

# CSS422 Final Project

## Thumb-2 Implementation Work of Memory/Time-Related C Standard Library Functions.

### 1. Objective

You'll understand the following concepts at the ARM assembly language level through this final project that implements memory/time-related C standard library functions in Thump-2.

- CPU operating modes: user and supervisor modes
- System-call and interrupt handling procedures
- C to assembler argument passing (APCS: ARM Procedure Call Standard)
- Stack operations to implement recursions at the assembly language level
- Buddy memory allocation

This document is quite dense. Please read it as soon as the final project spec. becomes available to you.

### 2. Project Overview

Using the Thumb-2 assembly language, you will implement several functions of the C standard library that will be invoked from a C program named driver.c. See Table 1. These functions must be code in the Thumb-2 assembly language. Some of them can be implemented in stdlib.s running in the unprivileged thread mode (=user mode), whereas the others need to be implemented as supervisor calls, (i.e., in the handler mode = supervisor mode). For more details, log in one of the CSS Linux servers and type from the Linux shell:

**Assignment Project Exam Help**  
`man 3 function` where function is either bzero, strcpy, malloc, free, signal, or alarm

**Table 1: C standard lib functions to be implemented in the final project**

C standard lib functions <a href="https://tutorcs.com">https://tutorcs.com</a>	In stdlib.s *1	SVC *2
<code>bzero( void *s, size_t n )</code> writes n zeroed bytes to the setting s. If n is zero, bzero() does nothing.	Yes	
<code>strncpy(char *dst, const char *src, size_t len)</code> copies at most len characters from src into dst. It returns dst.	Yes	
<code>malloc( size_t size )</code> allocates size bytes of memory and returns a pointer to the allocated memory. If successful, it returns a pointer to allocated memory. Otherwise, it returns a NULL pointer.		Yes
<code>free( void *ptr )</code> Deallocates the memory allocation pointed to by ptr. If ptr is a NULL pointer, no operation is performed. If successful, it returns a pointer to allocated memory. Otherwise, it returns a NULL pointer.		Yes
<code>void (*signal( int sig, void (*func)(int)))(int);</code> Invokes the func procedure upon receipt of a signal. Our implementation focuses only on SIGALRM, (whose system call number is 14.)		Yes
<code>unsigned alarm( unsigned seconds )</code> sets a timer to deliver the signal SIGALRM to the calling process after the specified number of seconds. It returns the amount of time left on the timer from a previous call to alarm(). If no alarm is currently set, the return value is 0.		Yes

\*1: To be implemented in stdlib.s in the unprivileged thread mode

\*2: To be passed as an SVC to SVC\_Hander in the privileged handler mode

The driver.c we use is shown in listing 1. It tests all the above six stdlib functions. Please note that printf() in the code will be removed when you test your assembly implementation, because we won't implement the printf( ) standard function.

**Listing 1: driver.c program to test your implementation**

```
#include <strings.h> // bzero, strncpy
#include <stdlib.h> // malloc, free
#include <signal.h> // signal
#include <unistd.h> // alarm
#include <stdio.h> // printf

int* alarmed;

void sig_handler1( int signum ) {
    *alarmed = 2;
}

void sig_handler2( int signum ) {
    *alarmed = 3;
}

int main( ) {
    char stringA[40] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabc\0";
    char stringB[40];
    bzero( stringB, 40 );
    strncpy( stringB, stringA, 40 );
    bzero( stringA, 40 );
    printf( "%s\n", stringA );
    printf( "%s\n", stringB );

    void* mem1 = malloc( 1024 );
    void* mem2 = malloc( 1024 );
    void* mem3 = malloc( 512 );
    void* mem4 = malloc( 4096 );
    void* mem5 = malloc( 512 );
    void* mem6 = malloc( 1024 );
    void* mem7 = malloc( 512 );

    free( mem6 );
    free( mem5 );
    free( mem1 );
    free( mem7 );
    free( mem2 );

    void* mem8 = malloc( 4096 );

    free( mem4 );
    free( mem3 );
    free( mem8 );

    alarmed = (int *)malloc( 4 );
    *alarmed = 1;
    printf( "%d\n", *alarmed );

    signal( SIGALRM, sig_handler1 );
    alarm( 2 );
    while ( *alarmed != 2 ) {
        void* mem9 = malloc( 4 );
```

Assignment Project Exam Help

WeChat: cstutorcs

```

        free( mem9 );
    }
    printf( "%d\n", *alarmed );

    signal( SIGALRM, sig_handler2 );
    alarm( 3 );
    while ( *alarmed != 3 ) {
        void* mem9 = malloc( 4 );
        free( mem9 );
    }
    printf( "%d\n", *alarmed );

    return 0;
}

```

This driver program repeats allocating and deallocating memory space and thereafter sets the `sig_handler1()` function to be called upon receiving the first timer interrupt (in 2 seconds) and `sig_ahndler2()` function to be called upon the second timer interrupt (in 3 seconds).

### 3. System Overview and Execution Sequence

#### 3.1. Memory overview

This project maps all code to `0x0000.0000 – 0x1FFF.FFFF` in the ARM's usual ROM space (as the Keil C compiler/ARM assembler does) and defines a heap space; user and SVC stacks; memory control block (MCB) to manage the heap space; and all the SVC-related parameters over `0x2000.1000 – 0x2000.7FFF` in the ARM's usual RAM space. See table

**Table 2: Memory overview**

Address	Size(hex)	Size(B)	Usage
<code>0x400F.E600 – 0x400F.F028</code>	<code>0x0000.0428</code>	2KB	uDMA registers (memory mapped IO)
<code>0x2000.7C00 – 0x2000.7FFF</code>	<code>0x0000.0400</code>	1KB	uDMA memory map (ch 30)
<code>0x2000.7B80 – 0x2000.7BFF</code>	<code>0x0000.0080</code>	128B	System variables used by timer.s
<code>0x2000.7B00 – 0x2000.7B7F</code>	<code>0x0000.0080</code>	128B	System call table used by svc.s
<code>0x2000.6C00 – 0x2000.7AFF</code>	<code>0x0000.0F00</code>	3.8KB	Not used for now
<code>0x2000.6800 – 0x2000.6BFF</code>	<code>0x0000.0400</code>	1KB	Memory control block to manage in heap.s
<code>0x2000.6000 – 0x2000.67FF</code>	<code>0x0000.0800</code>	2KB	Not used for now.
<code>0x2000.5800 – 0x2000.5FFF</code>	<code>0x0000.0800</code>	2KB	SVC (handler) stack: used by all the others
<code>0x2000.5000 – 0x2000.57FF</code>	<code>0x0000.0800</code>	2KB	User (thread) stack: used by driver.c stdlib.s
<code>0x2000.1000 – 0x2000.4FFF</code>	<code>0x0000.4000</code>	16KB	Heap space controlled by malloc/free
<code>0x2000.0000 – 0x2000.0FFF</code>	<code>0x0000.1000</code>	4KB	Keil C compiler-reserved global data
<code>0x0000.0000 – 0x1FFF.FFFF</code>	<code>0x2000.0000</code>	512MB	ROM Space: all code mapped to this space

Since we compile `driver.c` together with our assembly programs, the Keil C compiler automatically reserves `driver.c`-related global data to some space within `0x2000.0000 – 0x2000.0FFF`, which makes it difficult for us to start Master Stack Pointer (MSP) exactly at `0x2000.6000` toward to the lower address as well as to start Process Stack Pointer (PSP) at `0x2000.5800`. So, it's sufficient to map MSP and PSP around `0x2000.6000` and `0x2000.5800` respectively. For the purpose of this memory allocation, you should declare the space as shown in listing 2:

**Listing 2: The memory space definition in Thump-2**

```

Heap_Size          EQU 0x00005000
AREA      HEAP, NOINIT, READWRITE, ALIGN=3
_heap_base
Heap_Mem
_heap_limit       SPACE  Heap_Size

```

```

Handler_Stack_Size    EQU    0x00000800
Thread_Stack_Size     EQU    0x00000800
AREA    STACK, NOINIT, READWRITE, ALIGN=3
Thread_Stack_Mem      SPACE   Thread_Stack_Size
__initial_user_sp
Handler_Stack_Mem    SPACE   Handler_Stack_Size
__initial_sp

```

### 3.2. Initialization, system call, and interrupt sequences

- (1) **Initialization:** the ARM processor reads the first 8 bytes to set MSP and the next 8 bytes to jump to the Reset\_Handler routine (as you studied in the class). You don't have to change the original vector table. Reset\_Handler initializes all the data structures you've developed and finally calls \_\_main with listing 3.

**Listing 3: The last two instructions in Reset\_Handler (startup\_TM4C129.s)**

```

LDR    R0, =__main
BX    R0

```

These last two statements are from the original startup\_TM4C129.s. Then, the main( ) function in driver.c is invoked.

- (2) **System calls:** whenever main( ) calls any of stdlib functions including bzero, strcpy, malloc, free, signal, and alarm, the control needs to move to stdlib.s. In other words, you need to define these functions' code in stdlib.s, as shown in Listing 4.

**Listing 4: The framework of stdlib.s**

```

AREA    .text!, CODE, READONLY, ALIGN=2
THUMB
EXPORT _bzero
_bzero
; Implement the body of bzero( )
MOV      pc, lr ; Return to main( )
EXPORT _strcpy
_strncpy
; Implement the body of strcpy( )
MOV      pc, lr ; Return to main( )
EXPORT _malloc
_malloc
; Invoke the SVC_Handler routine in startup_TM4C129.s
MOV      pc, lr ; Return to main( )
EXPORT _free
_free
; Invoke the SVC_Handler routine in startup_TM4C129.s
MOV      pc, lr ; Return to main( )
EXPORT _signal
_signal
; Invoke the SVC_Handler routine in startup_TM4C129.s
MOV      pc, lr ; Return to main( )
EXPORT _alarm
_alarm
; Invoke the SVC_Handler routine in startup_TM4C129.s
MOV      pc, lr ; Return to main( )
END

```

Among these six stdlib functions, you'll implement the entire logic of bzero( ) and strcpy( ) as they may be executed in the user mode. However, the other four functions must be handled as a system call. You need to invoke SVC\_Handler in startup\_TM4C129.s. Based on the Linux system call convention, use R7 to maintain the system call number. Arguments to a system call should follow ARM Procedure Call Standard, as summarized in table 3.

**Table 3: System Call Parameters**

System Call Name	R7	R0	R1
alarm	1	arg0: seconds	
signal	2	arg0: sig	arg1: func
malloc	3	arg0: size	
free	4	arg0: ptr	

SVC\_Handler must invoke \_systemcall\_table\_jump in svc.s. This in turn means you must prepare the svc.s file to implement \_systemcall\_table\_jump. This function initializes the system call table in \_systemcall\_table\_init as shown in Table 4:

**Table 4: System Call Jump Table**

Memory address	System Calls	Jump destination
0x2000.7B10	#4: free()	kfree in heap.s
0x2000.7B0C	#3: malloc()	kalloc in heap.s
0x2000.7B08	#2: signal()	signal_handler in timer.s
0x2000.7B04	#1: alarm()	timer_start in timer.s
0x2000.7B00	#0	Reserved

Each table entry records the routine to jump. For this purpose, svc.s needs to import the addresses of these routines using the code snippet shown in Listing 5.

Assignment Project Exam Help

WeChat: estutores

<https://tutorcs.com>

**Listing 5: Entry points to kernel functions imported in svc.s**

```

IMPORT _kfree
IMPORT _kalloc
IMPORT _signal_handler
IMPORT _timer_start

```

When called from SVC\_Handler, \_system\_call\_table\_jump checks R7, (i.e., the system call#) and refers to the corresponding jump table entry and invokes the actual routine. The merit of using svc.c is to minimize your modifications onto startup\_TM4C129.s.

- (3) **Interrupts:** This final project only handles SysTick interrupts. The SysTick timer gets started with \_timer\_start that was invoked when main( ) calls alarm( ). Note that SysTick timer can count down up to 1 second. Therefore, if main( ) calls alarm( 2 ) or alarm( 3 ), you'll get a SysTick interrupts at least twice or three times. Upon receiving a SysTick interrupt, the control jumps to SysTick\_Handler in startup\_TM4C129.s. The handler routine will invoke \_timer\_update in timer.s to decrement the count provided by alarm( ), to check if the count reached 0, and if so to stop the timer as well as invoke func specified by signal( SIG\_ALRM, func ).

### 3.3. Structure of your library implementation

The software components you need for this final project are summarized in table 5.

**Table 5: A summary of software components implemented in this final project**

Source files	Functions to implement	Control[1:0]	Functions/routines to call
driver.c	main()	11 User/PSP <sup>1</sup>	<ul style="list-style-type: none"> <li>→ bzero()</li> <li>→ strcpy()</li> <li>→ malloc()</li> <li>→ free()</li> <li>→ signal()</li> <li>→ alarm()</li> </ul>

stdlib.s	bzero( ): entirely implemented here strncpy( ): entirely implemented here  malloc( ): invokes an SVC free( ): invokes an SVC signal( ): invokes an SVC alarm( ): invokes and SVC	11 User/PSP <sup>*1</sup>	→ SVC_Handler → SVC_Handler → SVC_Handler → SVC_Handler
startup_TM4C129.s	Reset_Handler	00 PriThr/MSP <sup>*2</sup>	→ _kinit → _systemcall_table_init → _timer_init → __main
	SVC_Handler	00 Handler/MSP <sup>*3</sup>	→ _systemcall_table_jump
	SysTick_Handler	00 Handler/MSP <sup>*3</sup>	→ timer_update
svc.s	_systemcall_table_init: see 3.2.(2) _systemcall_table_jump: see 3.2.(2)	00 Handler/MSP <sup>*3</sup>	→ _kalloc → _free → _signal_handler → timer_start
timer.s	timer_init: initializes SysTick here timer_update: see 3.2.(3) timer_start: see 3.2.(3) signal handler: see 3.2.(3)	00 Handler/MSP <sup>*3</sup>	Assignment Project Exam Help
heap.s	kmalloc: initializes memory structures. _kalloc: buddy allocation coded kfree: buddy de-allocation coded	00 Handler/MSP <sup>*3</sup>	

\*1: running under the unprivileged thread mode, using process stack pointer

\*2: running under the privileged thread mode, using master stack pointer

\*3: running under the privileged handler mode, using master stack pointer

#### 4. Buddy Memory Allocation and Test Scenario

The final project implements the buddy memory allocation in Thump-2.

##### 4.1. Algorithms

If you have already taken CSS430: Operating Systems, have your OS textbook in your hand and read Section 10.8.1 Buddy System. Since the CSS ordinary course sequence assumes CSS422 taken before CSS430, here is a copy of Section 10.8.1:

###### 10.8.1 Buddy System

The buddy system allocates memory from a fixed-size segment consisting of physically contiguous pages. Memory is allocated from this segment using a power-of-2 allocator, which satisfies requests in units sized as a power of 2 (4 KB, 8 KB, 16 KB, and so forth). A request in units not appropriately sized is rounded up to the next highest power of 2. For example, a request for 11 KB is satisfied with a 16-KB segment.

Let's consider a simple example. Assume the size of a memory segment is initially 256 KB and the kernel requests 21 KB of memory. The segment is initially divided into two buddies—which we will call AL and AR—each 128 KB in size. One of these buddies is further divided into two 64-KB buddies—BL and BR. However, the next-highest power of 2 from 21 KB is 32 KB so either BL or BR is again divided into

two 32-KB buddies, CL and CR. One of these buddies is used to satisfy the 21-KB request. This scheme is illustrated in Figure 10.26, where CL is the segment allocated to the 21-KB request.

An advantage of the buddy system is how quickly adjacent buddies can be combined to form larger segments using a technique known as coalescing. In Figure 10.26, for example, when the kernel releases the CL unit it was allocated, the system can coalesce CL and CR into a 64-KB segment. This segment, BL, can in turn be coalesced with its buddy BR to form a 128-KB segment. Ultimately, we can end up with the original 256-KB segment.

The obvious drawback to the buddy system is that rounding up to the next highest power of 2 is very likely to cause fragmentation within allocated segments. For example, a 33-KB request can only be satisfied with a 64-KB segment. In fact, we cannot guarantee that less than 50 percent of the allocated unit will be wasted due to internal fragmentation. In the following section, we explore a memory allocation scheme where no space is lost due to fragmentation.

physically contiguous pages

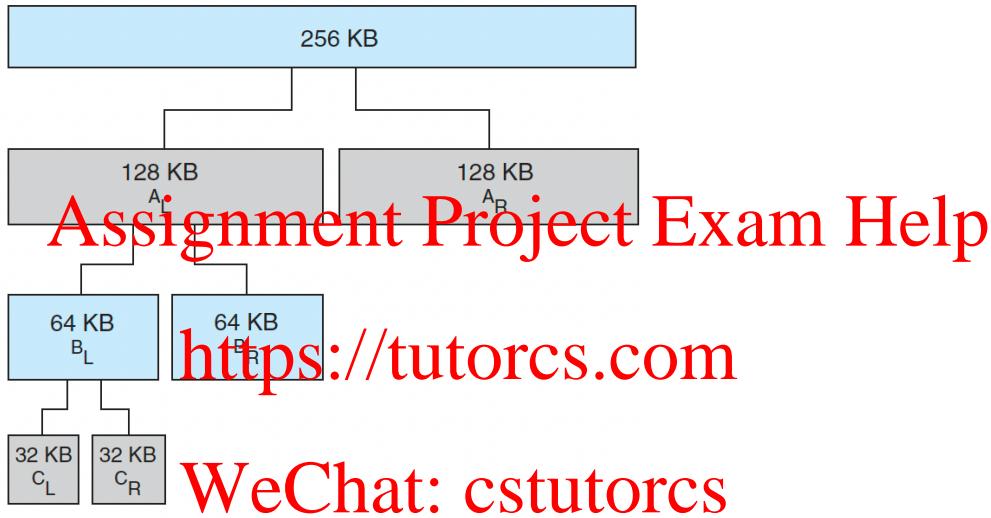


Figure 10.26 Buddy system allocation.

#### 4.2. Implementation over 0x20001000 – 0x20004FFF

As the memory range we use is 0x20001000 – 0x20004FFF, the entire contiguous size is 16KB. This space will be recursively divided into 2 subspaces of 8KB, each further divided into 2 pieces of 4KB, all the way to 32B. Therefore, one extreme allocates 16KB entirely at once, whereas the other extreme allocates 512 different spaces, each with 32 bytes. To address this finest case, (i.e., handling 512 spaces), we allocate a memory control block (MCB) of 512 entries, each with 2 bytes, in the 1KB space over 10x20006800 – 0x20006BFF. Each entry corresponds to a different 32-byte heap space. For instance, let MCB entries are defined as

```
short mcb[512];
```

Then, mcb[0] points to the heaps space at 0x20001000, whereas mcb[511] corresponds to 0x20004FE0. However, each mcb[i] does not have to manage only 32 bytes. It can manage up to a contiguous 16KB space. Therefore, each mcb[i] has the size information of a heap space it is currently managing. The size can be 32 bytes to 16KB and thus be represented with 5 to 16 bits, in other words with mcb[i]'s bits #15 - #4. We also use mcb[i]'s LSB, (i.e., bit #0) to indicate if the given heap space is available (= 0) or in use (= 1). Table 6 shows each mcb[i]'s bit usage:

**Table 6: each mcb entry's bit usage**

bits	descriptions
#15 – #4	The heap size this mcb entry is currently managing
#3 – #1	Reserved
#0	0: available, 1: in use

Let's consider a simple memory allocation scenario where main( ) requests 4KB and thereafter 8KB heap spaces with malloc( 4096 ) and malloc( 8192 ). Based on the buddy system algorithm, this scenario allocates 0x20001000 – 0x20001FFF for the first 4KB request and 0x20003000 – 0x20004FFF for the second 8KB request. Figure 1 shows this allocation. Only mcb[0], mcb[128], and mcb[256] are used to indicate in-use or available spaces. All the other mcb entries are not used yet.

Heap Address	Memory Availability	MCB	MCB Address	Contents
0x20001000 – 0x20001FFF	4KB in use	mcb[0]	0x20006800	4097 <sub>10</sub> (0x1001)
0x20002000 – 0x20002FFF	4KB available	mcb[128]	0x20006900	4096 <sub>10</sub> (0x1000)
0x20003000 – 0x20003FFF	8KB in use	mcb[256]	0x20006A00	8193 <sub>10</sub> (0x2001)
0x20004000 – 0x20004FFF				

**Figure 1: heap space and mcb contents**

#### 4.3. Implementation

For each implementation of `_kinit`, `_kalloc`, and `_kfree`, refer to figure 1 that illustrates how mcb entries are updated.

- (1) **`_kinit`:** The initialization must writes 16384<sub>10</sub> (0x4000) onto mcb[0] at 0x20006800-0x20006801, indicating that the entire 16KB space is available. All the other mcb entries from 0x20006802 to 0x20006BFE must be left initialized (step 1 in figure 1).
- (2) **`_kalloc`:** Your implementation must use recursions. When `_kalloc( size )` is called with a size requested, it should call a helper function, say `_ralloc`, as recursively choosing the left half or the right half of the current range until the requested size fits in a halved range. For instance in figure 1, the first `malloc( 4096 )` call is relayed to `_kalloc( 4096 )` that then calls `_ralloc( 4096, mcb[0], mcb[511] )` or `_ralloc( 4096, 20006800, 20006BFE )`. See step 2 in figure 2. The `_ralloc` call finds mcb[0] at 0x20006800 has 16384B available, halves it, and chooses the left half by calling itself with `_ralloc( 4096, mcb[0], mcb[255] )` or `_ralloc( 4096, 20006800, 200069FE )`. At this time, make sure that the right half managed by mcb[256] at 0x20006A00 must be updated with 8192 as its available space (step 3). Since the range is still 8192 bytes > 4096 bytes, `_ralloc` chooses the left by calling itself with `_ralloc( 4096, mcb[0], mcb[127] )` or `_ralloc( 4096, 20006800, 200068FE )`. Make sure that the right half managed by mcb[128] at 0x20006900 is updated to 4096. The left half in the range between mcb[0]-mcb[127] or 0x20006800-200068FF fits the requested size of 4096. Therefore, `ralloc( )` records 4097<sub>10</sub> (0x1001) into mcb[0] at 0x20006800-0x20006801. This is step 4 in figure 2.

The second `malloc( 8192 )` is handled as follows: `_kalloc( 8192 )` calls `_ralloc( 8192, mcb[0], mcb[511] )` or `_ralloc( 8192, 20006800, 20006BFE )` as in step 5 that needs to choose the right half with `_ralloc( 8192, 20006A00, 20006BFE )`, because mcb[0] at 0x20006800-0x20006801 has a value of 4097 indicating that the left half (20006800 – 200069FE) is in use. Since mcb[256] at 0x20006A00-0x20006A01 is available, `_ralloc` saves 8193 (0x2001) there (step 6).

- (3) **`_kfree`:** Your `_kfree` implementation must use recursions, too. The `_kfree( *ptr )` function calls a helper function, `_rfree( the corresponding mcb[] )`. If `main( )` calls `free( 20001000 )`, it is relayed to `_kfree( 20001000 )` that calls `_rfree( mcb[0] )` or `_rfree( 20006800 )` to reset its bit #0 from in-use to available (step 7). Then, check its right buddy at mcb[128] (or 0x20006900). If its bit #0 is 0, indicating the availability, zero-reinitialize mcb[128] at 0x20006900 and make sure that mcb[0]

at 0x20006800 shows an availability of 8192 bytes (step 8). Recursively check the buddy at higher layers. So, the next higher layer's buddy is mcb[256]-mcb[511] at 0x2006A00-0x2006BFE. Check mcb[256]'s contents, (at 0x20006A00-0x20006A01). In figure 2, the content is 8193 or (0x2001), showing that 8KB is being occupied. Therefore, stop `_kfree`'s recursive calls.

	step 1 <code>_kinit( )</code>	step 2 <code>_kalloc(4096)</code>	step 3 <code>ralloc(4096, 2006800, 2006BFE)</code>	step 4 <code>ralloc(4096, 2006800, 20069FE)</code>	step 5 <code>_kalloc(8192)</code>	step 6 <code>ralloc(8192, 2006800, 20068FE)</code>	step 7 <code>kfree(20001000)</code>	step 8 <code>rfree(20006800)</code>	<code>recursive_rfree(20006800)</code>
mcb[1]	MCB Address								
mcb[0]	0x20006800	0x4000	0x4000	0x2000	0x1001	0x1001	0x1000	0x2000	
:		0x0000	0x0000						
mcb[127]	0x200068FE	0x0000	0x0000						
mcb[128]	0x20006900	0x0000	0x0000		0x1000	0x1000	0x1000	0x0000	
:		0x0000	0x0000						
mcb[255]	0x200069FE	0x0000	0x0000						
mcb[256]	0x20006A00	0x0000	0x0000	0x2000	0x2000	0x2000	0x2001	0x2001	
:		0x0000	0x0000						
mcb[383]	0x20006A0E	0x0000	0x0000						
mcb[384]	0x20006B00	0x0000	0x0000						
:		0x0000	0x0000						
mcb[511]	0x20006BFE	0x0000	0x0000						

Figure 2: Recursive `_ralloc/_rfree` calls, each updating mcb entries

#### 4.4. Test Scenario

Looking back to listing 1. “driver.c”, you are supposed to verify your Thump-2 implementation of `malloc()` and `free()` with repetitive system call invocations that allocate/deallocate mem1 – mem8 spaces. Figure 2 illustrates how the heap space is allocated and deallocated when you run driver.c. Orange indicates allocated spaces and green means de-allocated spaces.

Heap Address	malloc 1	malloc 2	malloc 3	malloc 4	malloc 5	malloc 6	malloc 7	free 6	free 5	free 1	free 7	free 2	malloc 8	free 4	free 3	free 8	496 MCB Address
20001000																	20006800
20001200																	20006840
20001400																	20006860
20001600																	20006880
20001800																	200068A0
20001A00																	200068C0
20001C00																	200068E0
20001E00																	20006900
20002000																	20006920
20002200																	20006940
20002400																	20006960
20002600																	20006980
20002800																	200069A0
20002A00																	200069C0
20002C00																	200069E0
20002E00																	20006A00
20003000																	20006A20
20003200																	20006A40
20003400																	20006A60
20003600																	20006A80
20003800																	20006AA0
20003A00																	20006AC0
20003C00																	20006AE0
20003E00																	20006B00
20004000																	20006B20
20004200																	20006B40
20004400																	20006B60
20004600																	20006B80
20004800																	20006BA0
20004A00																	20006BC0
20004C00																	20006BE0
20004E00																	

Figure 2: Test scenario and memory allocation

#### 5. Signal and Alarm

The time management you will implement in your final project includes `signal( sig, *func )` and `alarm( seconds )`. The parameters `*func` and `seconds` should be memorized in memory address at 0x20007B84 and 0x20007B80, as shown in table 7.

Table 7: Signal/alarm parameters to be stored in memory

Memory address	Parameters to store
0x2000.7B84	<code>*func</code>
0x2000.7B80	<code>seconds</code>

### 5.1. SysTick Initialization

The ARM system timer, SysTick's description can be found at:

<https://developer.arm.com/documentation/dui0552/a/cortex-m3-peripherals/system-timer--systick>

Table 8 is a copy of Table 4.32. System timer register summary on that URL. Among four SysTick registers, you will use the first three registers: (1) SysTick Control and Status Register, (2) SysTick Reload Value Register, and (3) SysTick Current Value Register.

**Table 8: A copy from Cortex-M3 Devices Generic User Guide URL's Table 4.32.**

*Table 4.32. System timer registers summary*

Address	Name	Type	Required privilege	Reset value	Description
0xE000E010	SYST_CSR	RW	Privileged	[a]	<i>SysTick Control and Status Register</i>
0xE000E014	SYST_RVR	RW	Privileged	UNKNOWN	<i>SysTick Reload Value Register</i>
0xE000E018	SYST_CVR	RW	Privileged	UNKNOWN	<i>SysTick Current Value Register</i>
0xE000E01C	SYST_CALIB	RO	Privileged	-[a]	<i>SysTick Calibration Value Register</i>

[a] See the register description for more information.

Please click each register's hyperlink from table 4.32 to understand how the SysTick registers work.

<https://tutorcs.com>

For initialization in \_timer\_init

- (1) Make sure to stop SysTick:  
Set SYST\_CSR's Bit 2 (CLK\_SRC) = 1, Bit 1 (INT\_EN) = 0, Bit 0 (ENABLE) = 0
- (2) Load the maximum value to SYST\_RVR.  
The value should be 0x00FFFFFF which means MAX Value = 1/16MHz \* 16M = 1 second

### 5.2. Signal

The signal( sig, \*func ) function assumes only SIG\_ALRM as the sig argument, while it accepts any address of \*func (Keil C compiler automatically maps to memory). These sig and \*func arguments must be relayed from signal(sig, \*func) all the way to \_signal\_handler in timer.s as keeping sig in R0 and \*func in R1 respectively (based on APCS, see table 3). If R0 is SIG\_ALRM, (i.e., 14), save it in memory address at 0x20007B84. Return the previous value of 0x20007B84 to main( ) through R0.

### 5.3. Alarm

The alarm( seconds ) function relays this seconds argument in R0 from main( ) all the way to \_timer\_start in timer.s. Retrieve the previous value at 0x20007B80 that is recognized as the previous time value and returned to main( ) through R0, save the new seconds value to 0x20007B80, and start the SysTick timer.

- (1) Retrieve the seconds parameter from memory address 0x20007B80, which is the previous time value and should be returned to main( ).
- (2) Save a new seconds parameter from alarm( ) to memory address 0x20007B80.
- (3) Enable SysTick:  
Set SYST\_CSR's Bit 2 (CLK\_SRC) = 1, Bit 1 (INT\_EN) = 1, Bit 0 (ENABLE) = 1
- (4) Clear SYST\_CVR:  
Set 0x00000000 in SYST\_CVR.

#### 5.4. SysTick Interrupt

A SysTick interrupt is caught at SysTick\_Handler in startup\_TM4C129.s. It is relayed to \_timer\_update in timer.s

**This is the same as HW7-Q4.**

The timer\_update( ) function reads the value at address 0x20007B80, decrements the value by 1 (second), checks the value, branches to \_timer\_update\_done if the value hasn't reached 0, otherwise it needs to stop the timer and to invoke a user function whose address is maintained in 0x20007B84. To stop the timer, write “Bit 2 (CLK\_SRC) = 1, Bit 1 (INT\_EN) = 0, Bit 0 (ENABLE) = 0” to SYST\_CSR. (Don't forget to save back a decremented value into 0x20007B80.)

#### 6. Implementation Steps, Timeline, and Submissions

Since it is definitely hard to implement everything in assembly code at once, the final project will take the following two steps. To work on your project, distinguish the following three versions of driver.c program. They are all available from Canvas→files→final project.

**Table 9: driver programs**

Files you will work on	Tasks
driver.c	This is a complete C program that can be compiled with gcc and executable on Linux.
driver_cpg.c	This is a C program that should be used for testing your heap.c in step 1 toward your midpoint report. The difference from driver.c is: <ul style="list-style-type: none"> <li>- malloc( ) and free( ) are renamed _malloc( ) and _free( ), so that the compiler can use your own implementation of _malloc( ) and _free( ).</li> <li>- printf( ) are included to verify your implementation.</li> <li>- alarm( ) and signal( ) are commented out as you will implement in step 2.</li> </ul>
driver_keil.c	This is a C program that can be compiled with Keil C compiler and executable with your ARM/THUMB-2 assembly code. The difference from driver.c is: <ul style="list-style-type: none"> <li>- all stdlib functions bzero( ), strncpy( ), malloc( ), free( ), alarm( ), and signal( ) are renamed _bzero( ), _strncpy( ), _malloc( ), _free( ), _alarm( ), and _signal( ), so that the compiler can use your own implementation.</li> </ul>

#### 6.1. Step 1 toward the midpoint report (due on 2<sup>nd</sup> class date in week 8)

Step 1 intends to understand and develop the following two features:

- (1) **The reset sequence from the assembly language level all the way to main( ) in C which calls back down to stdlib.s in the assembly language level.**

startup\_tm4c129.s → main( ) in driver.c → stdlib.s

Your actual work on Keil uVersion is summarized below in table 10.

**Table 10: Keil uVersion work toward the midpoint report**

Files you will work on	Tasks
startup_tm4c129.s	Revise the Reset_Handler routine as follows: <ul style="list-style-type: none"> <li>- Set up and switch PSP (Process Stack Pointer)</li> <li>- Call main.</li> </ul>
driver_keil.c	Comment out the two while-loops, so that main( ) can complete with your

	stdlib.s partial implementation.  stdlib.s bzero and strcpy: Receive arguments from main( ), based on APCS, and complete the entire implementation within stdlib.s.  malloc, free, signal, and alarm: Receive arguments from main( ), based on APCS, but does nothing by simply returning back to main( ).
--	---

In Keil uVersion, start the debugger and take a memory snap of stringA and stringB after an execution.

## (2) A C-based implementation of the buddy memory allocation

Use driver\_cpg.c that calls \_malloc( ) and \_free( ) in heap.c. You can also find heap\_template.c in Canvas → files → final project folder. This is a template that hopefully makes it easy for you to implement the buddy memory allocation in C. Your C implementation must use a recursion. When you complete your C programs, rename this file “heap.c”. Table 11 summarizes your C implementation in step 1.

**Table 11: Linux C programming work toward the midpoint report**

Files you will work on	Tasks
driver_cpg.c	No need to change. But, if you like, you can include more printf or test statements.
heap.c	_malloc( ) and _free( ) in heap.c will internally call _kinit( ), _kalloc( ), and _kfree( ). As mentioned in section 4.3, _kalloc( ) and _kfree( ) will use recursive _ralloc( ) and _rfree( ) helper functions. In your step 2, _kinit( ), _kalloc( ), _ralloc( ), _kfree( ), and _rfree( ) will be implemented in ARM/THUMB-2 in heap.s.

Compile and run with:

gcc \*.c  
a.out

WeChat: cstutorcs

### Submission Items:

Please submit the following materials listed in table 12.

**Table 12: Step-1 Submission**

Materials	Remarks	Grade points (out of 25pts)
startup_tm4c129.s	From your Keil uVersion project	2pts
stdlib.s	From your Keil uVersion project	5pts
Two memory snapshots: stringA and stringB	From your Keil uVersion project	4pts
heap.c	From your Linux C program	10pts
a.out execution results	From your Linux C execution	4pts

### 6.2. Step 2 toward the final report (due on 2<sup>nd</sup> class date in week 11, i.e., final’s week)

After the midpoint report, the professor will disclose startup\_tm4c129.s, stdlib.s, and heap.c. You may refer to and use them to continue working on the rest of your final project. Step 2 intends to complete all assembly components in ARM/THUMB-2. Your work items in step 2 are summarized below in table 13.

**Table 13: Step-2 Work Items**

Files you will work on	Tasks
startup_tm4c129.s	<p>Correct the Reset_Handler routine if necessary, (based on the midpoint report feedback). Thereafter add subroutine calls such as:</p> <ul style="list-style-type: none"> <li>- _kinit: initialization in heap.s</li> <li>- _timer_init: initialization in timer.s</li> <li>- _systemcall_table_init: initialization in svc.s (table 4 in section 3.2.(2) )</li> </ul> <p>Implement the following two routines:</p> <ul style="list-style-type: none"> <li>- SVC_Handler: invoke _system_call_table_jump in svc.s</li> <li>- SysTick Handler: invoke timer_update in timer.s</li> </ul>
driver_keil.c	No more comment-out of the two while-loops. We entirely run driver_keil.c.
stdlib.s	<p>bzero and strcpy:</p> <p>Correct them if necessary, (based on the midpoint report feedback).</p> <p>malloc, free, signal, and alarm:</p> <p>Receive arguments from main( ), based on APCS and rely each call to SVC Handler.</p>
svc.s	Refer to section 3.2.(2). Based on the system call # in R7, jump to the corresponding function through the system call jump table in table 4.
heap.s	<p>Implement the following 5 routines, based on your C implementation in heap.c.</p> <ul style="list-style-type: none"> <li>_kinit: mcb initialization</li> <li>_kalloc: the entry point to invoke the _alloc recursive helper function</li> <li>_ralloc: a recursive helper function to allocate a space</li> <li>_kfree: the entry point to invoke the _rfree recursive helper function</li> <li>_rfree: a recursive helper function to free the space and merge the buddy space if possible</li> </ul>
timer.s	<p>Implement the following 4 routines, based on the specification in section 5.</p> <ul style="list-style-type: none"> <li>_timer_init: initialize SysTick.</li> <li>_timer_start: start SysTick.</li> <li>_timer_update: decrement seconds at 0x2000.7B80 and invokes *func at 0x2000.7B84.</li> <li>_signal_handler: register a user-provided *func at 0x2000.7B84.</li> </ul>

Test all your assembly language implementation with driver\_keil.c on Keil uVersion's debugger session. Take all memory snapshots of mcb addresses corresponding to mem1 – mem8 upon their allocation and deallocation as well as mem9's contents that should change from 1 to 2 and from 2 to 3.

#### Submission Items:

Please submit the following materials listed in table 14.

**Table 14: Step-2 Submission**

Materials	Remarks	Grade points (out of 75pts)
Your zipped Keil uVersion project (35pts)	<p>startup_tm4c129.s (5pts)</p> <p>Reset_Handler</p> <p>SVC_Handler</p> <p>SysTick_Handler</p>	<p>1pt</p> <p>2pts</p> <p>2pts</p>
	driver_keil.c	
	<p>stdlib.s (6pts)</p> <p>_bzero()</p> <p>_strncpy()</p>	<p>1pt</p> <p>1pt</p>



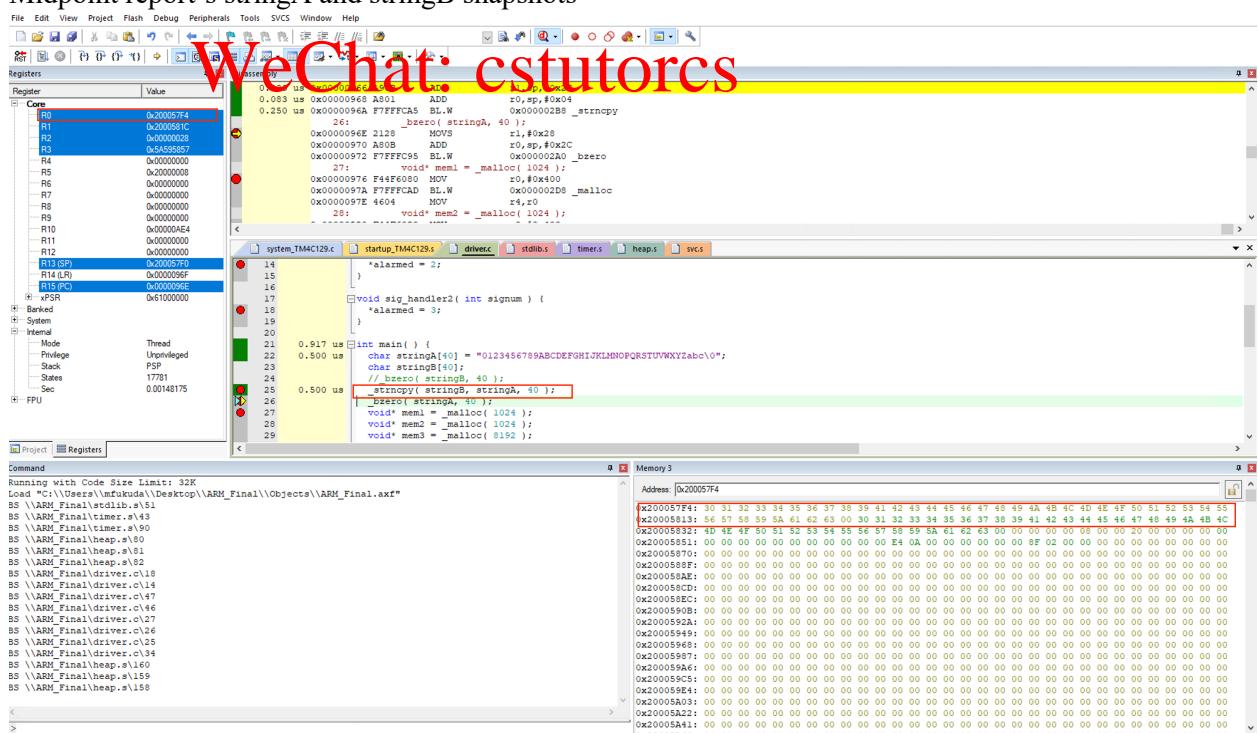
	<pre>         _free( mem9 );     }  void sig_handler1( int signum ) {     *alarmed = 2; }  void sig_handler2( int signum ) {     *alarmed = 3; } </pre>	1pt 1pt
Documentation (14pts)	A two-page summary of your implementation <ul style="list-style-type: none"> <li>- Narratives <ul style="list-style-type: none"> <li>o What you implemented.</li> <li>o What was missing.</li> </ul> </li> <li>- Any Diagrams (at least one)</li> </ul>	6pts 6pts 2pts
Extra credits (5pts)	If you implemented additional stdlib functions in ARM/THUMB-2, please write about them and highlight your narratives in our documentation	

## Assignment Project Exam Help

### 6.3. Execution Snapshots

To clarify what you need to turn in execution results, sample snapshots from the key answer are given below. Don't reuse them. Any reuse of these snapshots below will result in an academic misconduct.

(a) Midpoint report's stringA and stringB snapshots



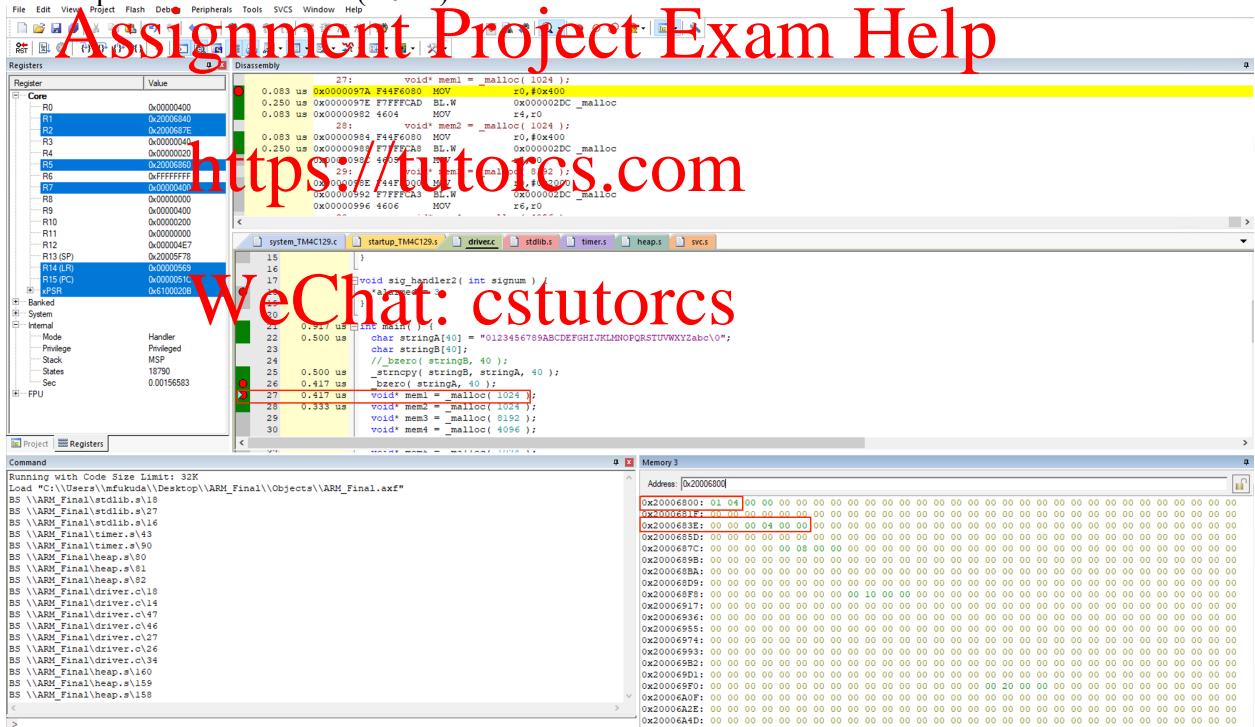
Memory 3

Address: 0x200057F4	30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55
0x20005813: 56 57 58 59 5A 61 62 63 00	
0x20005832: 00	
0x20005851: 00	
0x20005870: 00	
0x2000588F: 00	
0x200058AE: 00	
0x200058CD: 00	
0x200058EC: 00	
0x200058F1: 00	
0x200058A2: 00	
0x200058A3: 00	
0x200058A4: 00	
0x200058A5: 00	
0x200058A6: 00	
0x200058A7: 00	
0x200058A8: 00	
0x200058A9: 00	
0x200058A0: 00	
0x200058A1: 00	

(b) Midpoint report's a.out's outputs

```
[andromeda:C_Programs munehiro$ ./driver_cpg
0123456789ABCDEFHGIJKLMNOPQRSTUVWXYZabc
mem1 = 20001000
mem2 = 20001400
mem3 = 20003000
mem4 = 20002000
mem5 = 20001800
mem6 = 20001c00
mem7 = 20001a00
mem8 = 20001000
andromeda:C_Programs munehiro$ ]
```

(c) Final report's mem1 = malloc( 1024 )

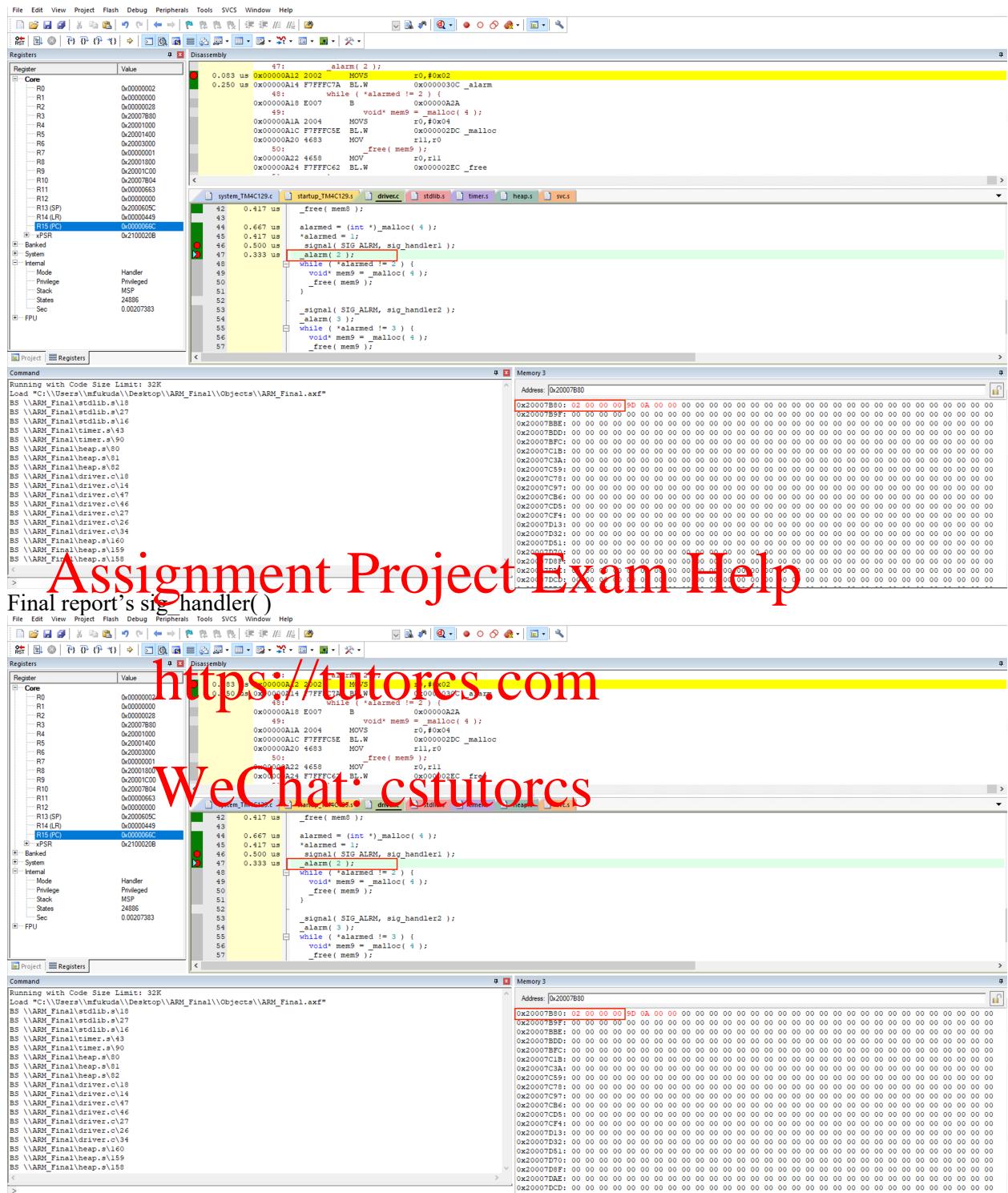


(d) Final report's free( mem6 )

The screenshot shows the Thump-2 debugger interface. The assembly window displays the following code snippet:

```

        34:    _free( mem6 );
        35:    mov r0, #0
        36:    ldr r1, [r0]
        37:    mov r2, r1
        38:    ldr r3, [r2]
        39:    mov r4, r3
        40:    ldr r5, [r4]
        41:    mov r6, r5
        42:    ldr r7, [r6]
        43:    mov r8, r7
        44:    ldr r9, [r8]
        45:    mov r10, r9
        46:    ldr r11, [r10]
        47:    mov r12, r11
        48:    ldr r13, [r12]
        49:    mov r14, r13
        50:    ldr r15, [r14]
        51:    mov r16, r15
        52:    ldr r17, [r16]
        53:    mov r18, r17
        54:    ldr r19, [r18]
        55:    mov r20, r19
        56:    ldr r21, [r20]
        57:    mov r22, r21
        58:    ldr r23, [r22]
        59:    mov r24, r23
        60:    ldr r25, [r24]
        61:    mov r26, r25
        62:    ldr r27, [r26]
        63:    mov r28, r27
        64:    ldr r29, [r28]
        65:    mov r30, r29
        66:    ldr r31, [r30]
        67:    mov r32, r31
        68:    ldr r33, [r32]
        69:    mov r34, r33
        70:    ldr r35, [r34]
        71:    mov r36, r35
        72:    ldr r37, [r36]
        73:    mov r38, r37
        74:    ldr r39, [r38]
        75:    mov r40, r39
        76:    ldr r41, [r40]
        77:    mov r42, r41
        78:    ldr r43, [r42]
        79:    mov r44, r43
        80:    ldr r45, [r44]
        81:    mov r46, r45
        82:    ldr r47, [r46]
        83:    mov r48, r47
        84:    ldr r49, [r48]
        85:    mov r50, r49
        86:    ldr r51, [r50]
        87:    mov r52, r51
        88:    ldr r53, [r52]
        89:    mov r54, r53
        90:    ldr r55, [r54]
        91:    mov r56, r55
        92:    ldr r57, [r56]
        93:    mov r58, r57
        94:    ldr r59, [r58]
        95:    mov r60, r59
        96:    ldr r61, [r60]
        97:    mov r62, r61
        98:    ldr r63, [r62]
        99:    mov r64, r63
        100:   ldr r65, [r64]
        101:   mov r66, r65
        102:   ldr r67, [r66]
        103:   mov r68, r67
        104:   ldr r69, [r68]
        105:   mov r70, r69
        106:   ldr r71, [r70]
        107:   mov r72, r71
        108:   ldr r73, [r72]
        109:   mov r74, r73
        110:   ldr r75, [r74]
        111:   mov r76, r75
        112:   ldr r77, [r76]
        113:   mov r78, r77
        114:   ldr r79, [r78]
        115:   mov r80, r79
        116:   ldr r81, [r80]
        117:   mov r82, r81
        118:   ldr r83, [r82]
        119:   mov r84, r83
        120:   ldr r85, [r84]
        121:   mov r86, r85
        122:   ldr r87, [r86]
        123:   mov r88, r87
        124:   ldr r89, [r88]
        125:   mov r90, r89
        126:   ldr r91, [r90]
        127:   mov r92, r91
        128:   ldr r93, [r92]
        129:   mov r94, r93
        130:   ldr r95, [r94]
        131:   mov r96, r95
        132:   ldr r97, [r96]
        133:   mov r98, r97
        134:   ldr r99, [r98]
        135:   mov r100, r99
        136:   ldr r101, [r100]
        137:   mov r102, r101
        138:   ldr r103, [r102]
        139:   mov r104, r103
        140:   ldr r105, [r104]
        141:   mov r106, r105
        142:   ldr r107, [r106]
        143:   mov r108, r107
        144:   ldr r109, [r108]
        145:   mov r110, r109
        146:   ldr r111, [r110]
        147:   mov r112, r111
        148:   ldr r113, [r112]
        149:   mov r114, r113
        150:   ldr r115, [r114]
        151:   mov r116, r115
        152:   ldr r117, [r116]
        153:   mov r118, r117
        154:   ldr r119, [r118]
        155:   mov r120, r119
        156:   ldr r121, [r120]
        157:   mov r122, r121
        158:   ldr r123, [r122]
        159:   mov r124, r123
        160:   ldr r125, [r124]
        161:   mov r126, r125
        162:   ldr r127, [r126]
        163:   mov r128, r127
        164:   ldr r129, [r128]
        165:   mov r130, r129
        166:   ldr r131, [r130]
        167:   mov r132, r131
        168:   ldr r133, [r132]
        169:   mov r134, r133
        170:   ldr r135, [r134]
        171:   mov r136, r135
        172:   ldr r137, [r136]
        173:   mov r138, r137
        174:   ldr r139, [r138]
        175:   mov r140, r139
        176:   ldr r141, [r140]
        177:   mov r142, r141
        178:   ldr r143, [r142]
        179:   mov r144, r143
        180:   ldr r145, [r144]
        181:   mov r146, r145
        182:   ldr r147, [r146]
        183:   mov r148, r147
        184:   ldr r149, [r148]
        185:   mov r150, r149
        186:   ldr r151, [r150]
        187:   mov r152, r151
        188:   ldr r153, [r152]
        189:   mov r154, r153
        190:   ldr r155, [r154]
        191:   mov r156, r155
        192:   ldr r157, [r156]
        193:   mov r158, r157
        194:   ldr r159, [r158]
        195:   mov r160, r159
        196:   ldr r161, [r160]
        197:   mov r162, r161
        198:   ldr r163, [r162]
        199:   mov r164, r163
        200:   ldr r165, [r164]
        201:   mov r166, r165
        202:   ldr r167, [r166]
        203:   mov r168, r167
        204:   ldr r169, [r168]
        205:   mov r170, r169
        206:   ldr r171, [r170]
        207:   mov r172, r171
        208:   ldr r173, [r172]
        209:   mov r174, r173
        210:   ldr r175, [r174]
        211:   mov r176, r175
        212:   ldr r177, [r176]
        213:   mov r178, r177
        214:   ldr r179, [r178]
        215:   mov r180, r179
        216:   ldr r181, [r180]
        217:   mov r182, r181
        218:   ldr r183, [r182]
        219:   mov r184, r183
        220:   ldr r185, [r184]
        221:   mov r186, r185
        222:   ldr r187, [r186]
        223:   mov r188, r187
        224:   ldr r189, [r188]
        225:   mov r190, r189
        226:   ldr r191, [r190]
        227:   mov r192, r191
        228:   ldr r193, [r192]
        229:   mov r194, r193
        230:   ldr r195, [r194]
        231:   mov r196, r195
        232:   ldr r197, [r196]
        233:   mov r198, r197
        234:   ldr r199, [r198]
        235:   mov r200, r199
        236:   ldr r201, [r200]
        237:   mov r202, r201
        238:   ldr r203, [r202]
        239:   mov r204, r203
        240:   ldr r205, [r204]
        241:   mov r206, r205
        242:   ldr r207, [r206]
        243:   mov r208, r207
        244:   ldr r209, [r208]
        245:   mov r210, r209
        246:   ldr r211, [r210]
        247:   mov r212, r211
        248:   ldr r213, [r212]
        249:   mov r214, r213
        250:   ldr r215, [r214]
        251:   mov r216, r215
        252:   ldr r217, [r216]
        253:   mov r218, r217
        254:   ldr r219, [r218]
        255:   mov r220, r219
        256:   ldr r221, [r220]
        257:   mov r222, r221
        258:   ldr r223, [r222]
        259:   mov r224, r223
        260:   ldr r225, [r224]
        261:   mov r226, r225
        262:   ldr r227, [r226]
        263:   mov r228, r227
        264:   ldr r229, [r228]
        265:   mov r230, r229
        266:   ldr r231, [r230]
        267:   mov r232, r231
        268:   ldr r233, [r232]
        269:   mov r234, r233
        270:   ldr r235, [r234]
        271:   mov r236, r235
        272:   ldr r237, [r236]
        273:   mov r238, r237
        274:   ldr r239, [r238]
        275:   mov r240, r239
        276:   ldr r241, [r240]
        277:   mov r242, r241
        278:   ldr r243, [r242]
        279:   mov r244, r243
        280:   ldr r245, [r244]
        281:   mov r246, r245
        282:   ldr r247, [r246]
        283:   mov r248, r247
        284:   ldr r249, [r248]
        285:   mov r250, r249
        286:   ldr r251, [r250]
        287:   mov r252, r251
        288:   ldr r253, [r252]
        289:   mov r254, r253
        290:   ldr r255, [r254]
        291:   mov r256, r255
        292:   ldr r257, [r256]
        293:   mov r258, r257
        294:   ldr r259, [r258]
        295:   mov r260, r259
        296:   ldr r261, [r260]
        297:   mov r262, r261
        298:   ldr r263, [r262]
        299:   mov r264, r263
        300:   ldr r265, [r264]
        301:   mov r266, r265
        302:   ldr r267, [r266]
        303:   mov r268, r267
        304:   ldr r269, [r268]
        305:   mov r270, r269
        306:   ldr r271, [r270]
        307:   mov r272, r271
        308:   ldr r273, [r272]
        309:   mov r274, r273
        310:   ldr r275, [r274]
        311:   mov r276, r275
        312:   ldr r277, [r276]
        313:   mov r278, r277
        314:   ldr r279, [r278]
        315:   mov r280, r279
        316:   ldr r281, [r280]
        317:   mov r282, r281
        318:   ldr r283, [r282]
        319:   mov r284, r283
        320:   ldr r285, [r284]
        321:   mov r286, r285
        322:   ldr r287, [r286]
        323:   mov r288, r287
        324:   ldr r289, [r288]
        325:   mov r290, r289
        326:   ldr r291, [r290]
        327:   mov r292, r291
        328:   ldr r293, [r292]
        329:   mov r294, r293
        330:   ldr r295, [r294]
        331:   mov r296, r295
        332:   ldr r297, [r296]
        333:   mov r298, r297
        334:   ldr r299, [r298]
        335:   mov r300, r299
        336:   ldr r301, [r300]
        337:   mov r302, r301
        338:   ldr r303, [r302]
        339:   mov r304, r303
        340:   ldr r305, [r304]
        341:   mov r306, r305
        342:   ldr r307, [r306]
        343:   mov r308, r307
        344:   ldr r309, [r308]
        345:   mov r310, r309
        346:   ldr r311, [r310]
        347:   mov r312, r311
        348:   ldr r313, [r312]
        349:   mov r314, r313
        350:   ldr r315, [r314]
        351:   mov r316, r315
        352:   ldr r317, [r316]
        353:   mov r318, r317
        354:   ldr r319, [r318]
        355:   mov r320, r319
        356:   ldr r321, [r320]
        357:   mov r322, r321
        358:   ldr r323, [r322]
        359:   mov r324, r323
        360:   ldr r325, [r324]
        361:   mov r326, r325
        362:   ldr r327, [r326]
        363:   mov r328, r327
        364:   ldr r329, [r328]
        365:   mov r330, r329
        366:   ldr r331, [r330]
        367:   mov r332, r331
        368:   ldr r333, [r332]
        369:   mov r334, r333
        370:   ldr r335, [r334]
        371:   mov r336, r335
        372:   ldr r337, [r336]
        373:   mov r338, r337
        374:   ldr r339, [r338]
        375:   mov r340, r339
        376:   ldr r341, [r340]
        377:   mov r342, r341
        378:   ldr r343, [r342]
        379:   mov r344, r343
        380:   ldr r345, [r344]
        381:   mov r346, r345
        382:   ldr r347, [r346]
        383:   mov r348, r347
        384:   ldr r349, [r348]
        385:   mov r350, r349
        386:   ldr r351, [r350]
        387:   mov r352, r351
        388:   ldr r353, [r352]
        389:   mov r354, r353
        390:   ldr r355, [r354]
        391:   mov r356, r355
        392:   ldr r357, [r356]
        393:   mov r358, r357
        394:   ldr r359, [r358]
        395:   mov r360, r359
        396:   ldr r361, [r360]
        397:   mov r362, r361
        398:   ldr r363, [r362]
        399:   mov r364, r363
        400:   ldr r365, [r364]
        401:   mov r366, r365
        402:   ldr r367, [r366]
        403:   mov r368, r367
        404:   ldr r369, [r368]
        405:   mov r370, r369
        406:   ldr r371, [r370]
        407:   mov r372, r371
        408:   ldr r373, [r372]
        409:   mov r374, r373
        410:   ldr r375, [r374]
        411:   mov r376, r375
        412:   ldr r377, [r376]
        413:   mov r378, r377
        414:   ldr r379, [r378]
        415:   mov r380, r379
        416:   ldr r381, [r380]
        417:   mov r382, r381
        418:   ldr r383, [r382]
        419:   mov r384, r383
        420:   ldr r385, [r384]
        421:   mov r386, r385
        422:   ldr r387, [r386]
        423:   mov r388, r387
        424:   ldr r389, [r388]
        425:   mov r390, r389
        426:   ldr r391, [r390]
        427:   mov r392, r391
        428:   ldr r393, [r392]
        429:   mov r394, r393
        430:   ldr r395, [r394]
        431:   mov r396, r395
        432:   ldr r397, [r396]
        433:   mov r398, r397
        434:   ldr r399, [r398]
        435:   mov r400, r399
        436:   ldr r401, [r400]
        437:   mov r402, r401
        438:   ldr r403, [r402]
        439:   mov r404, r403
        440:   ldr r405, [r404]
        441:   mov r406, r405
        442:   ldr r407, [r406]
        443:   mov r408, r407
        444:   ldr r409, [r408]
        445:   mov r410, r409
        446:   ldr r411, [r410]
        447:   mov r412, r411
        448:   ldr r413, [r412]
        449:   mov r414, r413
        450:   ldr r415, [r414]
        451:   mov r416, r415
        452:   ldr r417, [r416]
        453:   mov r418, r417
        454:   ldr r419, [r418]
        455:   mov r420, r419
        456:   ldr r421, [r420]
        457:   mov r422, r421
        458:   ldr r423, [r422]
        459:   mov r424, r423
        460:   ldr r425, [r424]
        461:   mov r426, r425
        462:   ldr r427, [r426]
        463:   mov r428, r427
        464:   ldr r429, [r428]
        465:   mov r430, r429
        466:   ldr r431, [r430]
        467:   mov r432, r431
        468:   ldr r433, [r432]
        469:   mov r434, r433
        470:   ldr r435, [r434]
        471:   mov r436, r435
        472:   ldr r437, [r436]
        473:   mov r438, r437
        474:   ldr r439, [r438]
        475:   mov r440, r439
        476:   ldr r441, [r440]
        477:   mov r442, r441
        478:   ldr r443, [r442]
        479:   mov r444, r443
        480:   ldr r445, [r444]
        481:   mov r446, r445
        482:   ldr r447, [r446]
        483:   mov r448, r447
        484:   ldr r449, [r448]
        485:   mov r450, r449
        486:   ldr r451, [r450]
        487:   mov r452, r451
        488:   ldr r453, [r452]
        489:   mov r454, r453
        490:   ldr r455, [r454]
        491:   mov r456, r455
        492:   ldr r457, [r456]
        493:   mov r458, r457
        494:   ldr r459, [r458]
        495:   mov r460, r459
        496:   ldr r461, [r460]
        497:   mov r462, r461
        498:   ldr r463, [r462]
        499:   mov r464, r463
        500:   ldr r465, [r464]
        501:   mov r466, r465
        502:   ldr r467, [r466]
        503:   mov r468, r467
        504:   ldr r469, [r468]
        505:   mov r470, r469
        506:   ldr r471, [r470]
        507:   mov r472, r471
        508:   ldr r473, [r472]
        509:   mov r474, r473
        510:   ldr r475, [r474]
        511:   mov r476, r475
        512:   ldr r477, [r476]
        513:   mov r478, r477
        514:   ldr r479, [r478]
        515:   mov r480, r479
        516:   ldr r481, [r480]
        517:   mov r482, r481
        518:   ldr r483, [r482]
        519:   mov r484, r483
        520:   ldr r485, [r484]
        521:   mov r486, r485
        522:   ldr r487, [r486]
        523:   mov r488, r487
        524:   ldr r489, [r488]
        525:   mov r490, r489
        526:   ldr r491, [r490]
        527:   mov r492, r491
        528:   ldr r493, [r492]
        529:   mov r494, r493
        530:   ldr r495, [r494]
        531:   mov r496, r495
        532:   ldr r497, [r496]
        533:   mov r498, r497
        534:   ldr r499, [r498]
        535:   mov r500, r499
        536:   ldr r501, [r500]
        537:   mov r502, r501
        538:   ldr r503, [r502]
        539:   mov r504, r503
        540:   ldr r505, [r504]
        541:   mov r506, r505
        542:   ldr r507, [r506]
        543:   mov r508, r507
        544:   ldr r509, [r508]
        545:   mov r510, r509
        546:   ldr r511, [r510]
        547:   mov r512, r511
        548:   ldr r513, [r512]
        549:   mov r514, r513
        550:   ldr r515, [r514]
        551:   mov r516, r515
        552:   ldr r517, [r516]
        553:   mov r518, r517
        554:   ldr r519, [r518]
        555:   mov r520, r519
        556:   ldr r521, [r520]
        557:   mov r522, r521
        558:   ldr r523, [r522]
        559:   mov r524, r523
        560:   ldr r525, [r524]
        561:   mov r526, r525
        562:   ldr r527, [r526]
        563:   mov r528, r527
        564:   ldr r529, [r528]
        565:   mov r530, r529
        566:   ldr r531, [r530]
        567:   mov r532, r531
        568:   ldr r533, [r532]
        569:   mov r534, r533
        570:   ldr r535, [r534]
        571:   mov r536, r535
        572:   ldr r537, [r536]
        573:   mov r538, r537
        574:   ldr r539, [r538]
        575:   mov r540, r539
        576:   ldr r541, [r540]
        577:   mov r542, r541
        578:   ldr r543, [r542]
        579:   mov r544, r543
        580:   ldr r545, [r544]
        581:   mov r546, r545
        582:   ldr r547, [r546]
        583:   mov r548, r547
        584:   ldr r549, [r548]
        585:   mov r550, r549
        586:   ldr r551, [r550]
        587:   mov r552, r551
        588:   ldr r553, [r552]
        589:   mov r554, r553
        590:   ldr r555, [r554]
        591:   mov r556, r555
        592:   ldr r557, [r556]
        593:   mov r558, r557
        594:   ldr r559, [r558]
        595:   mov r560, r559
        596:   ldr r561, [r560]
        597:   mov r562, r561
        598:   ldr r563, [r562]
        599:   mov r564, r563
        600:   ldr r565, [r564]
        601:   mov r566, r565
        602:   ldr r567, [r566]
        603:   mov r568, r567
        604:   ldr r569, [r568]
        605:   mov r570, r569
        606:   ldr r571, [r570]
        607:   mov r572, r571
        608:   ldr r573, [r572]
        609:   mov r574, r573
        610:   ldr r575, [r574]
        611:   mov r576, r575
        612:   ldr r577, [r576]
        613:   mov r578, r577
        614:   ldr r579, [r578]
        615:   mov r580, r579
        616:   ldr r581, [r580]
        617:   mov r582, r581
        618:   ldr r583, [r582]
        619:   mov r584, r583
        620:   ldr r585, [r584]
        621:   mov r586, r585
        622:   ldr r587, [r586]
        623:   mov r588, r587
        624:   ldr r589, [r588]
        625:   mov r590, r589
        626:   ldr r591, [r590]
        627:   mov r592, r591
        628:   ldr r593, [r592]
        629:   mov r594, r593
        630:   ldr r595, [r594]
        631:   mov r596, r595
        632:   ldr r597, [r596]
        633:   mov r598, r597
        634:   ldr r599, [r598]
        635:   mov r600, r599
        636:   ldr
```



## 7. Final notes

- (1) Follow the final project specification.
  - a. Use the memory spaces exactly specified in this document.
  - b. Use the function and routine names specified in this document.
  - c. Attach the execution results as specified in this document (see tables 12 and 14).
- (2) Check Canvas → files → final project folder for additional materials.
- (3) Start your implementation early and keep up your plan.