a64strcpy

This documents details the strcpy function working for all alignments for ajit. We will introduce the basic flow of the program and then look at an example. We will then look at the testing methodology and the pseudocode will be given at the end.

a64strcpy – introduction

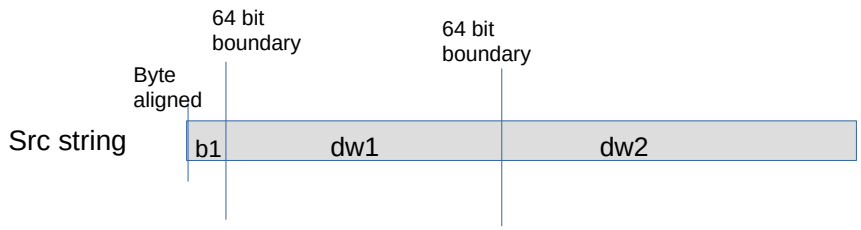
The a64strcpy function takes two inputs – a destination and a source string, and copies the source string into the destination string. The input can be of any alignment.

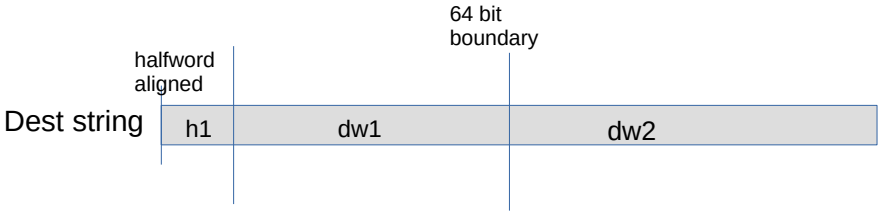
The logic used for the function can be summarized as follows -

1. We first check for alignment of source and destination
2. If both are aligned, we can move to direct 64 bit copies
3. Else if source is aligned we move to part of the function that deals with unaligned destination (step 6 onwards).
4. If the source is unaligned, we will first align it to 64 bits. We do this by copying the strng byte by byte till source is aligned. This will happen a maximum of 7 times (as there are only 8 bytes in a doubleword)
5. After we do this, destination alignement will have changed, check and if aligned go to 64 bit alignments else move ahead
6. We need to check destination alignment – word, halfword or byte
7. If it is word aligned
    1. store a word, making it 64 bit aligned
    2. save the unstored word in another register
    3. load a new doubleword
    4. create a doubleword with the unstored word from previous doubleword, and the most significant word from recently loaded doubeword (use shifts and or operations)
    5. load this doubleword at once in destination
    6. repeat from step2
8. If it is halfword or byte aligned – we follow similar steps as above, i.e., perform store till destination is aligned to 64 bit, save the rest of the part of doubleword. Load a new doubleword. Create the doubleword to be stored by combining required parts of previously saved and recemtly loaded doubleword and finally perform a doubleword store with it.

a64strcpy – example

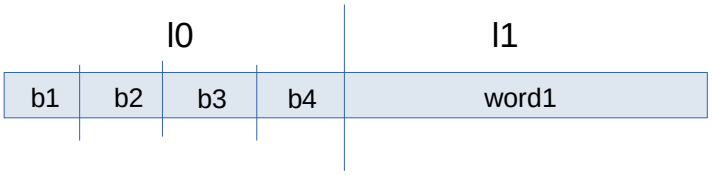
Assume – source address is byte aligned and destination address is halfword aligned



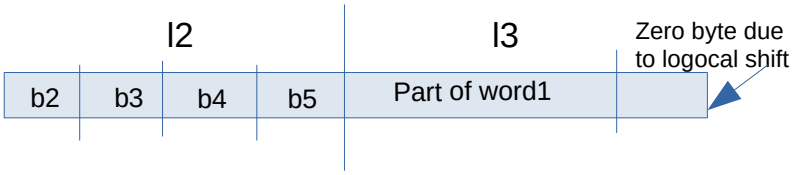


Step 1: Store byte b1 from source string to destination string at h1.
Now, source is doubleword aligned and destination is byte aligned as b1 will only fill half of h1.

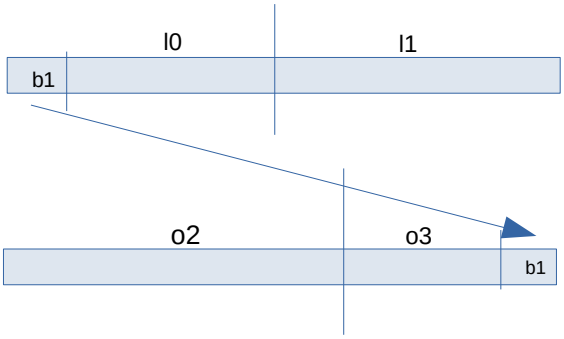
Step 2: Load a new doubleword from source in register pair. And load b1 into destination. This makes the destination doubleword aligned.



Step 3: Shift l0-l1 left by 8 bits and store in l2-l3. As we have stored b1 already, we do not need it.



Step 4: Load a new doubleword into l0-l1. We need the most significant byte from the new doublword. Hence we shift l0-l1 right by 7 bytes and save it in another register pair, o2-o3.



Step 5: Now we OR the two registers – l2-l3 and o2-o3.
We now have a doubleword we can store in the destination.

Step 6: Store the doubleword and and repeat from step2 till we see null byte.

a64strcpy – Testing

We need to test for all possible scenarios of alignment which will cover the whole code.

We start by defining a doubleword aligned source and destination. We then give the doubleword aligned source and destination as input.

For the second iteration, we input the doubleword aligned but we give source string address as doubleword aligned +1, i.e., byte aligned input. This will reduce the length of the string.
In this case, as we first align the source string to 64 bits, we would have to store a byte and then destination would be byte aligned and we would be able to test that scenario.

Hence, the misalignment in the source will be eventually reflected in the destination. As there are 8 bytes, if we add one 8 times to the source string, we would cover all alignments for source, and as a result for destination too. This would cover the whole code atleast once.

Pseudocode -

1. Define 64 bit aligned destination and source strings. -
2. Initialize for loop to loop 9 times
3. For first iteration – input aligned source and destination addresses
4. For further iterations – input aligned destination address, but input source address is (aligned address + iteration count)

This excludes the test for finding null byte at various locations as that will be covered in a different test.

a64strcpy - pseudocode

Pseudocode of the function is given below.

```
1. check alignment of src and dest
        if both aligned
            goto 64 bit string operation
        elsif src unaligned
            goto alignsrc
        else dest unaligned
            goto destuna


2. 64bit string operation -    ()
        load doubleword of src string
        check for zero byte
        if found
            goto checkzero
        else
            store doubleowrd to dest string
            goto 64 bit string operation


3. checkzero:  (checks which byte has zero and stores in dest string accordingly)
        shift most significant byte to least significant byte position
        test if it is zero
        if yes
            load it
            return
        repeat for all other bytes


4. alignsrc:    (aligns src string address to doubleowrd)
```

```
load a src string byte
check if its zero
if yes
        store it and return
check if source is doublword aligned
if no
        store byte in destination
        goto alignsrc
else check destination for doubleword alignment
if aligned
        goto 64bit string operation
else
        goto destinua
```

5. destinua: (here, we will have a doubleword aligned source and will perform the string operation)

```
load a doubleword from src string into l0 - l1
check for zero bytes
if yes
        goto check zero
        add 8 to dest string
else
        check alignment of destination
        if
                word aligned goto wordal
        elsif
                halfword aligned goto hwordal
        else
                goto bytealigned
```

6. wordal: (dest string is word aligned, source string is doublword aligned)

```
        store the word in l0 to destination
        add 4 to dest address
wordalag: shift l1 to l6
        load another doubleword from source string into l0-l1
        check for zero byte
        if no zero byte
                move l0 to l7
                store l6-l7 to destintaion
                goto wordalag to repeat process
        else
                store the word in l6 to destination string
                add 4 to destination address
                goto check zero
```

7. hwordal: (here, destination string is halfword aligned and source string is double word aligned)

```
        store the most significant halfword of l0 to destination string
        check if destination is dowrd aligned
        if yes
                goto hword2al
        else
                store next word in destination address
hword1al:
        shift the least significant halfword in l0 to most significant position in l6
        load another doubleword in l0
        shift l0 to the right by a halfword and store in o2
        or l6-l7 and o2-o3 and store in l6-l7
        check for zero
        if no
                store the double word in l6-l7
                goto hword1al
        else
                store halfoword in l6 into destination string
                goto check zero

hword2al:
        shift the top three halfwords in l0-l1 to l6-l7 in the same position
        load another doubleword in l0-l1
        shift the most significant hword in l0-l1 to least significant hword in o2-o3
        or l6-l7 and o2-o3 and store in l6-l7
        check for zero byte
        if no
                store l6-l7 into destination
                goto hword2al
        else
                store the three halfwords in l6-l7 into destination
                check for zero in l0-l1
```

8. bytealigned: (here destination string is byte aligned and source string is doubleword aligned)

```
        store the most significant byte to destination string
        check if destination is doubleword aligned
        if yes
                goto byte1
        store the next  halword to destination string
        check if destination is doubleword aligned
        if yes
                goto byte3
        store the next  halword to destination string
```

check if destination is doubleword aligned
if yes
goto byte5
store the next halfword to destination string

byte7:
shift the least significant byte of l0 into l6 at most significant position
load the next doubleword in l0
shift the top 7 bytes to l0-l1 by 8 and store into o2-o3
or l6-l7 and o2-o3 and store into l6-l7
check for zero byte
if no
store l6-l7 into destination and goto byte7
else
store the byte in l6 into destination
check for zero byte in l0-l1