S-87.3190 Computer Architecture

Computer architecture assignment E1

A subroutine, which searches for given byte pattern from an byte array.

Mikko Vestola
Student ID: xxxxx
Department: xxxx
Email: xxxxx

02.05.2007

Table of contents

1 Original assignment....1
2 Function of the subroutine....2
3 Description of the subroutine....3
4 Testing....6
4.1 Test case 1: Pattern in the middle of the array....7
4.2 Test case 2: The pattern and the array are the same....7
4.3 Test case 3: Search a byte found at the beginning of the byte array....7
4.4 Test case 4: Search a pattern found at the beginning of the array....8
4.5 Test case 5: Search a byte found at the end of the byte array....8
4.6 Test case 6: Search a pattern at the end of the byte array....8
4.7 Test case 7: Search a pattern that is not found in the array....8
4.8 Test case 8: Search a pattern that is not entirely found in the array....9
4.9 Test case 9: Search a pattern that is found two times from the array....9
4.10 Test case 10: Search a pattern which exceeds beyond the array....9
4.11 Test case 11: Search a pattern which is longer that the byte array....9
4.12 Test case 12: Search a pattern with abnormal size parameters....10
4.13 Test case 13: Search a pattern from array with size of 1....10
5 Run-time log file....10
6 Program code....10

1 Original assignment

```
################################################################################
#
#       Computer architecture assignment E1
#
################################################################################
#
#    Write a subroutine, which searches for given byte pattern from an
#    byte array.
#
#    The arguments of the subroutine are passed as follows:
#
#    $a0 contains the address of byte array
#    $a1 contains the length of array
#    $a2 contains the address of byte pattern
#    $a3 contains the length of pattern
#
#    Subroutine should not alter the array nor byte pattern.
#
#    The C-prototype of the subroutine is as follows:
#
#    int memmem(void *m1, int size1, void *m2, int size2);
#
#    The subroutine should return in register $v0:
#     1) the address of first match or
#     2) zero, if there were no matches.
#
################################################################################
#
# Main program to test memmem()
#
    .text
    .globl main
main:
# Create a stack frame for main program
    subu    $sp $sp,8
    sw      $ra 4($sp)  # return address
    sw      $fp 0($sp)  # old frame pointer
    addu    $fp $sp,8   # update frame pointer
#
# Get arguments and call the memmem
#
    la      $a0 mem_area
    li      $a1 16
    la      $a2 search
    li      $a3 4
    jal     memmem
    .data
mem_area:
    .byte    0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
search:
    .byte    5,6,7,8
    .text
#
# Check result
#
    la      $t0 mem_area+5
    bne     $v0,$t0 .failure1
#
```

```
# Insert your own tests here
#

#
# Tell user that memmem works..
#
    .data
.ok:    .asciiz "The memmem() passed all tests\n"
    .text
    li      $v0 4
    la      $a0 .ok
    syscall

#
# Exit from test program
#
.exit:
    li      $v0 10
    syscall

#
# Test failed
#
.failure1:
    .data
.fail1:     .asciiz " The memmem() failed test 1.\n"
    .text
    li      $v0 4
    la      $a0 .fail1
    syscall
    j       .exit

#
# The subroutine memmem - search for given memory area
#
    .text
    .globl  memmem
memmem:
    XXX
    jr      $ra

    .end
################################################################################
```

2 Function of the subroutine

The function of the subroutine is to search a given byte pattern from a byte array. The subroutine returns the memory address of the first occurrence of the byte pattern if it is found from the byte array. If there are more than one occurrences in the byte array, only the address of the first occurrence is returned. If the byte pattern is not found from the array, zero is returned. The C-prototype of the subroutine is as follows:

```
int memmem(void *m1, int size1, void *m2, int size2);
```

So the function takes a pointer to the byte array, size of that array, pointer to where the byte pattern begins and its length. The parameters are saved respectively to registers $a0, $a1, $a2 and $a3. The return value (the memory address) is saved to register $v0.

3 Description of the subroutine

Subroutine `memmem` searches a given byte pattern from a given byte array. For example, consider a byte array consisting from numbers *0,1,2,3,4,5,6,7* and a byte pattern *3,4,5*. Subroutine tries to search the byte pattern, which now is found, and returns the memory address where the byte pattern starts in the array (which in this case equals: address of the array + 3).

The array is just a line of bytes one after another. The pointer (or memory address) to the first byte of the array is given in parameters and also the size of the array (which tells how many bytes are in that array). Byte patterns is also just a series of bytes. A pointer to the first byte of the pattern and the pattern's length are given in the parameters.

So at first, the subroutine tries to find the first byte of the pattern from the byte array by comparing the first byte values. If the byte is not found, it compares the second byte of the array to the first byte of the pattern. This continues when the first byte of the pattern is found (or may not be found at all). When it is found, its memory address is saved and the subroutine compares the second byte from the byte pattern to the next byte of the byte array. This is continued when the byte pattern ends (when the pattern is entirely found) or the bytes don't match (when the program starts again to find the first byte of the pattern from the byte array).

The operation of the subroutine can be described by the following C-program:

```
#include <stdio.h>

/* This function searches the given byte pattern
 * (beginning from pointer given at parameter m2) from
 * the given byte array (beginning from pointer given at parameter m1).
 * Parameter size1 is the size of the byte array (m1) and
 * size2 is the length of the byte pattern (m2).
 * Returns 0 (NULL pointer's memory address) if no occurrence of
 * the byte patterns was found. Otherwise returns the memory address
 * of the beginning of the first occurrence of the byte pattern
 * in the byte array.
 */

int memmem(void *m1, int size1, void *m2, int size2) {

        /* Check that the sizes are reasonable */
```

```
        if (size2<=0 || size1<=0)
                return 0;

        /* Cast void pointers to char pointers,
         * (char is 8 bits = 1 byte). */
        char* bTable = (char*) m1;
        char* bPattern = (char*) m2;

        /* Copy pointer of the beginning of the byte pattern.*/
        char* bPatternBegin = bPattern;

        int i;  /* byte array iterator */
        int j = 0;  /* byte pattern iterator */

        /* The memory address of the first occurrence of
         * byte pattern is saved here. Initialized to zero
         * (NULL pointer's address).*/
        int v = 0;

        /* Loop the whole byte array through to find the
         * byte pattern. */
        for (i = 0; i< size1; i++) {

        /* If the remaining byte pattern is longer than the remaining
         * byte array, immediately return 0 since the pattern
         * can't be found. */
                if (size2-j > size1-i)
                        return 0;

                if (*bTable != *bPattern) {
                        /* If the corresponding values of byte array and
                         * byte pattern are not equal, move to next value
                         * in byte array and set the byte pattern iterator
                         * to 0 and also the pointer to the beginning on
                         * the byte pattern (so we try to find the firs
                         * byte of byte pattern from the byte array).*/
                        bTable++;
                        v=0;
                        j=0;
                        bPattern = bPatternBegin;
                } else {

                        /* If v is 0, that means that we have found the
                         * (possible) beginning of the byte pattern from
                         * the byte array. So lets save it's memory address
                         * (this must be returned if the byte pattern
                         * is entirely found from byte array). */
                        if (v==0) {
                                v=(int)&(*bTable);
                        }

                        /* If byte pattern's value matches the byte array's
                         * value, move the pointer to next cells and
                         * increment the byte pattern iterator (because we
                         * next compare the next value of the byte pattern
                         * */
                        bTable++;
                        bPattern++;
                        j++;

                        /* If j equals to size of the byte pattern,
```

```
                              * that means we have found the entire byte pattern
                              * from the byte array. So return with value v
                              * where the beginning index of the byte pattern
                              * in byte array is saved. */
                             if (j==size2)
                                     return v;

                     }  // end of for loop

          }

          /* If the program loops the whole array through and does
           * not return any memory value, this means that the given
           * byte patterns was not found from the byte array. */
          return 0;

}

int main (int argc, char *argv []) {

          /* Test the memmem function with these byte arrays
           * (strings are byte arrays, so pointers to the beginning
           * of the array). */
          char *byteArray = "123456789";
          char *pattern = "10";

          int occurence = memmem(byteArray, 9, pattern, 2);

          /* Count the index of byteArray where the byte pattern
           * starts. */
          int index = occurence-(int)&(*byteArray);

          /* Print the results */
          if (occurence != 0) {
                    printf("First occurence is at address %d (index %d)",
                    occurence, index);
          }
          else
                    printf("No byte pattern found!");

          return 0;
}
```

So the subroutine `memmem` corresponds to the function `int memmem(void *m1, int size1, void *m2, int size2)`. The function `main` runs the `memmem` function and prints the result of one test.

The search algorithm is as simple as going through the byte array byte by byte. The search algorithm stops when it sees that the byte pattern is longer than the remaining length of the byte array, so it doesn't search the whole array in vane. In this case, this algorithm is the best choice when the byte array is quite short, and one doesn't know that is the byte array ordered (when a binary search might be efficient to find the first byte of the byte pattern). No need to build a more complicated algorithm - “keep it simple” method works also in software as well as in hardware.

The actual subroutine, written in MIPS assembly language, uses the following registers:

- `$a0`: The pointer to the byte array given in parameters. Corresponds to parameter **`void *m1`** in C-code.
- `$a1`: The size of the byte array given in parameters. Corresponds to parameter **`int size1`** in C-code.
- `$a2`: The pointer to the byte pattern given in parameters. Corresponds to parameter **`void *m2`** in C-code.
- `$a3`: The size of the byte pattern given in parameters. Corresponds to parameter **`int size2`** in C-code.
- `$v0`: The memory address of the first occurrence of the byte pattern in byte array (or 0 if the pattern is not found in the byte array). Corresponds to variable **`v`** in C-code.
- `$t0`: The byte array iterator. Corresponds to variable **`i`** in C-code.
- `$t1`: The byte pattern iterator. Corresponds to variable **`j`** in C-code.
- `$t2`: The memory address of the byte array's cell which value is compared (initialized as `$a0`). Corresponds to pointer **`bTable`** in C-code.
- `$t3`: The memory address of the byte pattern's cell which value is compared (initialized as `$a2`). Corresponds to pointer **`bPattern`** in C-code.
- `$t4`: Temporal register, used to calculate if the remaining byte pattern is longer than the remaining byte array. Also used to load one byte from byte array. Corresponds to values **`size2-j`** and **`*bTable`** in C-code.
- `$t5`: Temporal register, used to calculate if the remaining byte pattern is longer than the remaining byte array. Also used to load one byte from byte pattern. Corresponds to values **`size1-i`** and **`*bPattern`** in C-code.

The subroutine does neither use any callee-saved registers (`$sX`) nor modify the argument registers (`$aX`). Therefore, a stack frame is not required.

4 Testing

The purpose of the testing is to verify that the subroutine operates correctly and that all the branches of the subroutine will be tested. It is reasonable to presume that the pointers to byte arrays ($a0 and $a2) and their lengths ($a1 and $a3) are correct. Subroutine can't by no means figure out that is the pointer pointing to right memory location or is the size of that array given right. So these can't be

verified and the callee must give these parameters right if it wants to get right results. However, the subroutine does do a check that the size parameters are greater than zero, so abnormal size parameters return always zero.

Thirteen different test cases were built. They were produced that they should cover all reasonable boundary cases. The test cases are performed at the beginning of the file one after another. The test just simply compares the memory address got as an output of the `memmem` subroutine (stored in $v0) and a hand written value (stored at $t0) which is the expected value. If the values are not the same, test fails. If one test fails, the program doesn't run the remaining test but exits the execution and prints out information about which test case failed.

4.1 Test case 1: Pattern in the middle of the array

This test case tries to search a byte pattern that is found at the middle of the array.

Byte array: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte array size: `16`

Byte pattern: `5,6,7,8`

Byte pattern size: `4`

Expected output: Pattern is found, $v0 = memory address of the byte array + 5

4.2 Test case 2: The pattern and the array are the same

This test case tries to search a byte pattern that is the same as the byte array:

Byte array: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte array size: `16`

Byte pattern: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte pattern size: `16`

Expected output: Pattern is found, $v0 = memory address of the byte array

4.3 Test case 3: Search a byte found at the beginning of the byte array

This test case tries to search only one byte that is found at the beginning of the byte array.

Byte array: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte array size: `16`

Byte pattern: `0`

Byte pattern size: `1`

Expected output: Pattern is found, $v0 = memory address of the byte array

4.4 Test case 4: Search a pattern found at the beginning of the array

This test case tries to search a pattern, length of three, that is found at the beginning of the array.

Byte array: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte array size: `16`

Byte pattern: `0,1,2`

Byte pattern size: `3`

Expected output: Pattern is found, $v0 = memory address of the byte array

4.5 Test case 5: Search a byte found at the end of the byte array

This test case tries to search only one byte that is found at the end of the byte array.

Byte array: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte array size: `16`

Byte pattern: `15`

Byte pattern size: `1`

Expected output: Pattern is found, $v0 = memory address of the byte array+15

4.6 Test case 6: Search a pattern at the end of the byte array

This test case tries to search a pattern, length of three, that ends at the end of the byte array.

Byte array: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte array size: `16`

Byte pattern: `13,14,15`

Byte pattern size: `3`

Expected output: Pattern is found, $v0 = memory address of the byte array+13

4.7 Test case 7: Search a pattern that is not found in the array

This test case tries to search a pattern that is not found from the byte array (none of the bytes in pattern are in the byte array).

Byte array: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte array size: `16`

Byte pattern: `16,17`

Byte pattern size: `2`

Expected output: Pattern is not found, $v0 = 0

4.8 Test case 8: Search a pattern that is not entirely found in the array

This test case tries to search a pattern that is not found from the byte array but some of the bytes in the pattern are found from the byte array.

Byte array: `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15`

Byte array size: `16`

Byte pattern: `4,5,6,8`

Byte pattern size: `4`

Expected output: Pattern is not found, $v0 = 0

4.9 Test case 9: Search a pattern that is found two times from the array

This test case tries to search a pattern that is found two times from the byte array. This test case also tests how the subroutine works with strings as parameters (strings are also just arrays of bytes where one byte represents one letter).

Byte array: `"This is test test string"`

Byte array size: `24`

Byte pattern: `"test"`

Byte pattern size: `4`

Expected output: Pattern is found, $v0 = memory address of the byte array+8

4.10 Test case 10: Search a pattern which exceeds beyond the array

This test case tries to search a pattern whose beginning is found at the end of the byte array but whose end exceeds beyond the array so the pattern is not found.

Byte array: `"This is test test string"`

Byte array size: `24`

Byte pattern: `"test string that fails"`

Byte pattern size: `22`

Expected output: Pattern is not found, $v0 = 0

4.11 Test case 11: Search a pattern which is longer that the byte array

This test case tries to search a pattern which is longer than the byte array.

Byte array: `"This is test test string"`

Byte array size: `24`

Byte pattern: `"This string is too long!!"`

Byte pattern size: `25`

Expected output: Pattern is not found, $v0 = 0

4.12 Test case 12: Search a pattern with abnormal size parameters

This test case is the same as test case 9, so the pattern is found from the array, but now the given size parameters are abnormal (they must be greater than zero) so the subroutine should return zero.

Byte array: `"This is test test string"`

Byte array size: `0`

Byte pattern: `"test"`

Byte pattern size: `0`

Expected output: Pattern is not found, $v0 = 0

4.13 Test case 13: Search a pattern from array with size of 1

This test case tries to search a pattern, length of one, from an array with size of one.

Byte array: `7`

Byte array size: `1`

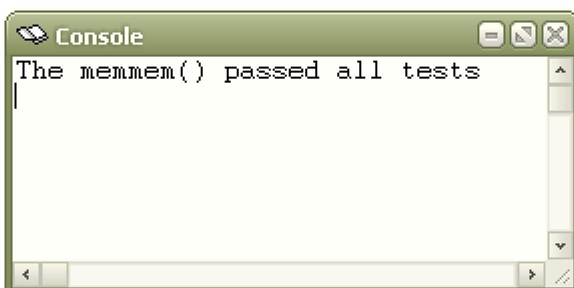
Byte pattern: `7`

Byte pattern size: `1`

Expected output: Pattern is found, $v0 = memory address of the byte array

5 Run-time log file

The following result was obtained while running the test program using SPIM on Windows XP. No test fails, so the `memmem` subroutine seems to be working OK.



Picture 1: Console log after running the test program

6 Program code

See the file `memmem.s` enclosed with the report. Also the C-prototype of the subroutine is enclosed as a file named `memmem.c`.