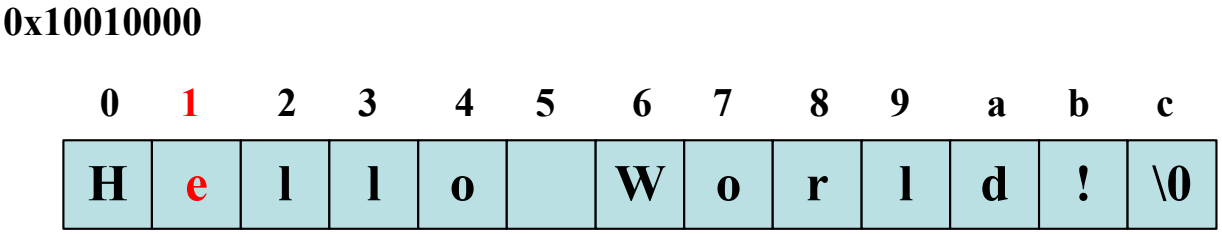
0x10010000



$t0	$a0
10010000	0
10010000	'H'
10010001	'H'

```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

While Loop




```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

$t0	$a0
10010000	0
10010000	'H'
10010001	'H'
10010001	'e'

While Loop

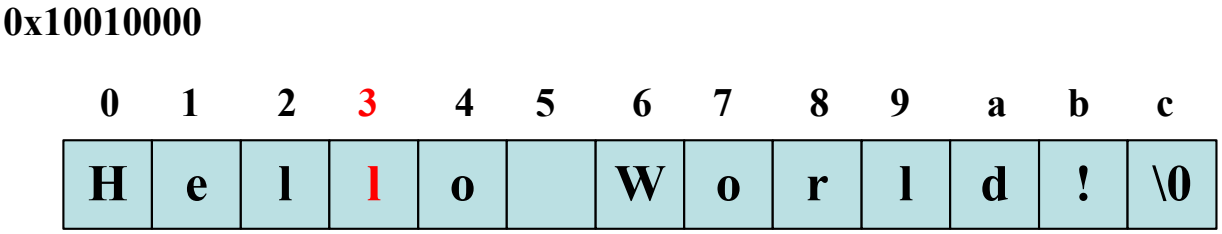
0x10010000

0	1	2	3	4	5	6	7	8	9	a	b	c
H	e	l	l	o		W	o	r	l	d	!	\0

```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

$t0	$a0
10010000	0
10010000	'H'
10010001	'H'
10010001	'e'
10010002	'l'

While Loop

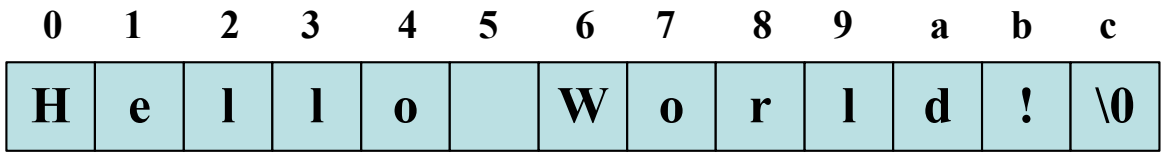



```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

$t0	$a0
10010000	0
10010000	'H'
10010001	'H'
10010001	'e'
10010002	'l'
10010003	'l'

While Loop

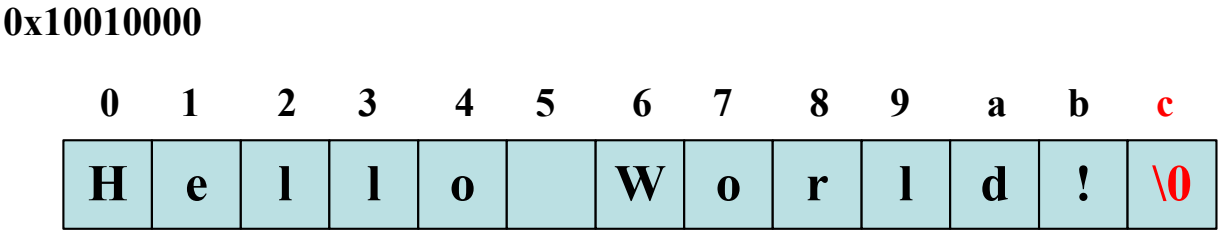
0x10010000




```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

$t0	$a0
10010000	0
10010000	'H'
10010001	'H'
10010001	'e'
10010002	'l'
10010003	'l'
10010004	'o'
10010005	' '
10010006	'W'
10010007	'o'
10010008	'r'
10010009	'l'
1001000a	'd'
1001000b	'!'

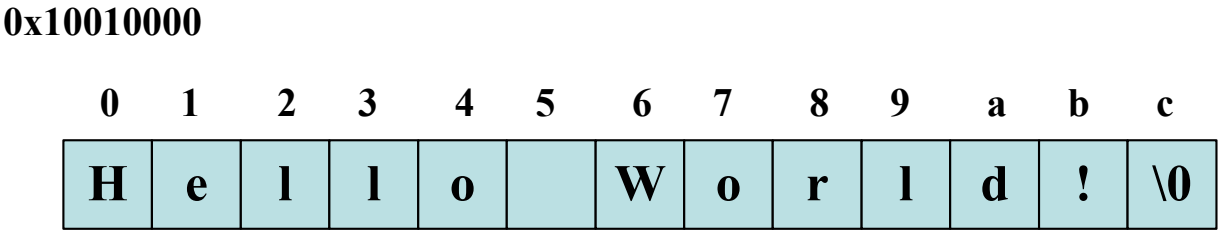
While Loop




```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

$t0	$a0
10010000	0
10010000	'H'
10010001	'H'
10010001	'e'
10010002	'l'
10010003	'l'
10010004	'o'
10010005	' '
10010006	'W'
10010007	'o'
10010008	'r'
10010009	'l'
1001000a	'd'
1001000b	'!'
1001000c	'\0'

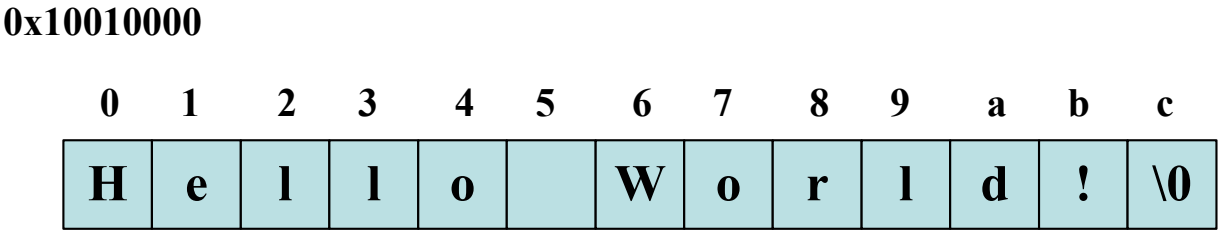
While Loop




```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

$t0	$a0
10010000	0
10010000	'H'
10010001	'H'
10010001	'e'
10010002	'l'
10010003	'l'
10010004	'o'
10010005	' '
10010006	'W'
10010007	'o'
10010008	'r'
10010009	'l'
1001000a	'd'
1001000b	'!'
1001000c	'\0'

While Loop




```
        .data
str:    .asciiz     "Hello World!"
        .text
        la      $t0,str
loop:   lb      $a0,0($t0)
        beq     $a0,$0,exit
        li      $v0,11 # print character
        syscall
        addi    $t0,$t0,1
        j       loop
exit:
```

$t0	$a0
10010000	0
10010000	'H'
10010001	'H'
10010001	'e'
10010002	'l'
10010003	'l'
10010004	'o'
10010005	' '
10010006	'W'
10010007	'o'
10010008	'r'
10010009	'l'
1001000a	'd'
1001000b	'!'
1001000c	'\0'

Do While Loop

```
do { some work; } while (condition == true);
```

use comparison, branch to exit, and jump to continue

generally written as:

```
loop:  some work
       check condition
       branch if condition is true to loop
exit:
```

Do While Loop

```
char *str = "Hello World!";
char *s = str;
/* note: this breaks with the empty string! */
do { printf("%c",*s); s = s + 1; } while (*s != '\0')
```

```
      la    $t0,str
      lb    $a0,0($t0)   # start the loop
loop: li    $v0,11       # print character
      syscall
      addi  $t0,$t0,1
      lb    $a0,0($t0)
      bne   $a0,$0,loop
```

See mips11.asm

Another loop example…

- Reverse one string in place in memory (without copying them from one location to another)

Reversing a String

- Suppose we want to reverse a string in memory
  - Reverse the string in place without copying it
  - Assume $a0 holds the address of the string in memory
  - String is null terminated (asciiz type)

- How can we do this?
  - Understand layout (arrangement) of the string in memory
  - Swap positions (characters) of the string one-by-one
  - Loop to process the positions (characters) of the string

Reverse a String

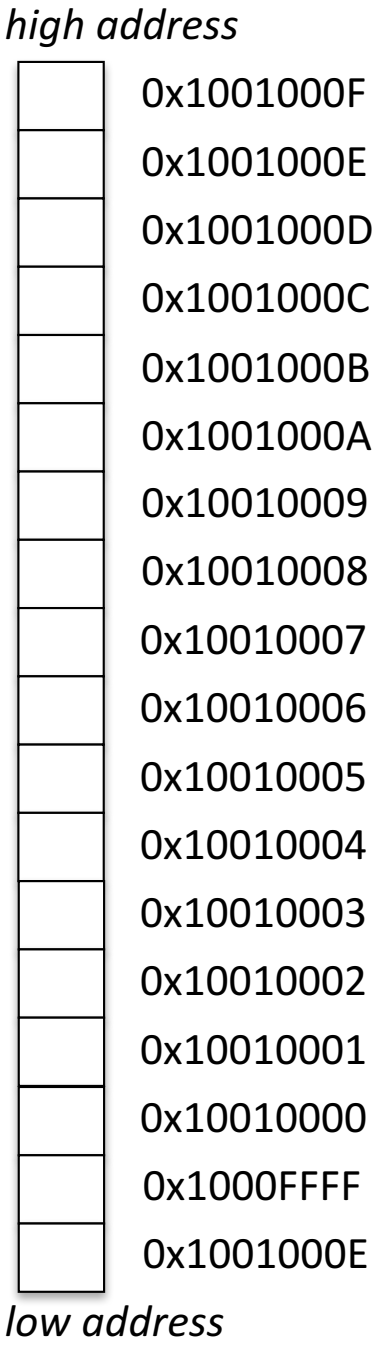
```
        .data
msg:    .asciiz  “Hello CS 447!”
```

Reverse a String

```
        .data
msg:    .asciiz  “Hello CS 447!”
```

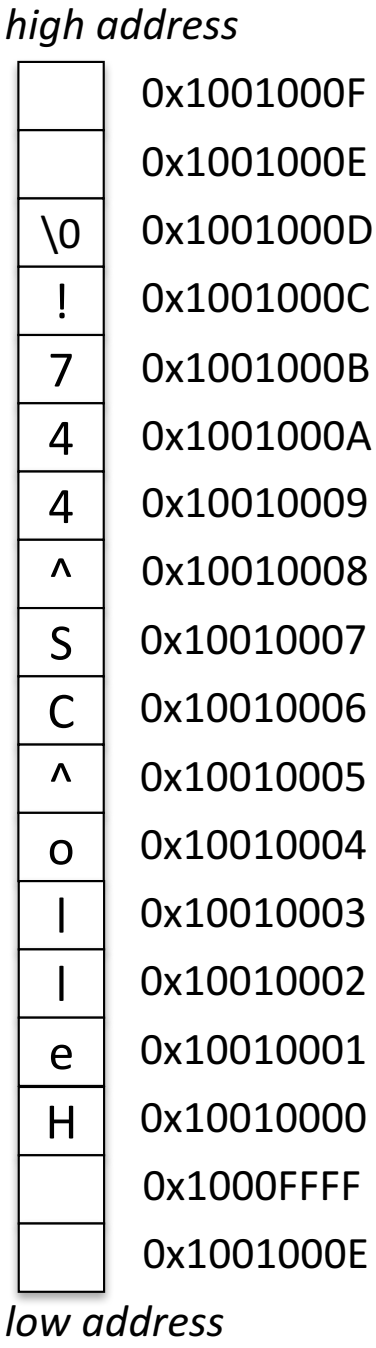
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



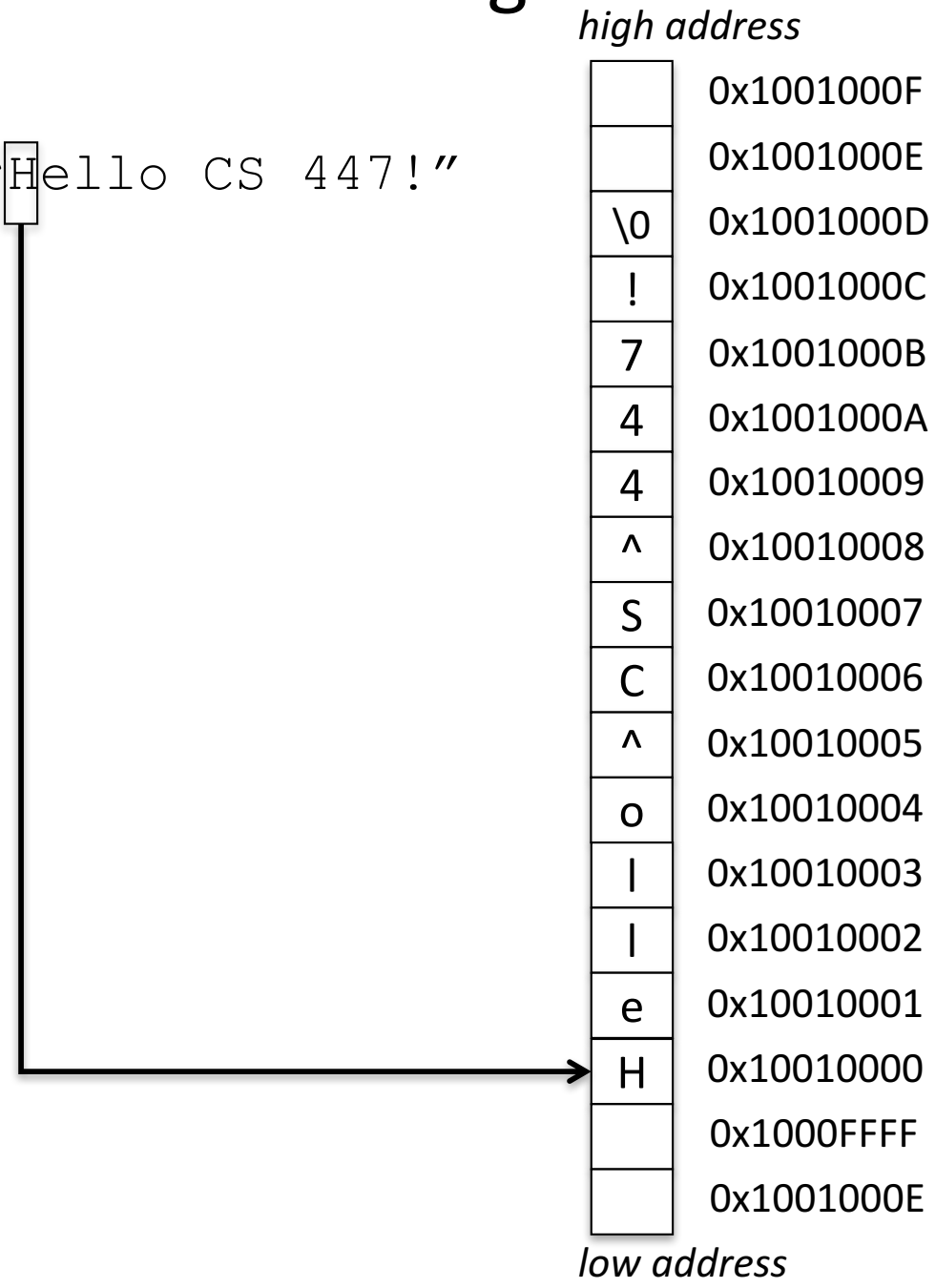
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



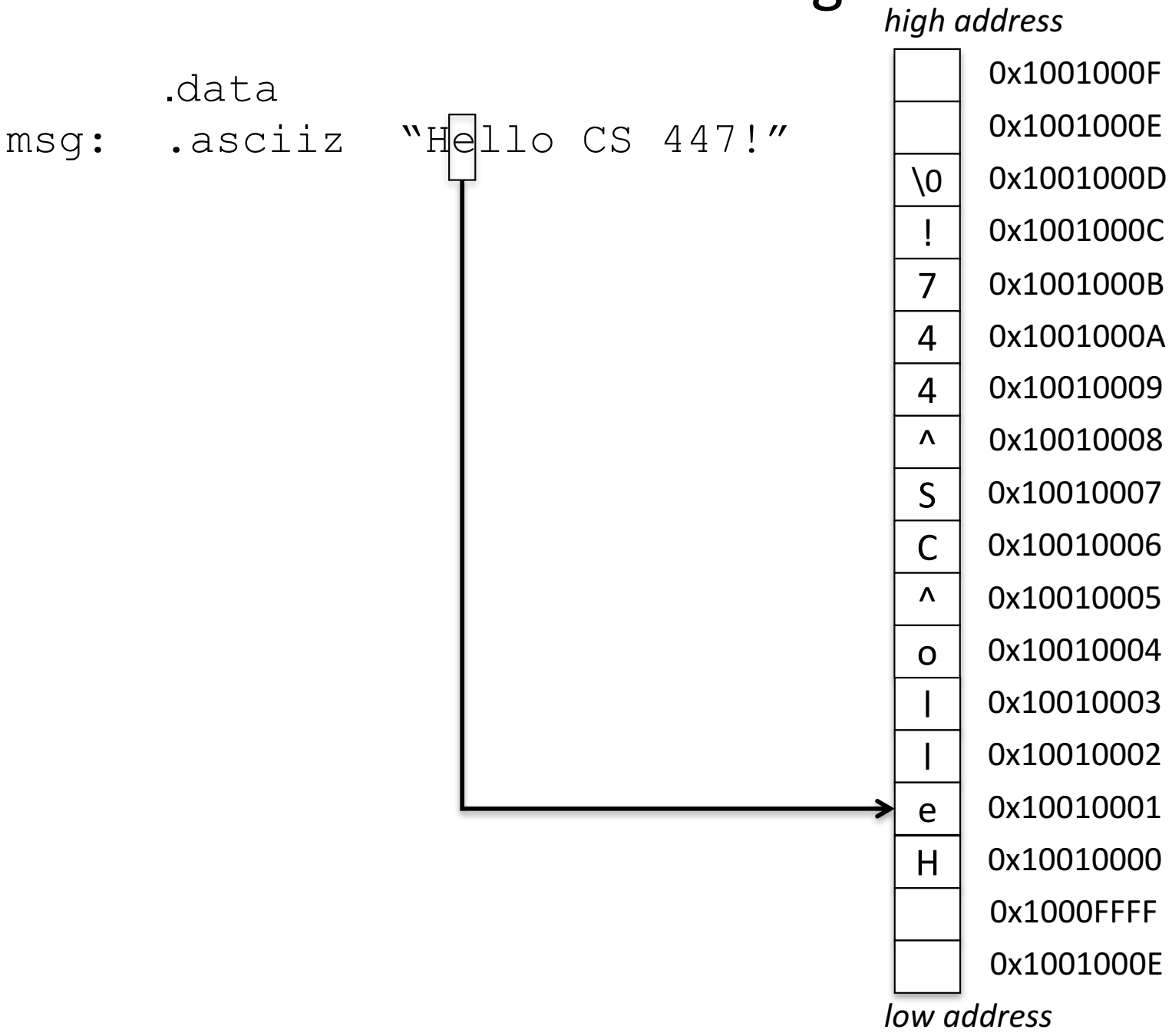
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



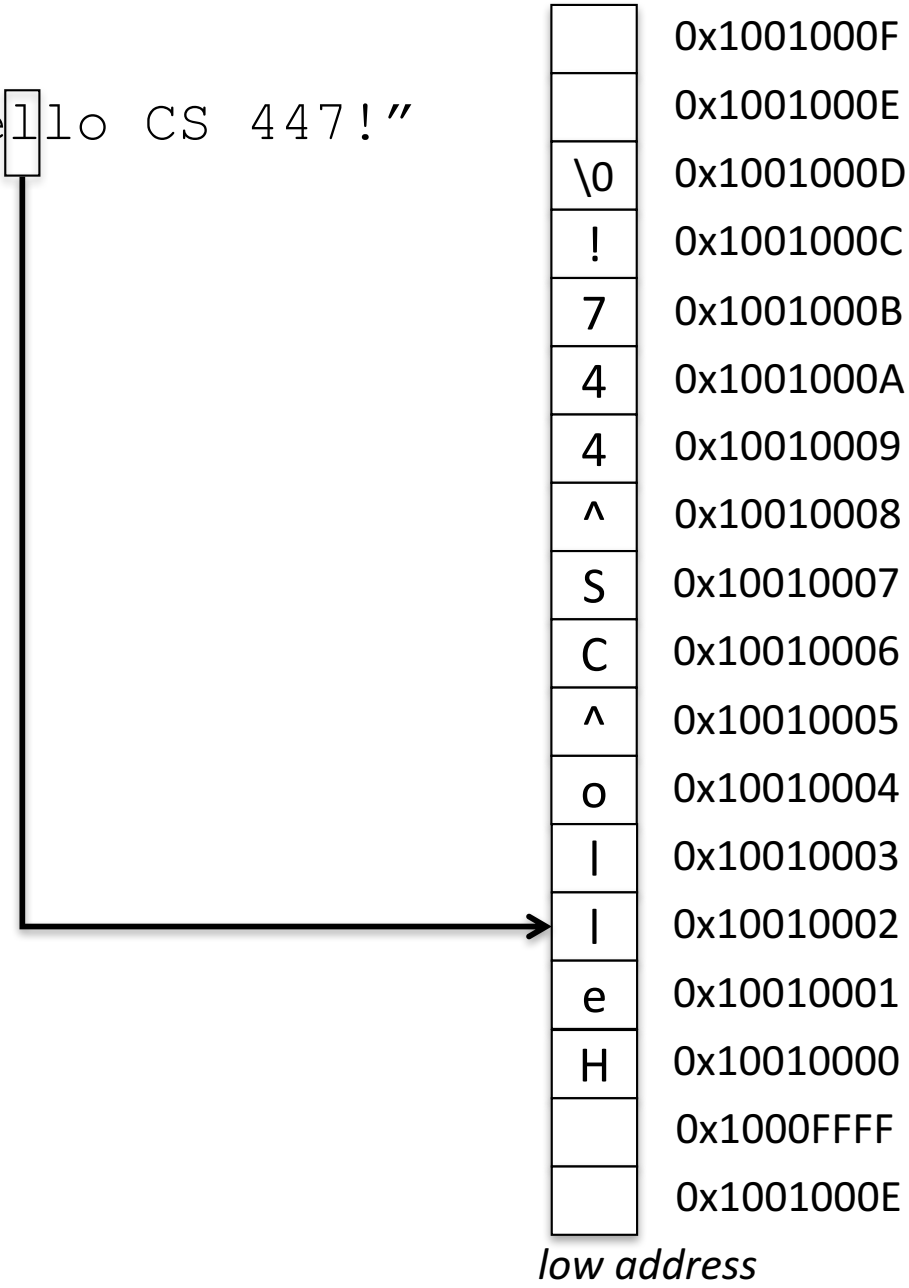
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



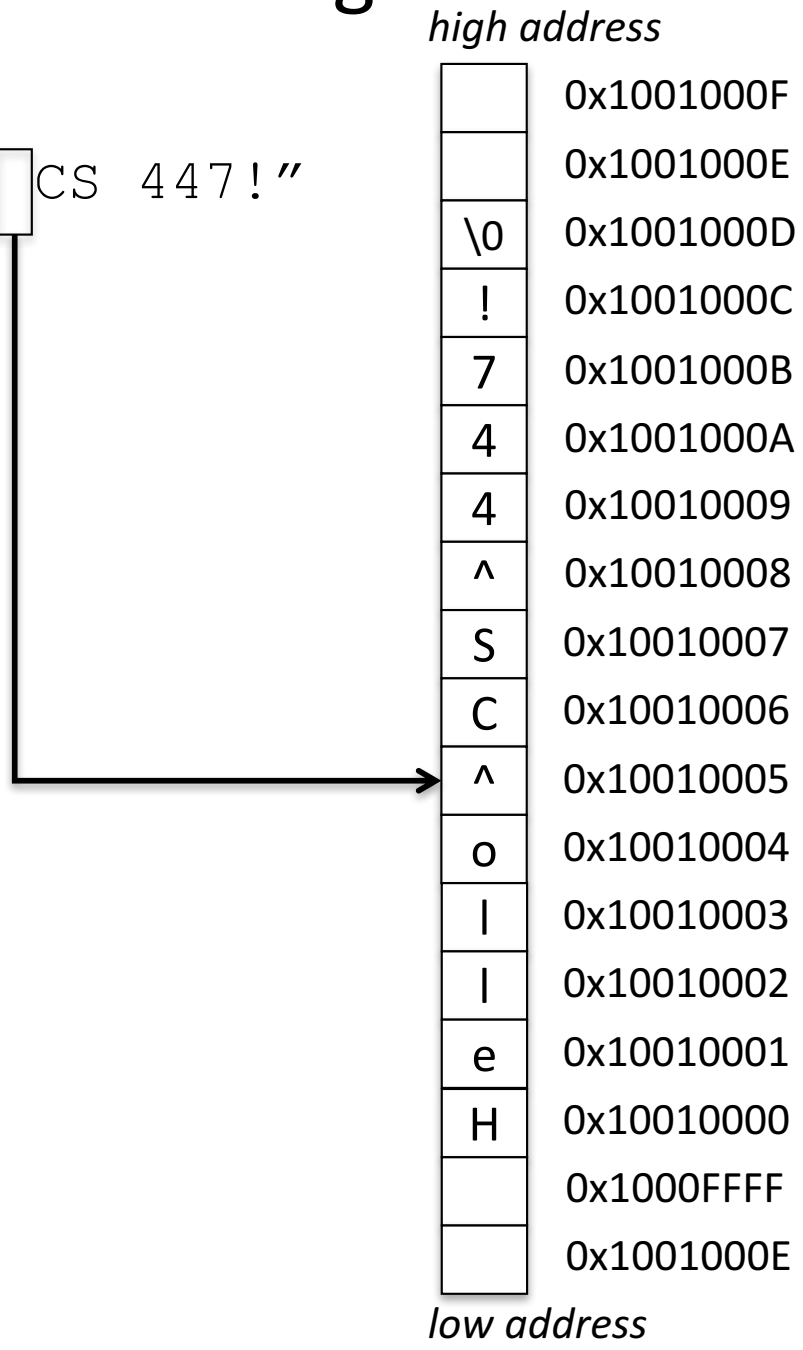
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



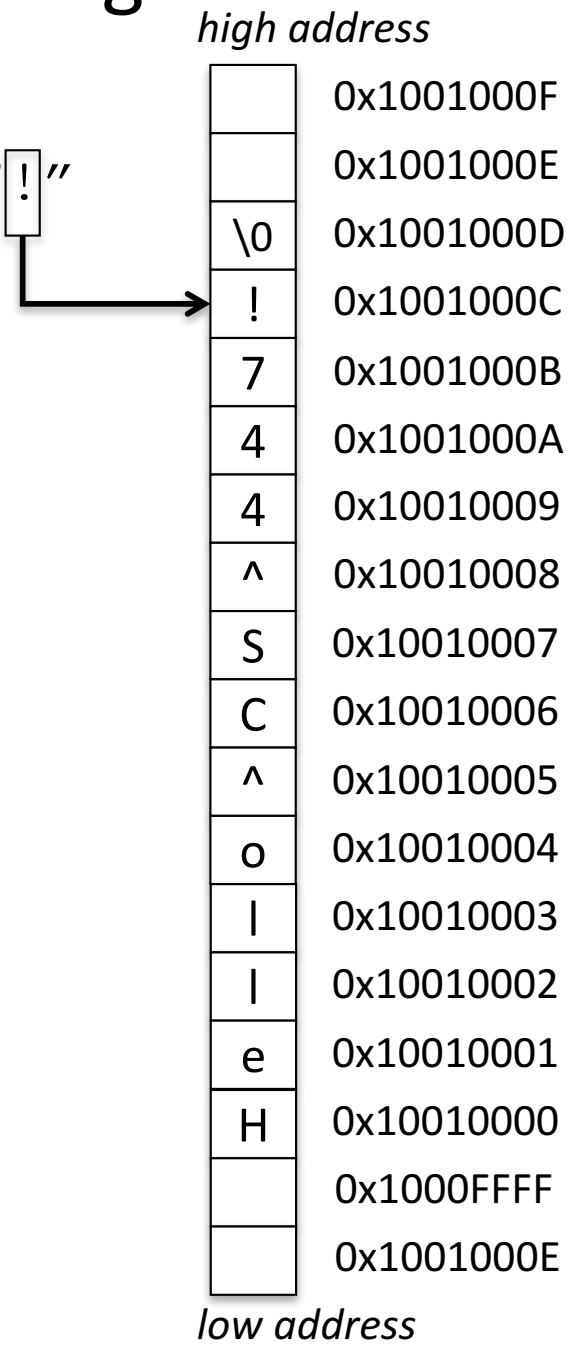
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



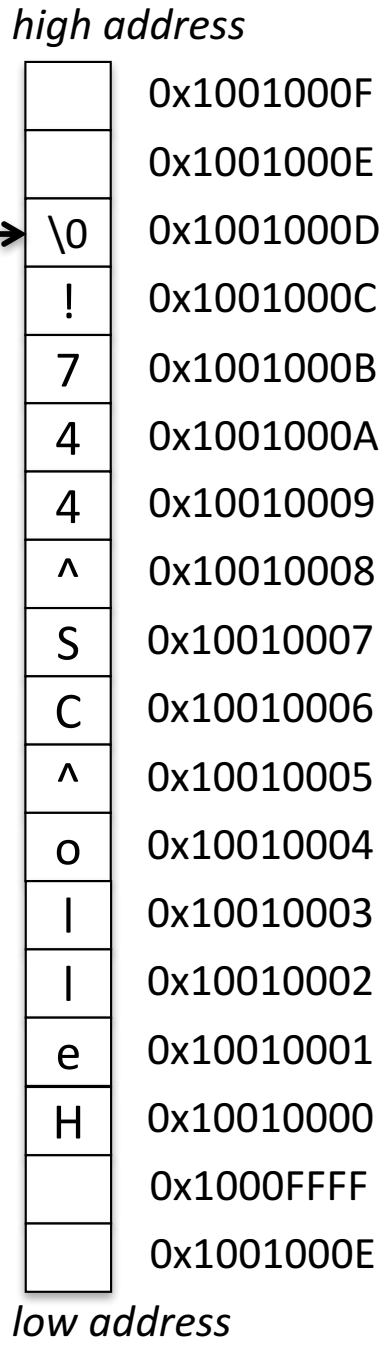
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



Reverse a String

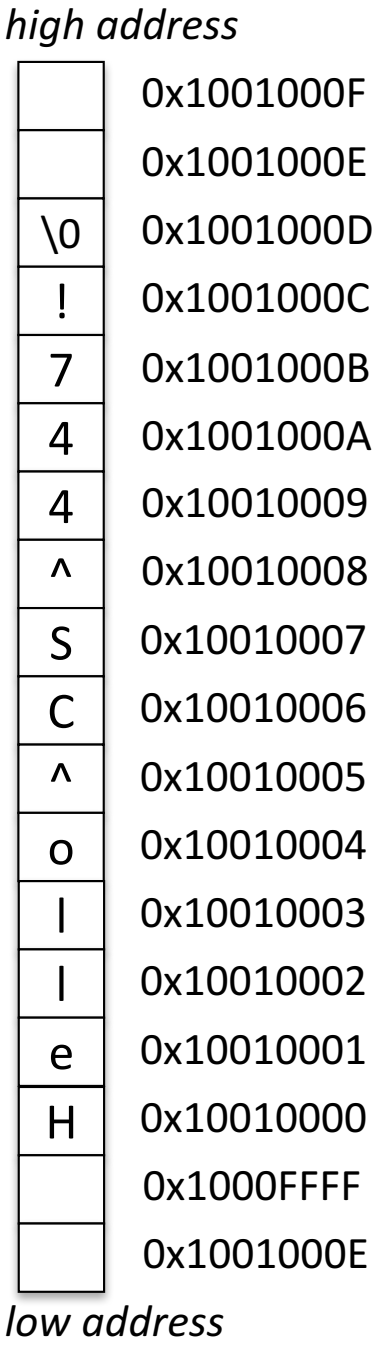
```
        .data
msg:    .asciiz  "Hello CS 447!"
```



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

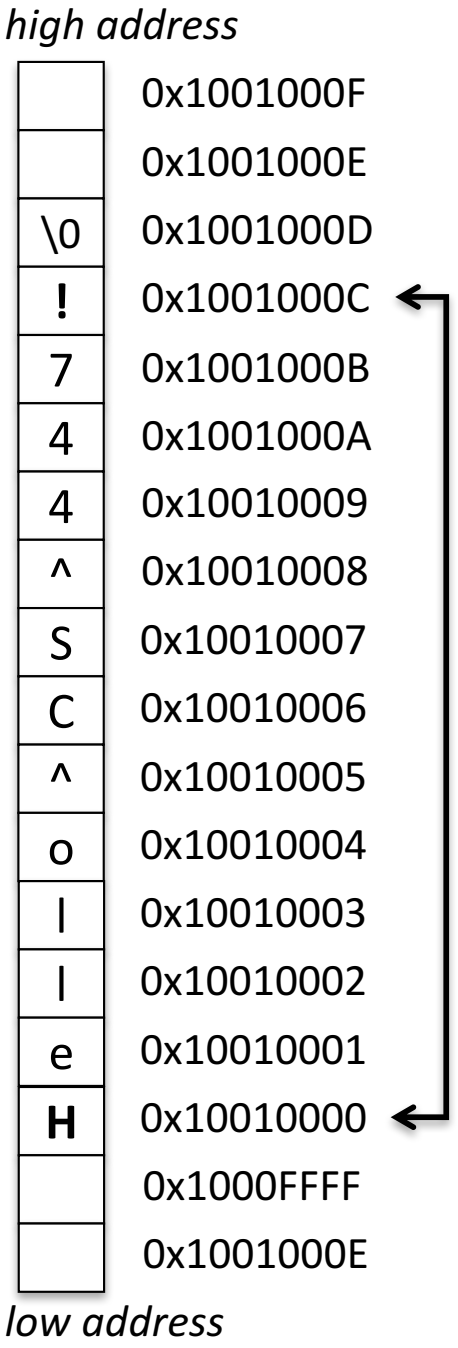
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

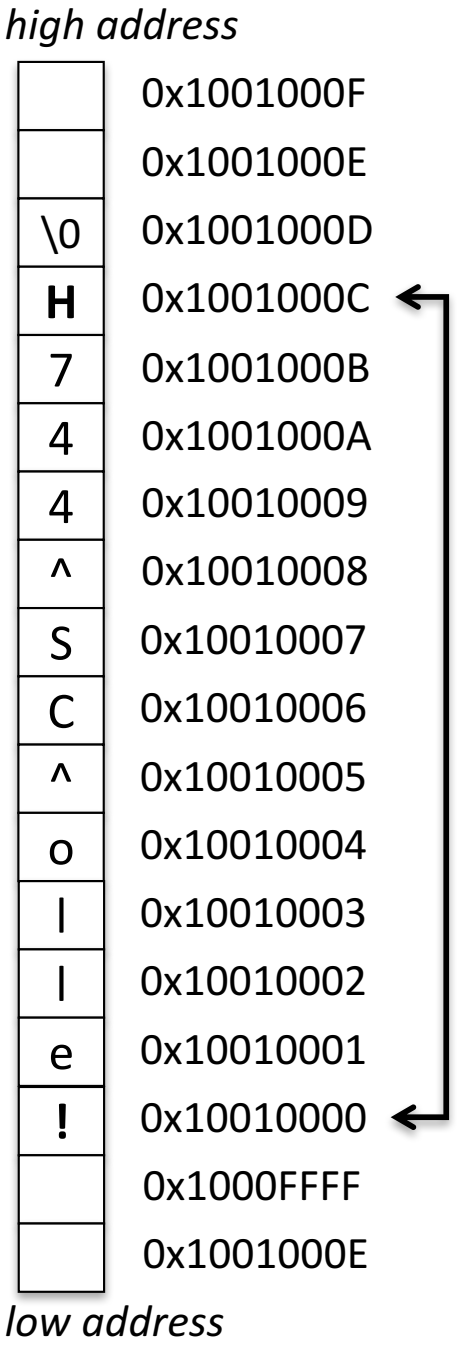
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

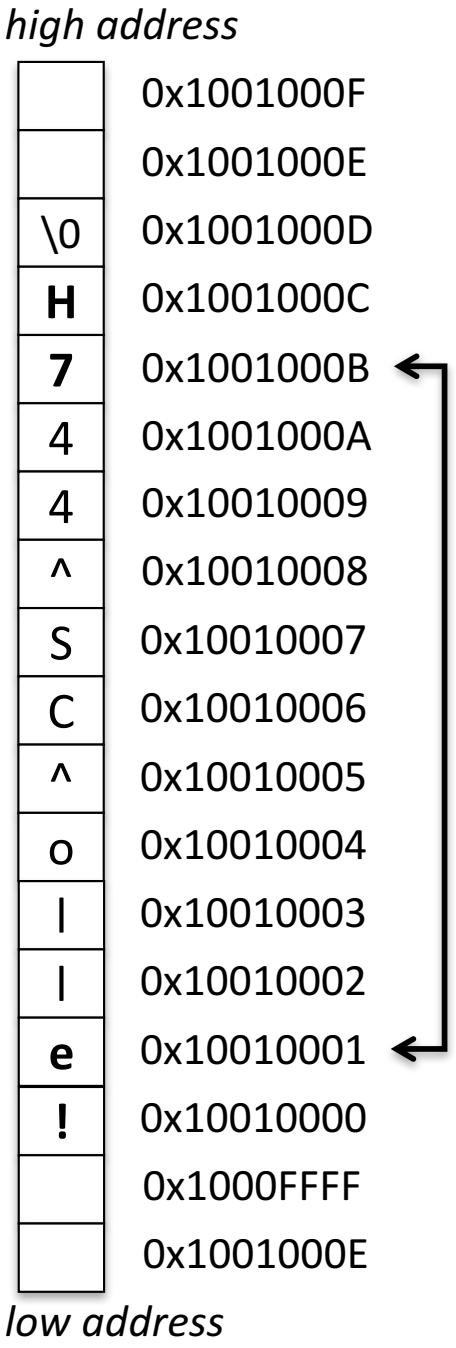
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

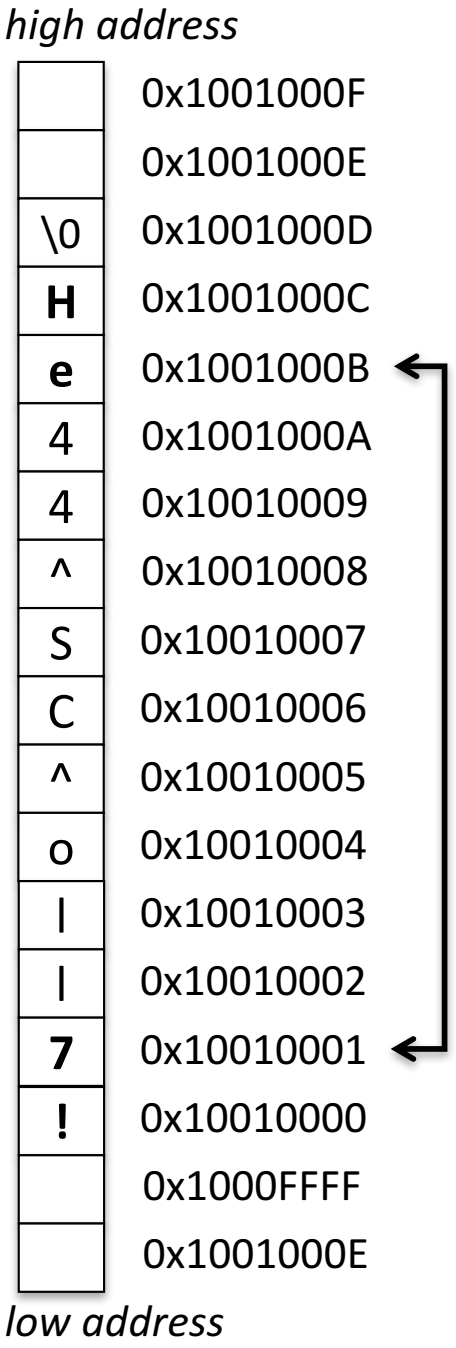
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

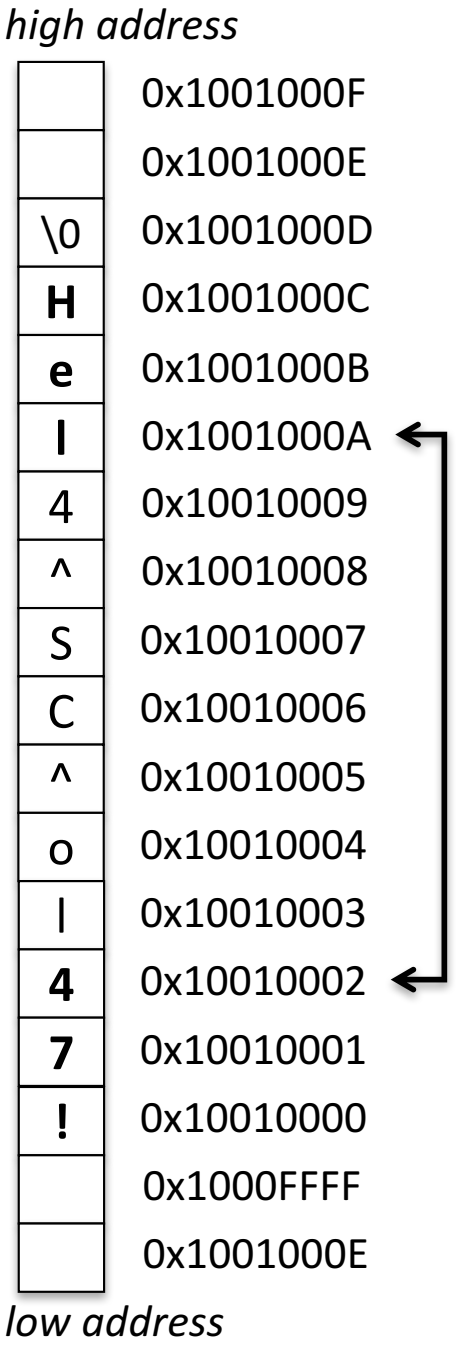
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

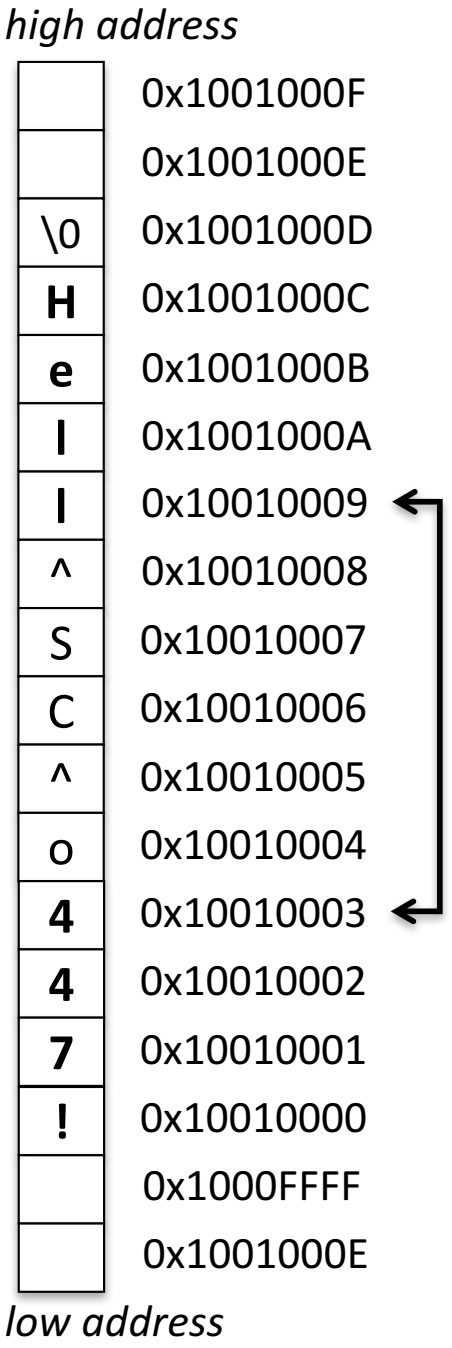
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

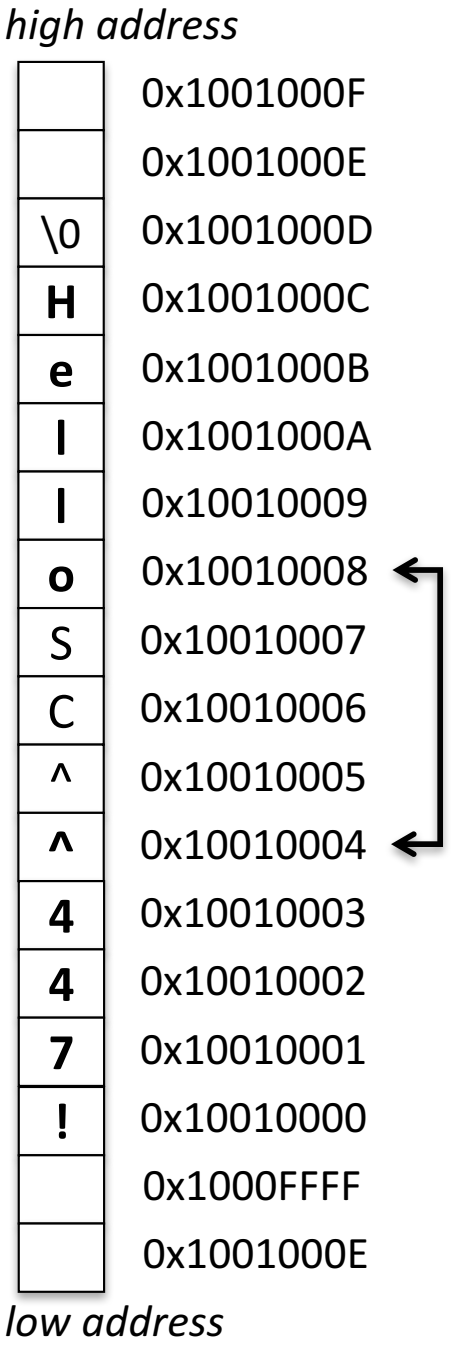
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

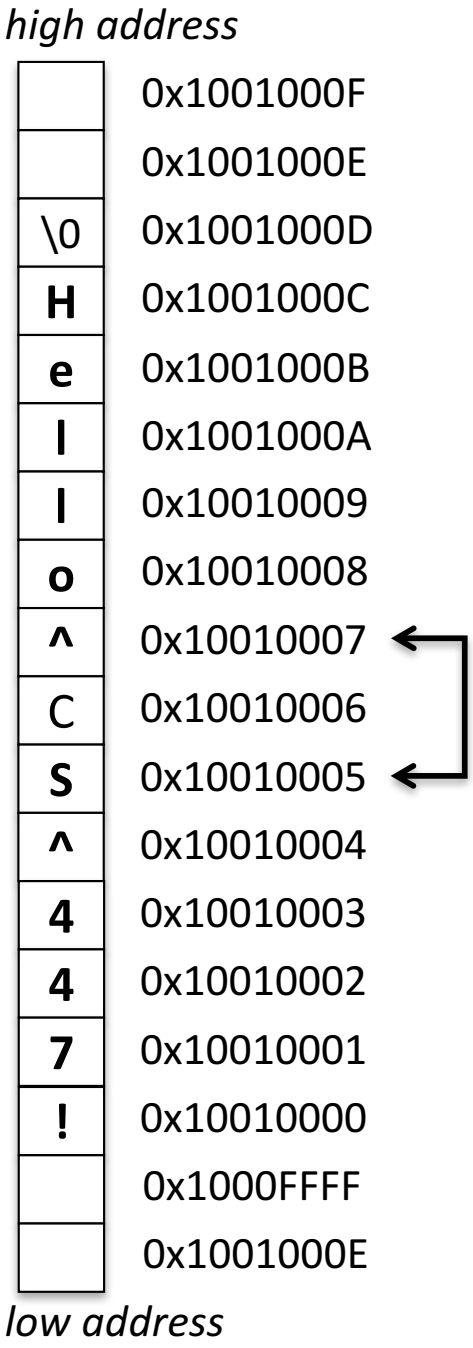
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

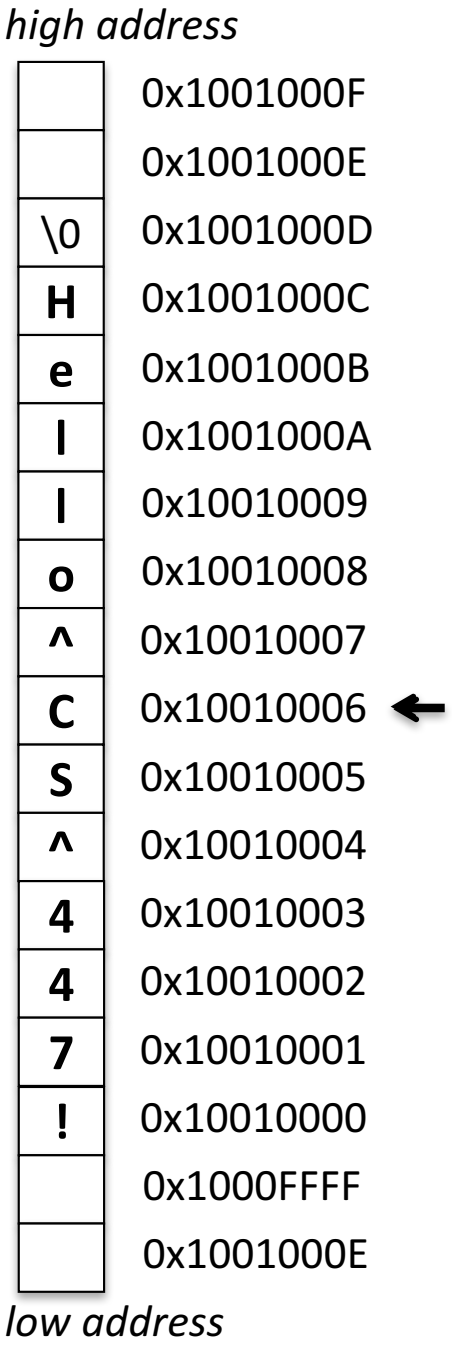
we can reverse by swapping characters



Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

we can reverse by swapping characters

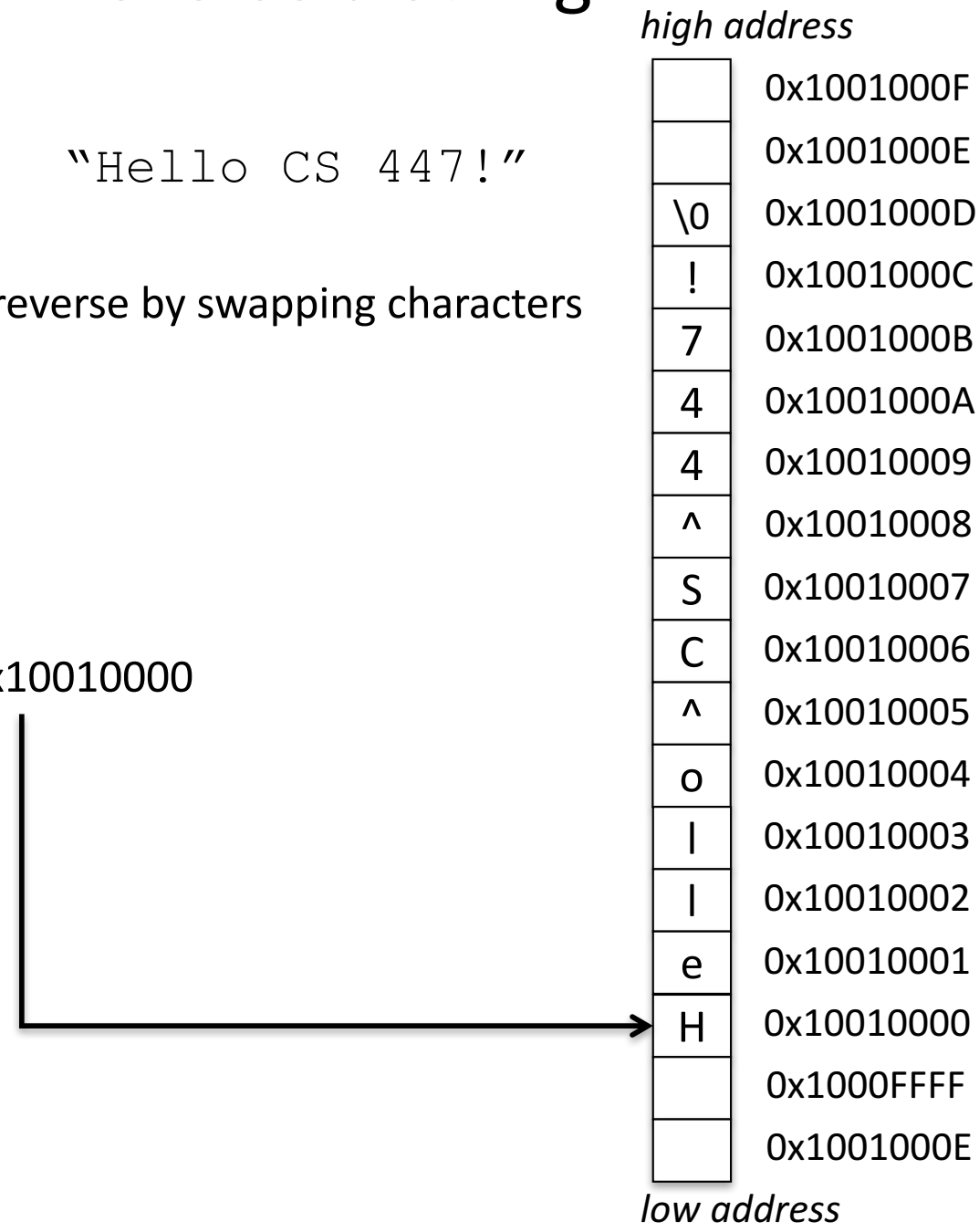


Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

we can reverse by swapping characters

$a0 = 0x10010000



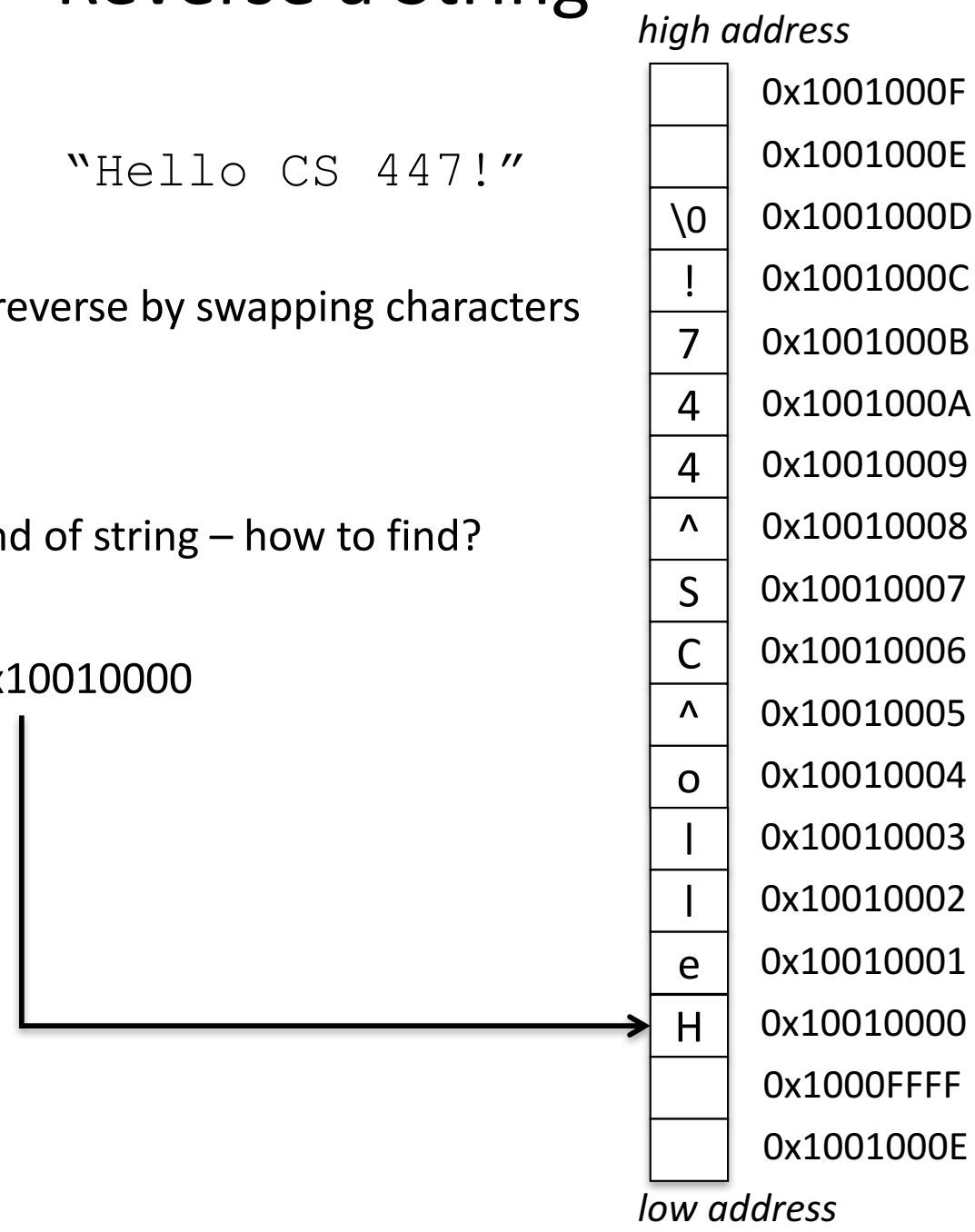
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

we can reverse by swapping characters

$a1 = end of string – how to find?

$a0 = 0x10010000

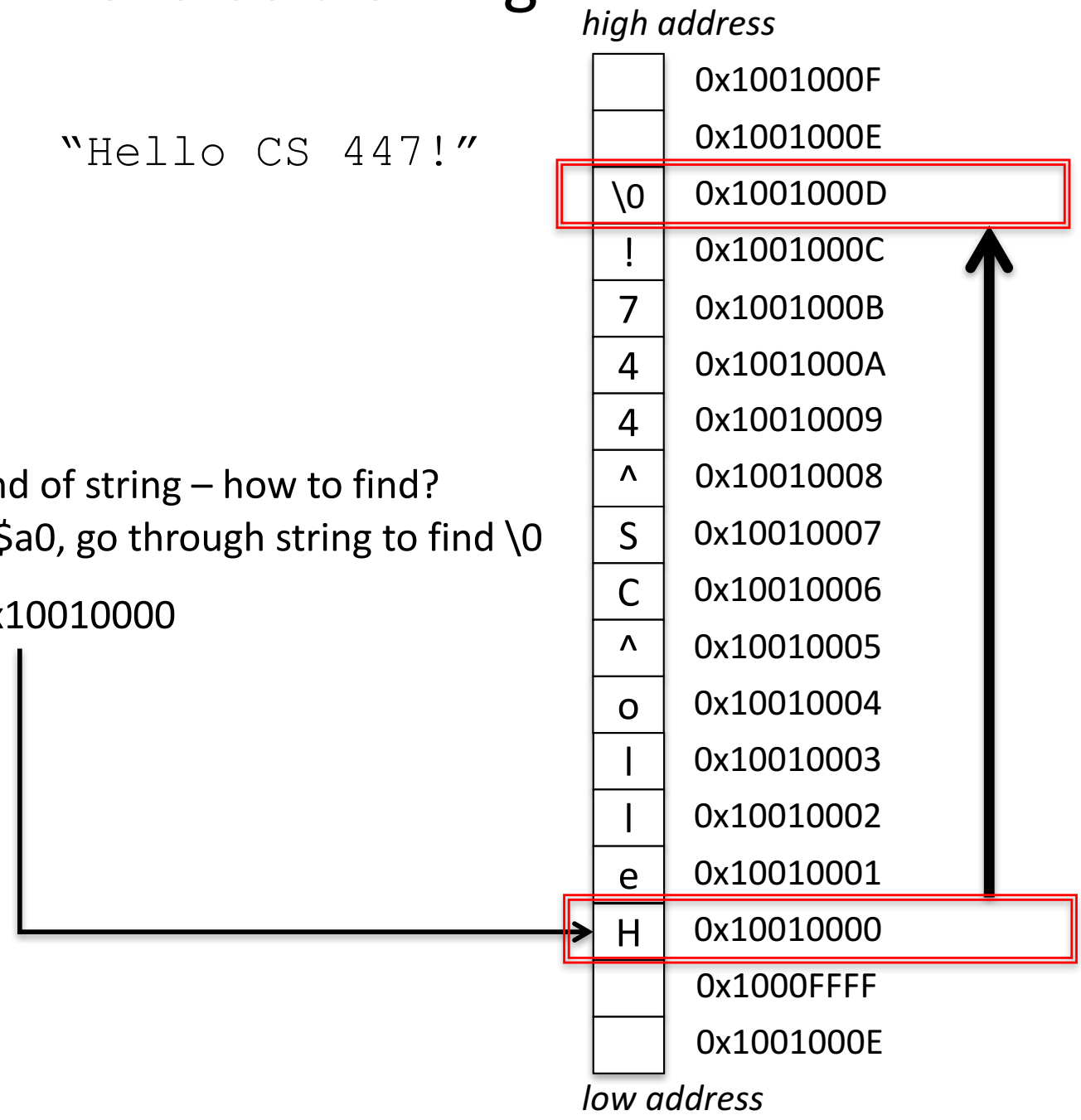


Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```

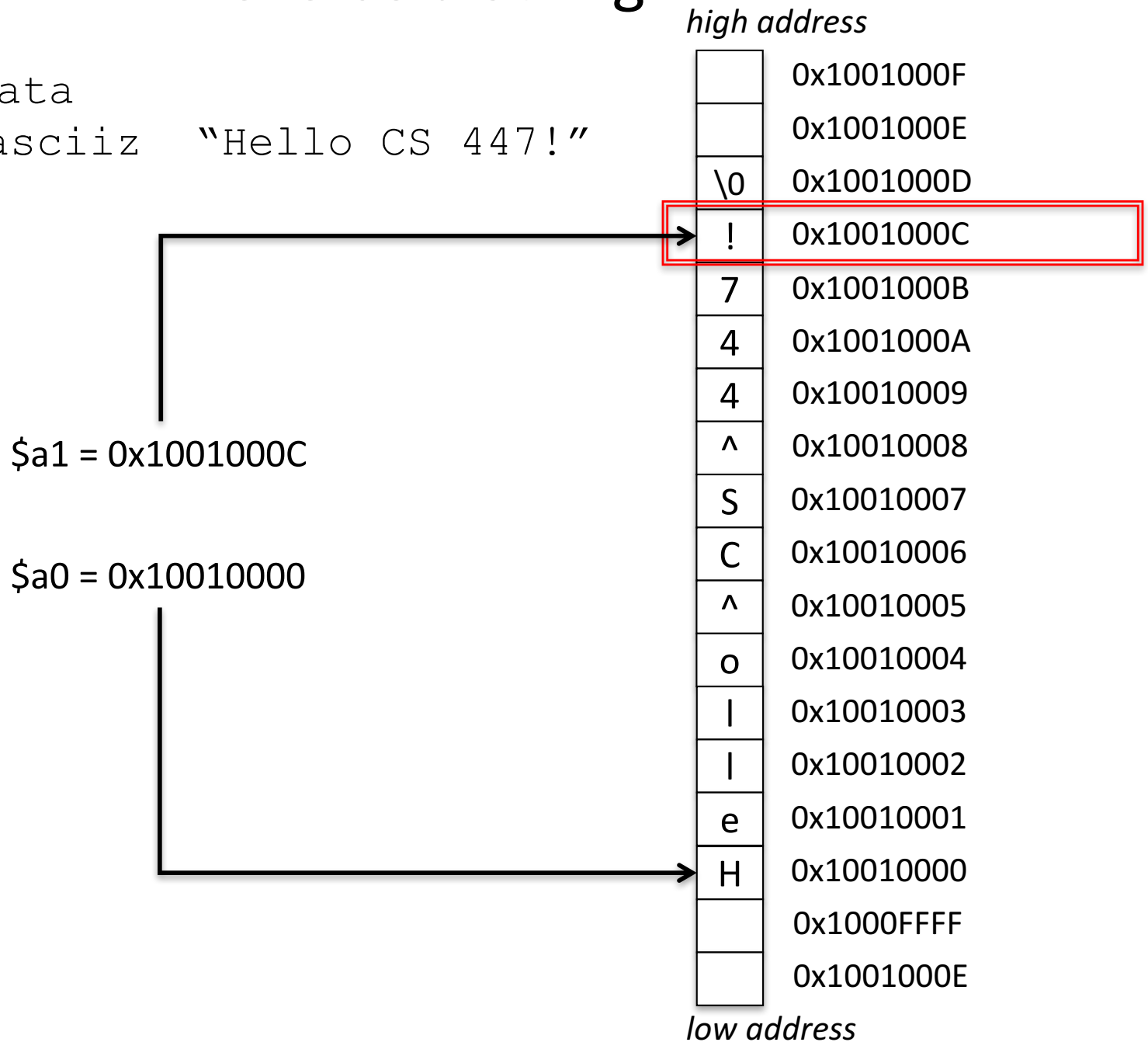
$a1 = end of string – how to find?
start at $a0, go through string to find \0

$a0 = 0x10010000



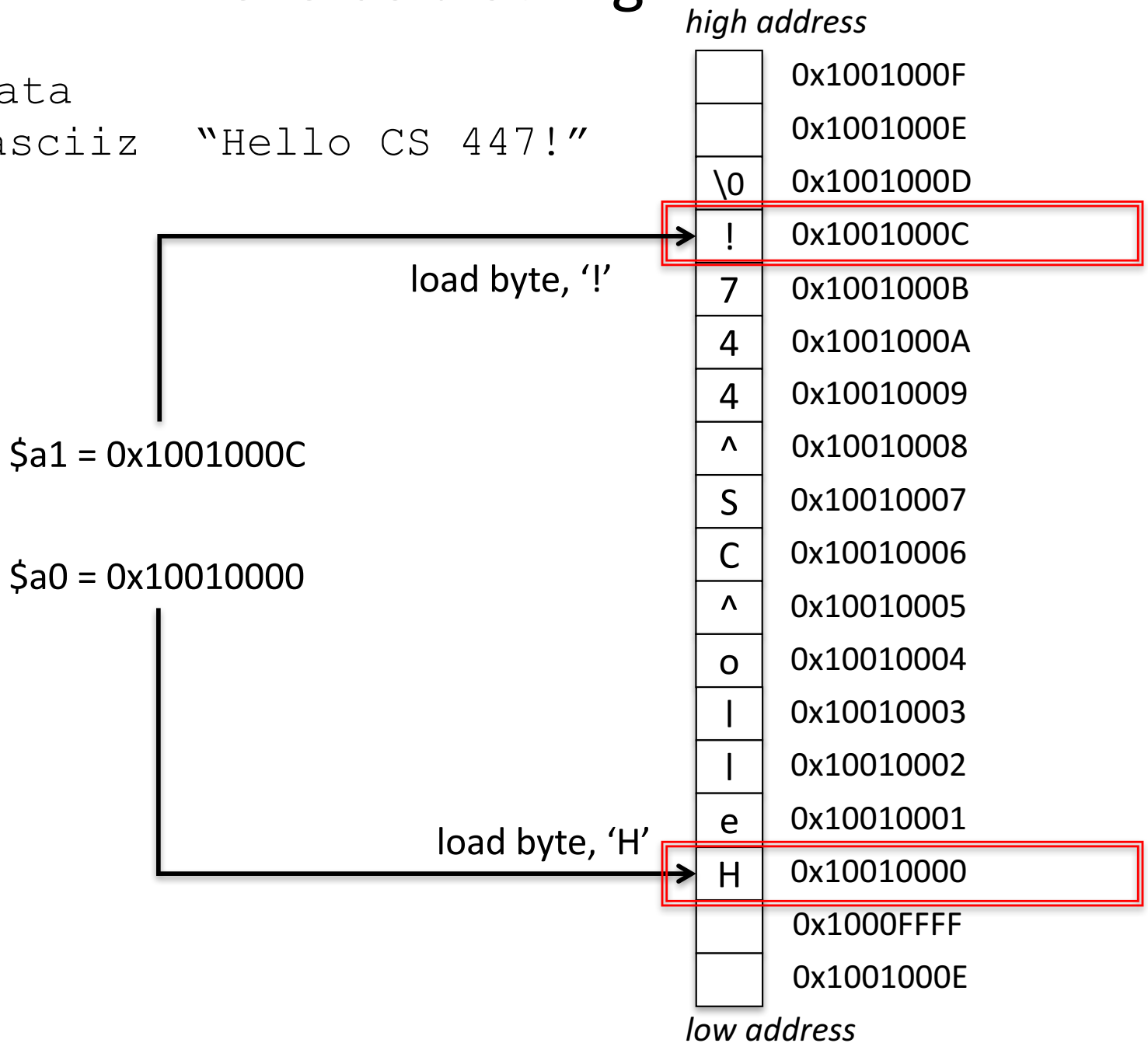
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



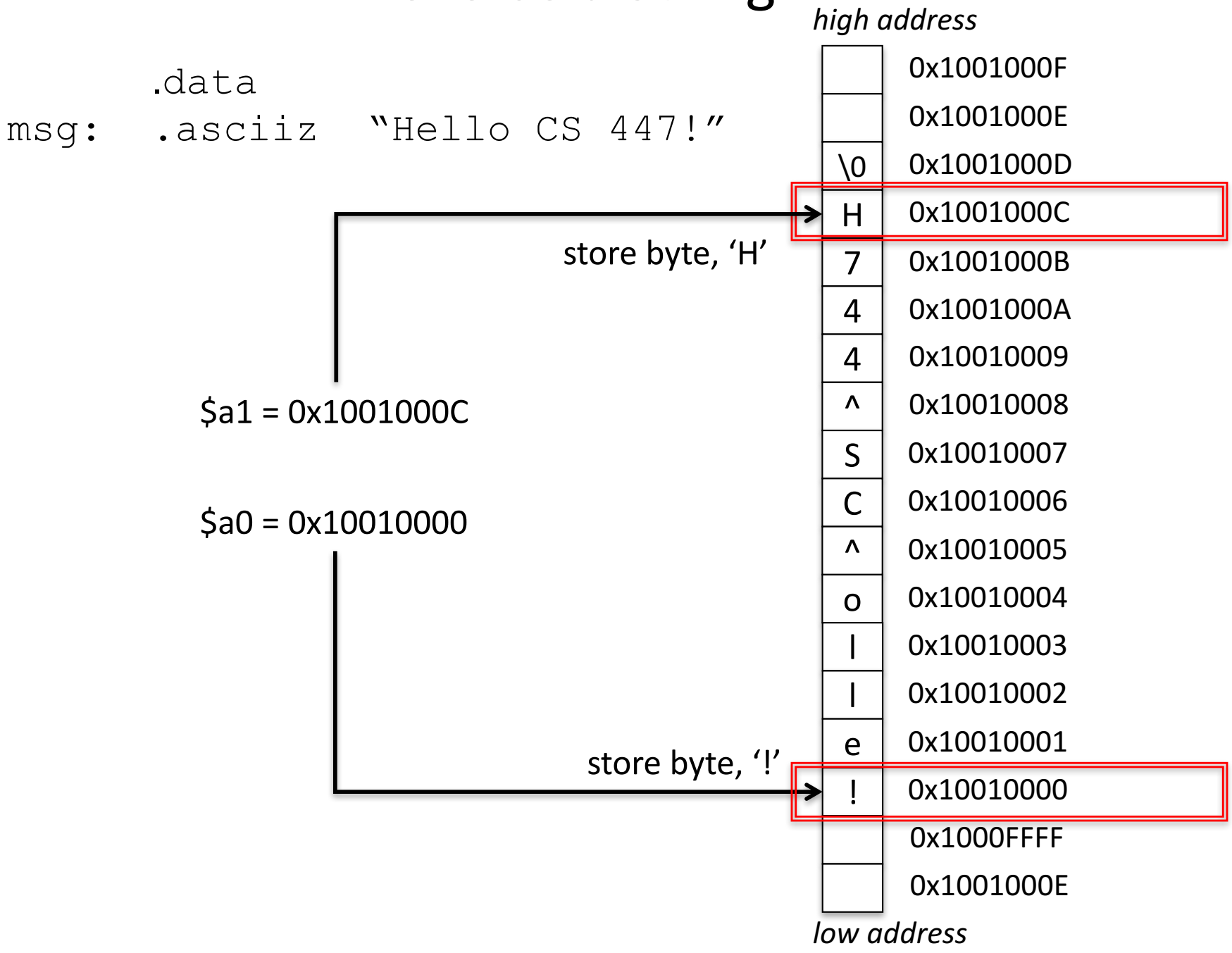
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



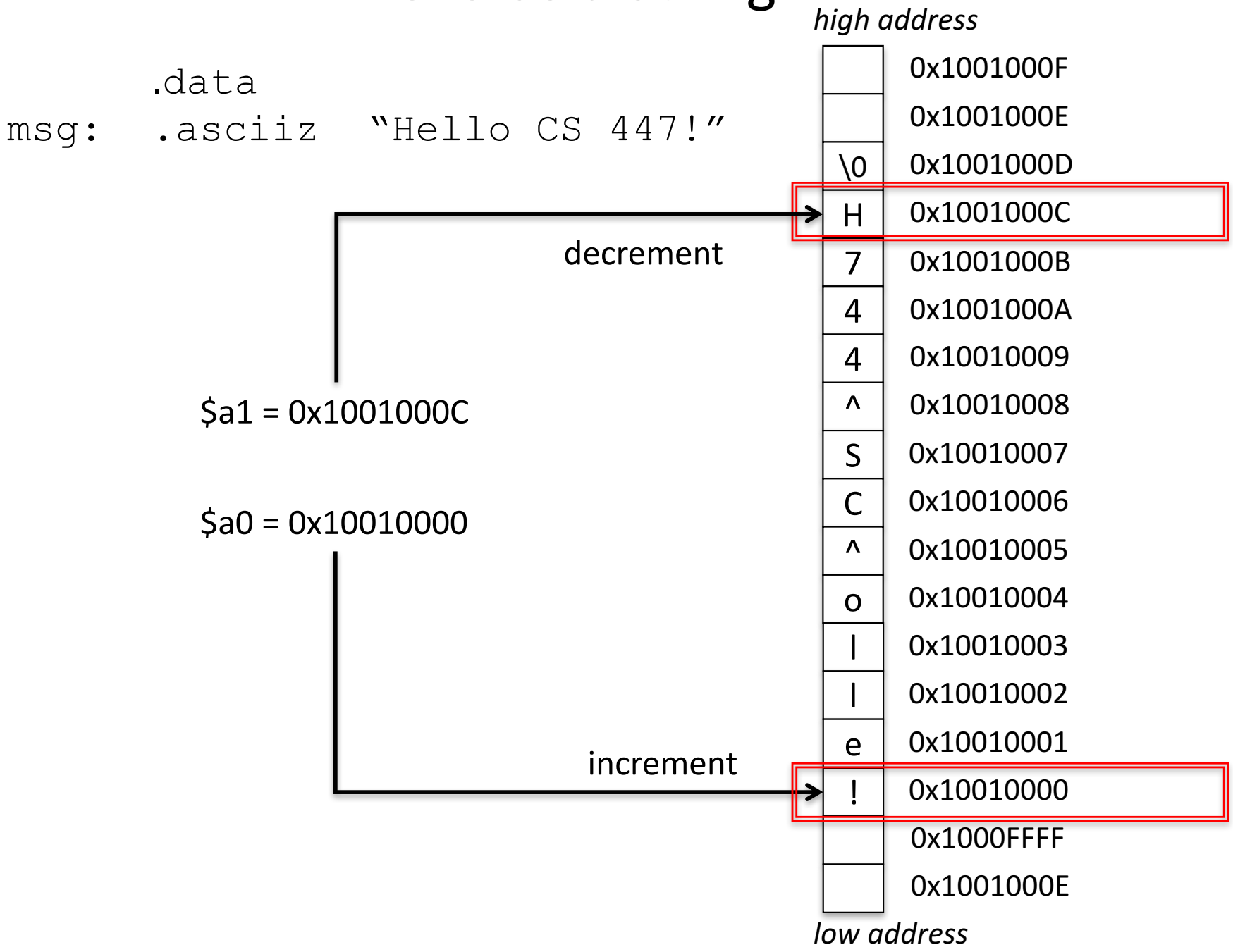
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



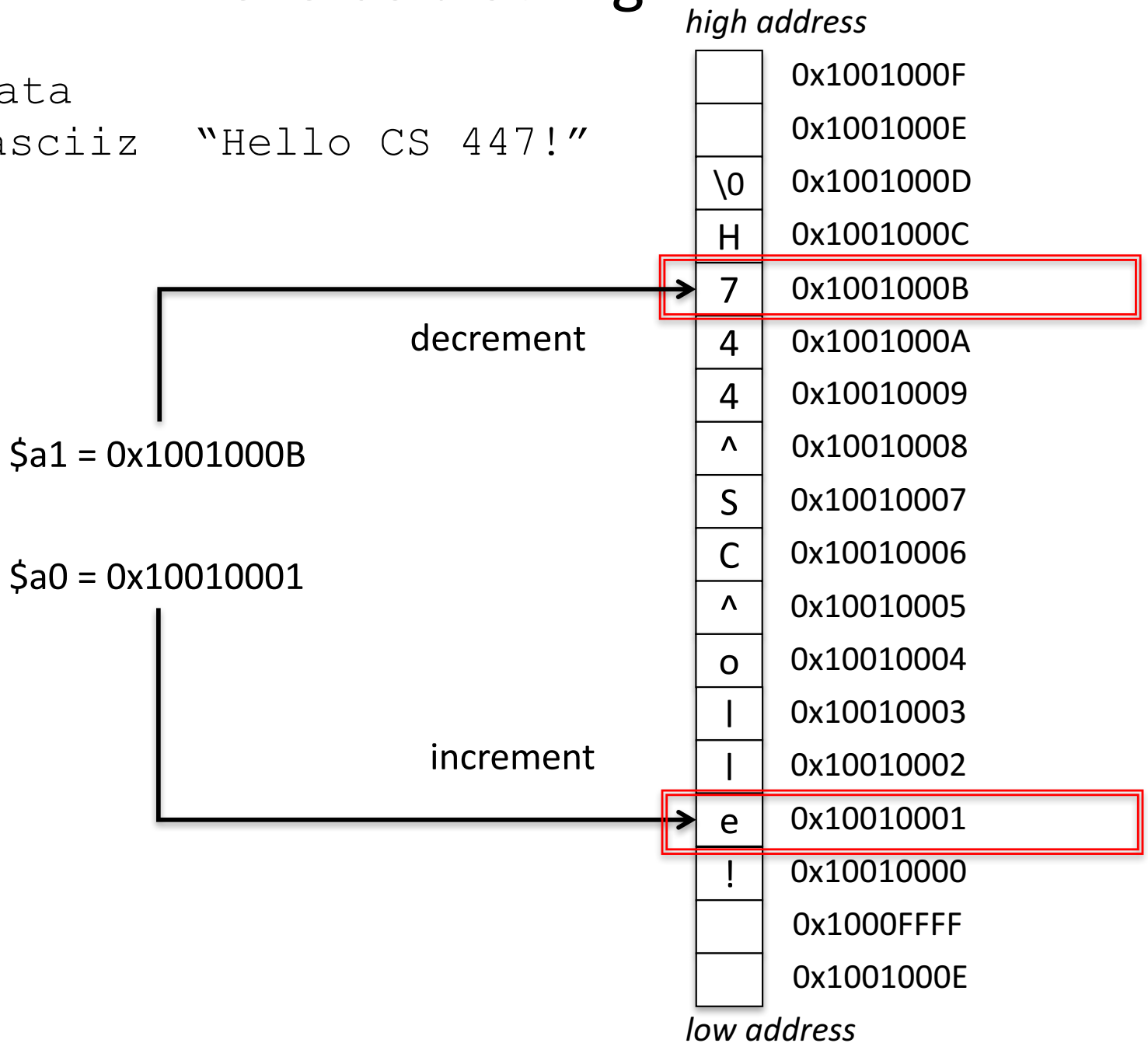
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



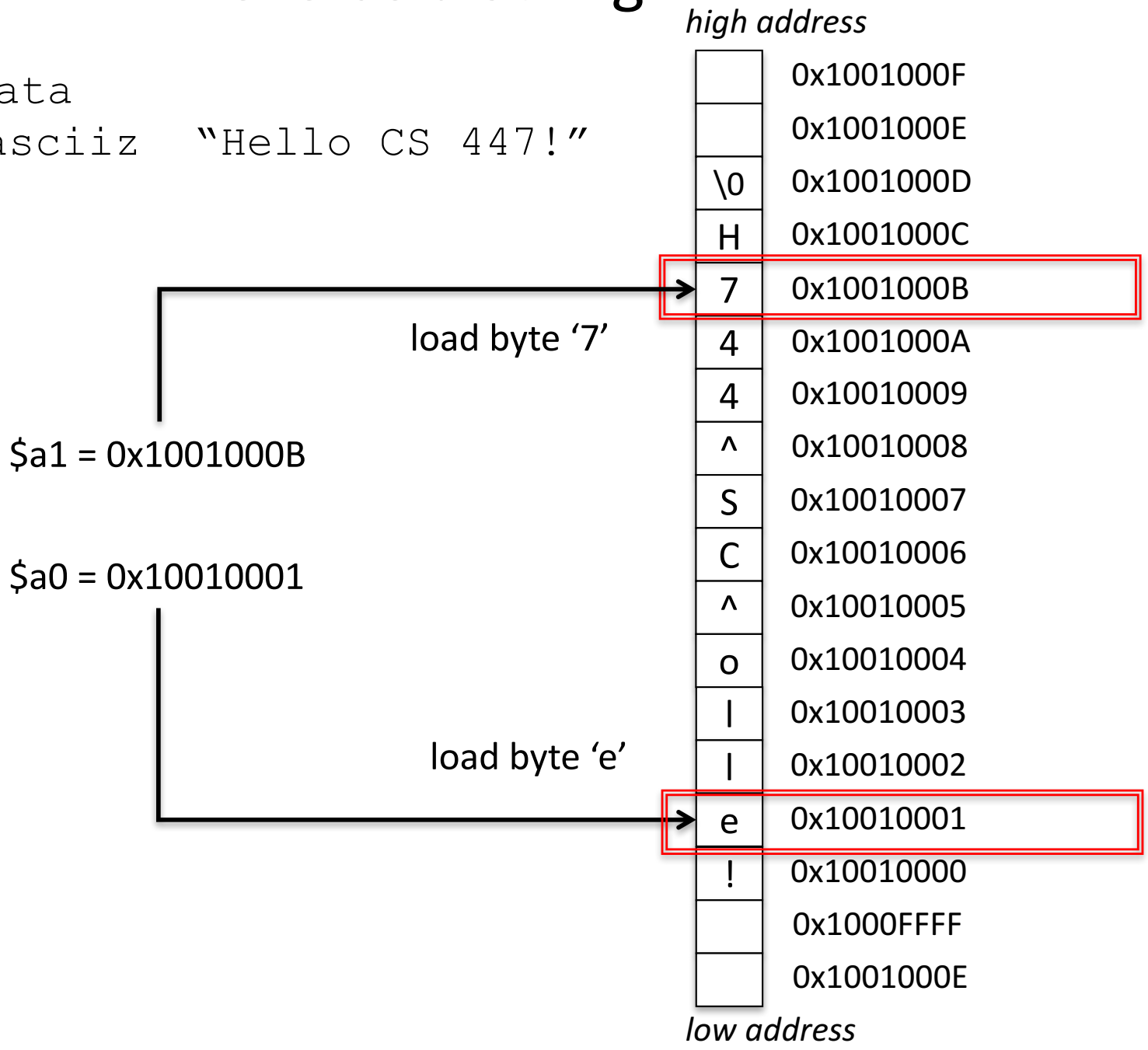
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



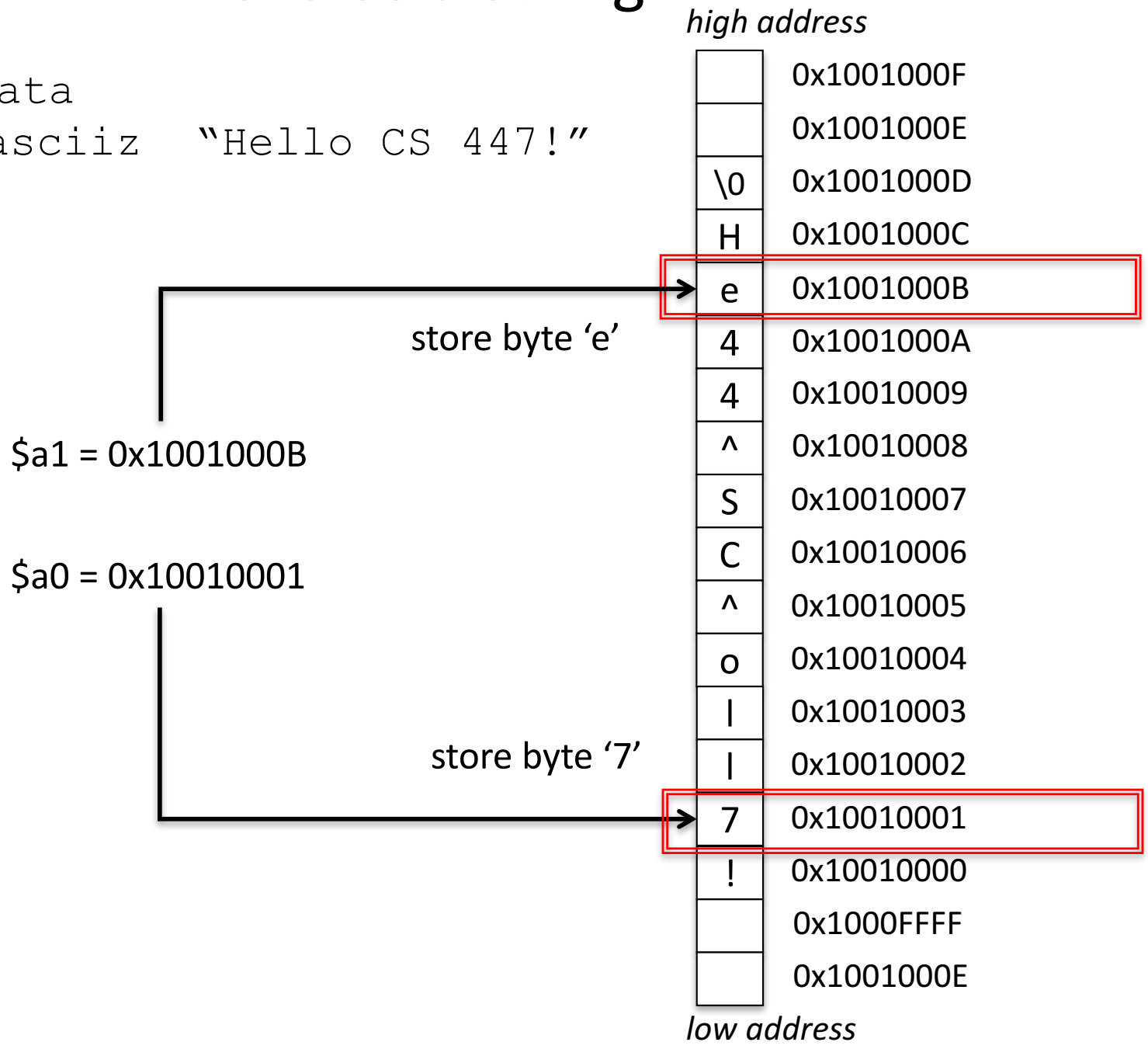
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



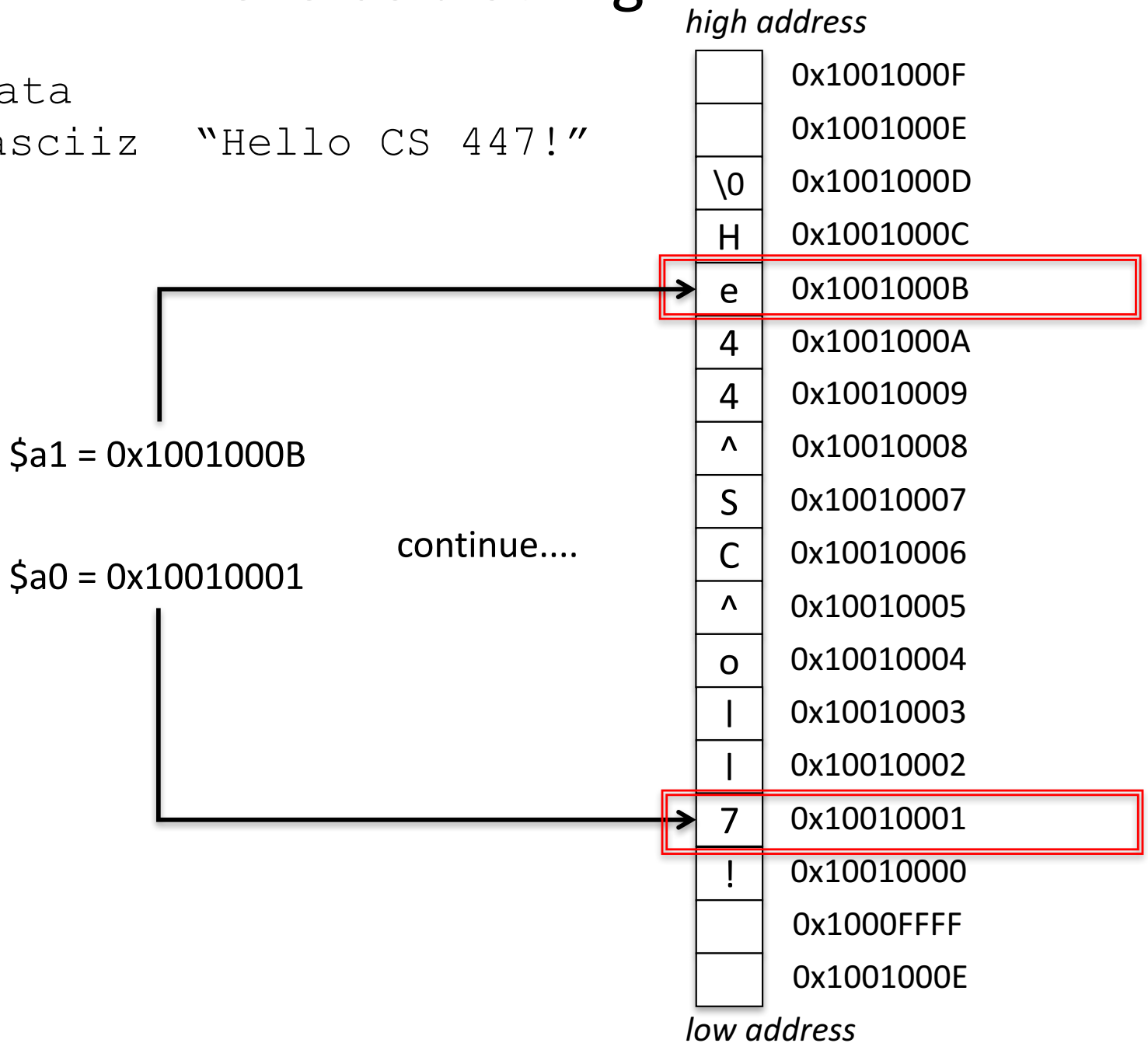
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



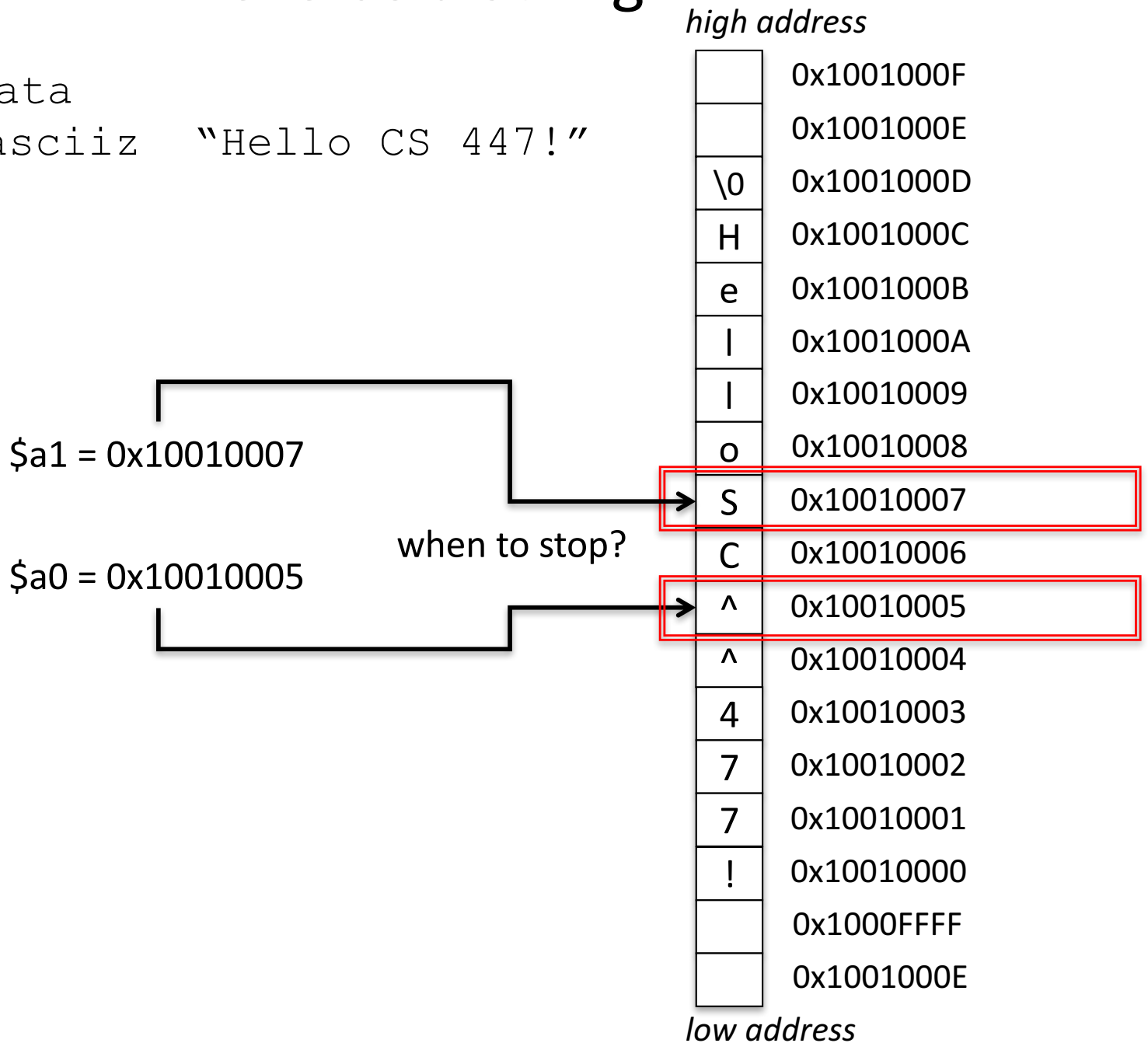
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



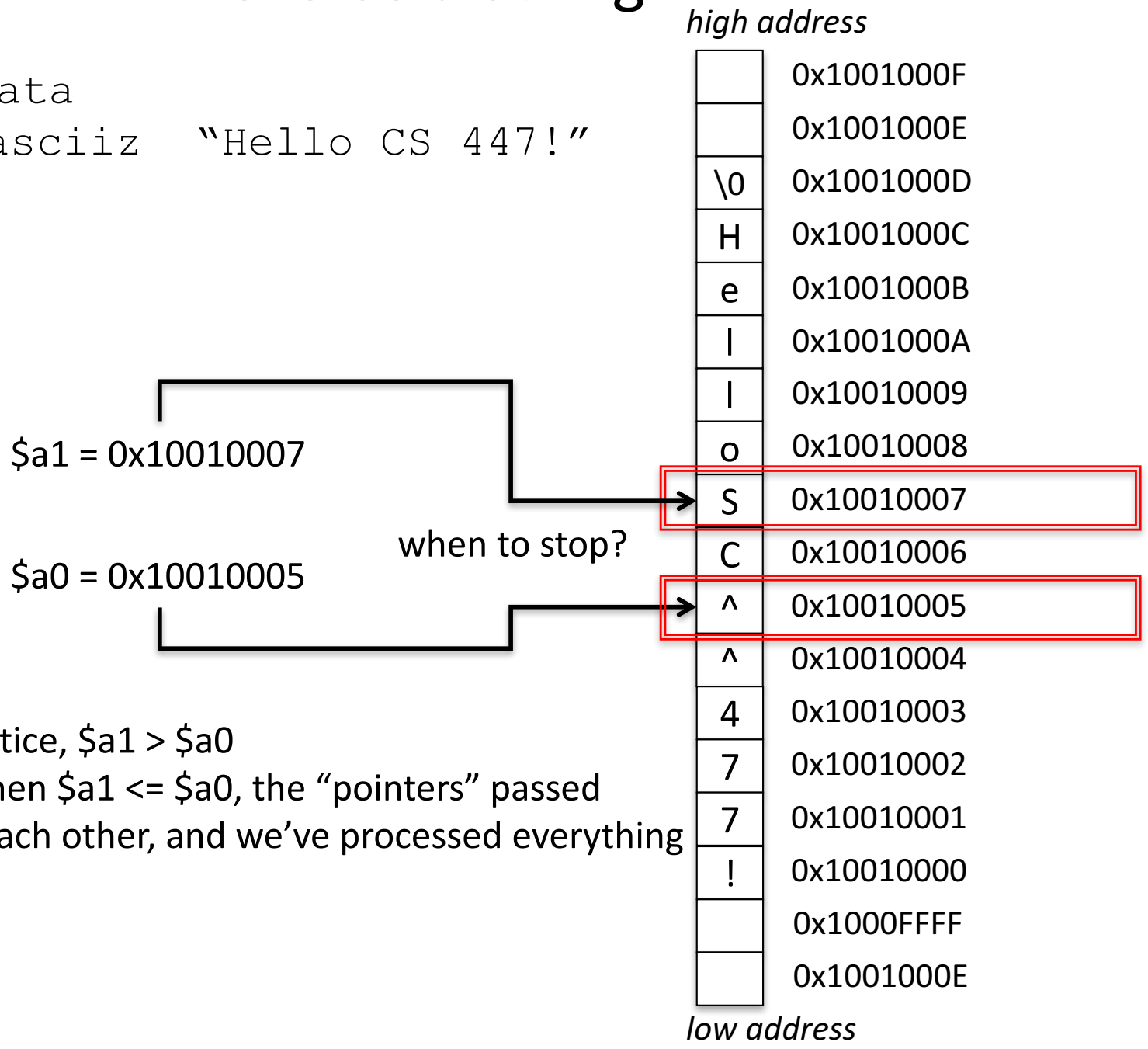
Reverse a String

```
        .data
msg:    .asciiz  "Hello CS 447!"
```



Reverse a String

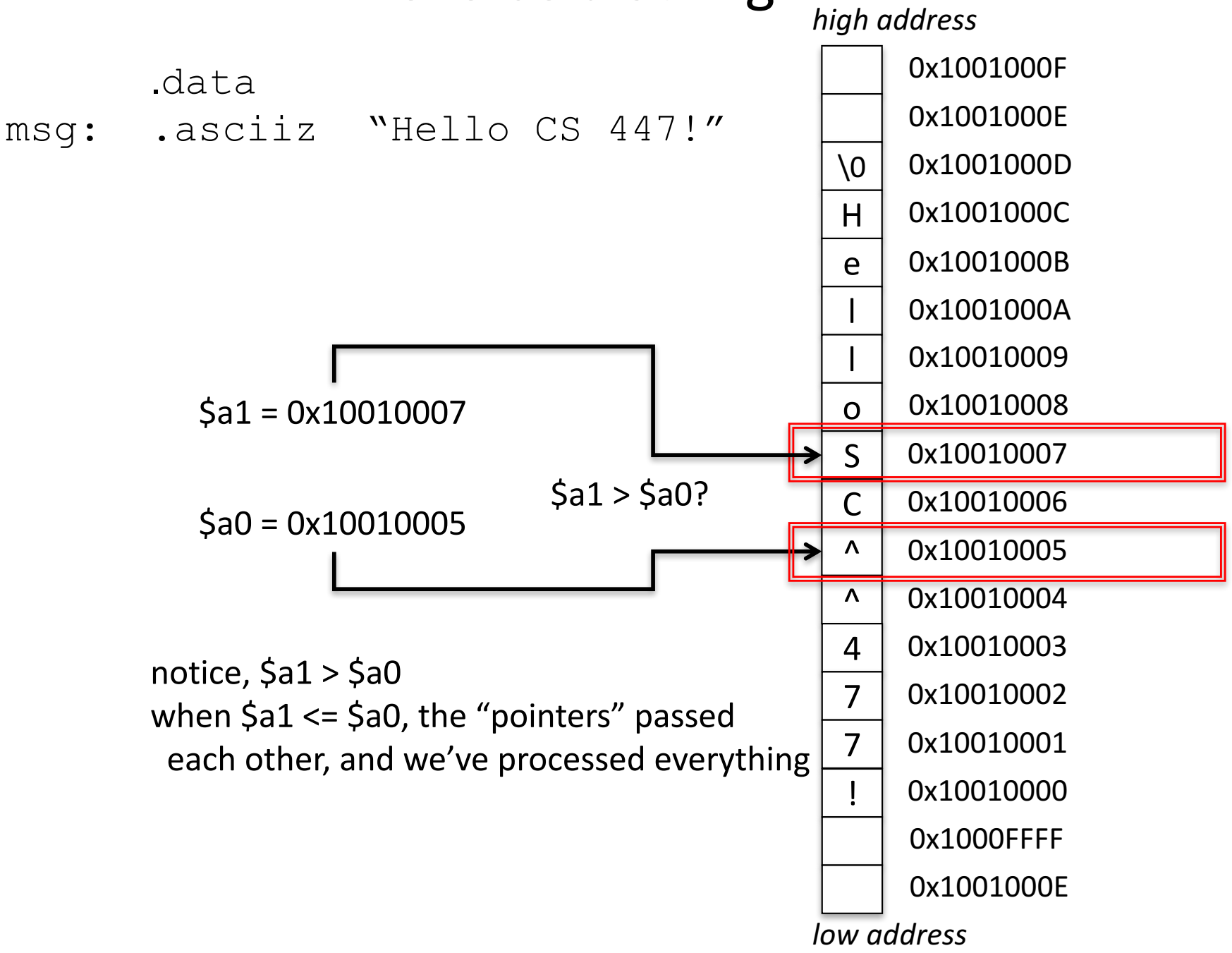
```
        .data
msg:    .asciiz  "Hello CS 447!"
```



notice, $a1 > $a0
when $a1 <= $a0, the "pointers" passed each other, and we've processed everything

Reverse a String

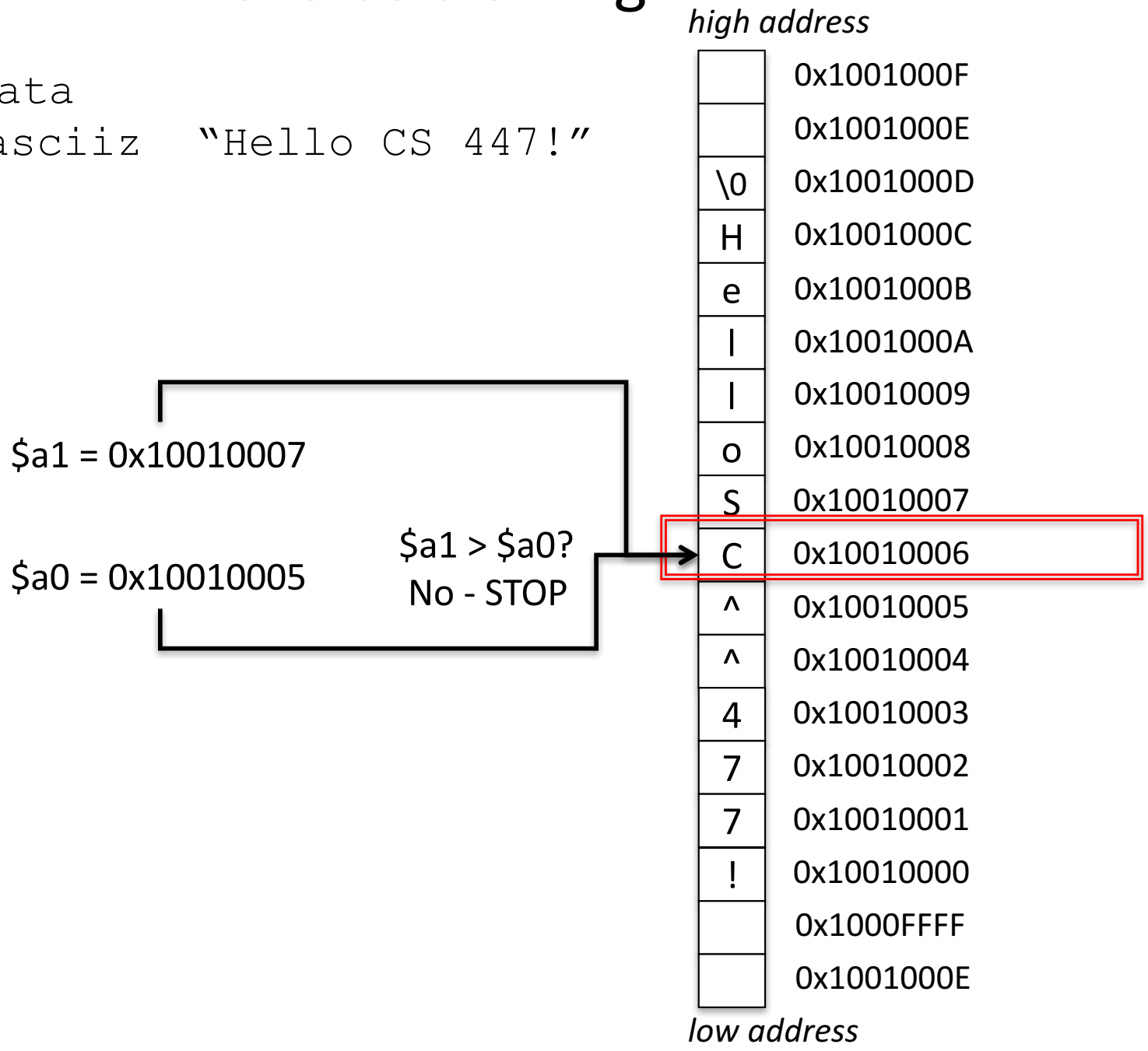
```
        .data
msg:    .asciiz  "Hello CS 447!"
```



notice, $a1 > $a0
when $a1 <= $a0, the "pointers" passed each other, and we've processed everything

Reverse a String

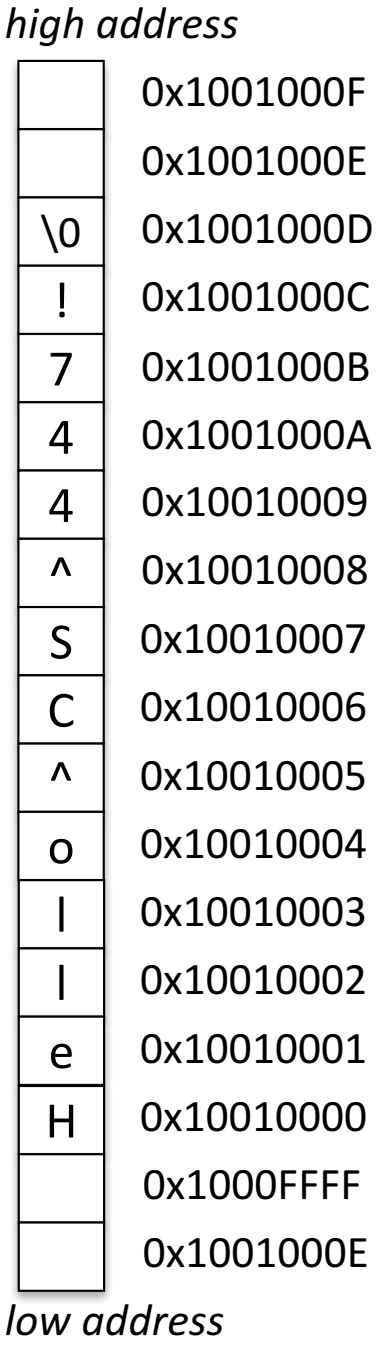
```
        .data
msg:    .asciiz  "Hello CS 447!"
```



Reverse a String

```
        .data
    msg: .asciiz  "Hello CS 447!"

        .text
    la   $a0,msg
    move $a1,$a0
    lb   $t0,0($a0)          # check for null string
    beq  $t0,$0,endreverse
findnull:
    lb   $t0,1($a1)          # one beyond position
    beq  $t0,$0,reverse
    addi $a1,$a1,1
    j    findnull
reverse:
    lb   $t0,0($a0)          # left char (low addr)
    lb   $t1,0($a1)          # right char (high addr)
    sb   $t1,0($a0)          # store in reverse
    sb   $t0,0($a1)          #    does the swap
    addi $a0,$a0,1           # move to right (to high)
    addi $a1,$a1,-1          # move to left (to low)
    bgt  $a1,$a0,reverse     # at boundary
    li   $v0,4               # print reversed string
    la   $a0,msg
    syscall
endreverse:                  # terminate program
    li   $v0,10
    syscall
```



Examples of control flow

- Weather programs
  - weather.c
  - weather.asm - illustrates if-then-else and while loop
  - weather2.asm - a slightly improved version (still if-then-else)
  - weather3.asm - illustrates a “computed goto” (switch)
  - weather4.asm - illustrates an algorithm change, using a table
- sam12.asm
  - convert a 32-bit number into hex characters, which are displayed with the OS print string service
- We’ll see many more examples!