

Name: _____

Problem 1: (15 points) (10 minutes) You, Detective Jack Clouseau, are guarding the coveted Pink Panther diamond, which resides in a safe with a special combination lock. The lock has four dials – each dial consists of numbers 0-9 and letters a, b.

- a) **(5 points)** How many unique combinations are there in your special combination lock?

Total number of alphabets on each dial = 12 (0-9, a, b)
No. of dials = 4
 $\therefore 12^4$ unique combinations

- b) **(5 points)** Meanwhile, the Phantom is trying to steal the diamond. If it takes him a second to check each combination, and you check on the safe every 15 minutes, what are your chances of catching him? Justify your answer.

Assuming the initial combination on the lock was completely randomized, it is reasonable to use the ~~worst case~~ time it takes the Phantom to find the right combination (in the worst case which is 12^4 seconds $\approx$ 345 min

If you check on the lock every 15 min, looks like you can catch him with high likelihood!

Name: _____

- c) (5 points) You decide that you want to make a new lock with hundred times the number of possible combinations as the old one. If each dial consists of the same numbers and letters as before, what is the minimum number of dials you need on the new lock?

We need to find the lowest power of 12 that
would just fit 100×12^4

So we need 2 more dials.

Name: _____

Problem 2: (40 points) (30 minutes) C Medley

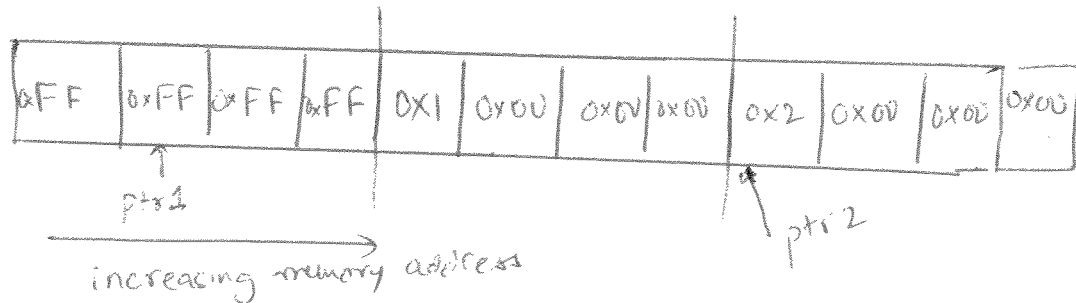
(a) (20 points) Fun with Pointers

For the C code below make the following assumptions

- i. The base address of array is 0x10000000
- ii. The byte ordering is Little Endian
- iii. int is 4 bytes

```
int array[3] = {-1, 1, 2};  
int main() {  
    unsigned char *ptr1 = (((unsigned char *) array) + 1);  
    unsigned char val;  
    int *ptr2 = array + 2;  
    val = *ptr1;  
    return 0;  
}
```

- i) **(10 points)** Show the byte level representation of array in memory. You have to mark the increasing direction of memory addresses.



- ii) **(5 points)** What is the value in *ptr2 before the return statement?

2

- iii) **(5 points)** What is the value of the variable val before the return statement?

0xFF

Name: _____

(b) (20 points) Buggy Buggy Debuggy

Consider the following C code.

```
int main()
{
    short disappear[]={1,2,3};
    short index=2;

    do
    {
        disappear[index]=0;

    }while(index-- >=0);
}
```

- i) **(10 points)** Show the entire contents of the array disappear at the end of each iteration of the do-while loop, for three iterations.

i) (10 points) Show the entire contents of the array disappear at the end of each iteration of the do-while loop, for three iterations.

Iteration 1	<table><tr><td>1</td><td>2</td><td>0</td></tr></table>	1	2	0	index = 2	Showing 'disappear' when program execution reaches 'while' statement
1	2	0				
Iteration 2	<table><tr><td>1</td><td>0</td><td>0</td></tr></table>	1	0	0	index = 1	
1	0	0				
Iteration 3	<table><tr><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	index = 0	
0	0	0				
Iteration 4 (if no seg fault)	<table><tr><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	index = -1	
0	0	0				

- ii) **(5 points)** There is a bug in the above code. Identify it and explain its possible effects on the compilation and execution of the program

Out of bound array access because of the condition
 $\text{while}(\text{index}-- \geq 0);$

This causes run-time errors: either a seg fault or data corruption which may lead to unpredictable program behavior.

Name: _____

- iii) (5 points) Change the code to correct for the above bug. What are the final values of index and the contents of the array disappear, right before main returns?

Change last line to `while (index --> 0)`

After this change :

index = -1 & disappear =

0	0	0
---	---	---

before main returns

Name: _____

Problem 3: (25 points) (20 minutes) Fill in the code

(a) (15 points) Data Structures

Consider the following definition of a node in a doubly linked list.

```
typedef struct Node ListNode;  
  
struct Node {  
    int data;           // this is the data in this node element  
    ListNode* next;     // this is a pointer to the next element in the list  
    ListNode* prev;     // this is a pointer to the previous element in the list  
};
```

The following C function should completely reverse the order of the elements in the list. A part of the function is implemented using recursion. Fill in the missing implementation. The function is initially called with a pointer to the first `ListNode` object of the list. It should ultimately return a pointer to the new first element of the list (which was previously the last element)

```
ListNode * reverse(ListNode *aNode)  
{  
    /* Your code goes here */  
    if (aNode == NULL)  
        return NULL;  
  
    ListNode *tmp = aNode->prev;  
    aNode->prev = aNode->next;  
    aNode->next = tmp;  
  
    if(aNode->prev == NULL)    //stop if end of list  
        { return aNode; }    // has been reached  
    else  
        { return reverse(aNode->prev); } //otherwise continue  
                                           // recursively  
}
```

Please see supplement on last page for an iterative implementation in C & Assembly

Name: _____

(b) (10 points) **C Strings:** The following function should copy n characters from string source to string dest starting with the first element of dest . Fill in the blanks to complete the code. There should only be one expression in each while loop, meaning you must only fill in the right hand side of the assignment statement in the first while loop, and the left hand side of the assignment statement in the second while loop. You should not add any additional statements to the code.

```
/* strncpy: copy n characters from source to dest */
void strncpy(char * dest, char * source, int n)
{
    while(*source && n-- > 0)
    {
        *dest++ = *source++;
    }

    while(n-- > 0)
    {
        *dest++ = 0;
    }
}
```

Name: _____

Problem 4: (25 points) Bit Manipulation (30 minutes) Write ARM code for the function `isolateRangeOfBits`. This function isolates the bits between lower and upper and returns only those bits. You must follow the ARM Procedure Call Standard. You must comment your code extensively. Code that is not adequately commented will be penalized.

```
isolateRangeOfBits(0x12345678, 0, 8) returns 0x78
isolateRangeOfBits(0x12345678, 8, 12) returns 0x6
isolateRangeOfBits(0x12345678, 30, 32) returns 0x0
```

```
/* Precondition: lower <= upper */
int isolateRangeOfBits(int number, unsigned int lower,
unsigned int upper);
```

```
isolateRangeOfBits:  push {r4, lr} ; Save r4, lr
                    LSR  r0, r0, r1 ; Mov bits originally in [r1:r2]
                        into r0 = (r2-r1-1)

                    SUB  r3, r2, r1
                    RSB  r4, r3, #32 ; r4 = 32 - (r2-r1)
                        ; No. of bits to the left
                        ; of pattern of interest

                    LSL  r0, r0, r4 ;
                    LSR  r0, r0, r4 ; Clear bits to the left of
                        the (r2-r1-1)th bit

                    pop {r4, lr} ; Restore r4, lr
                    BX  lr ; Restore Return
```

Name: _____

Name: _____

Problem 5: (25 points) Compilation (30 minutes)

The following function performs a finite impulse response (FIR) filter using input from array z and coefficients found in integer array h .

```
int compute_FIR( int* h, int* z, int ntaps, int input)
{
    int ii;
    int accum;

    /* store input at the beginning of the delay line */
    z[0] = input;

    /* calc FIR and shift data */
    accum = h[ntaps - 1] * z[ntaps - 1];
    for (ii = ntaps - 2; ii >= 0; ii--) {
        accum += h[ii] * z[ii];
        z[ii + 1] = z[ii];
    }
    return accum;
}
```

Write the above function in assembly using ARM instructions. Specify the mapping between variables and registers in your code. You must follow the ARM Procedure Call Standard.

Mapping:

$h: r0$	$z: r1$
$ntaps: r2$	$input: r3$
$z[i]: r4$	$h[i]: r5$
$z[i+1]: r6$	$accum: r7$
$result: r0$	

compute_FIR:

```
push {r4-r11, lr}
STR r3, [r1]; z[0] = input
SUB r4, r2, #1; ntaps-1 → r4
LDR r5, [r0, r4, LSL #2]; r5 = h[ntaps-1]
LDR r6, [r1, r4, LSL #2]; r6 = z[ntaps-1]
MUL r7, r5, r6; accum = h[ntaps-1] * z[ntaps-1]
```

Name: _____

SUB r4, r4, #1; r4 = naps - 2

for-loop CMP r4, #0;

BLT return

LDR r5, [r0, r4, LSL #2]; r5 = h[i]

LDR r6, [r1, r4, LSL #2]; r6 = z[i]

MLA r7, r5, r6, r7; accum += h[i] * z[i]

ADD r4, r4, #1; i = i + 1;

STR r6, [r1, r4, LSL #2]; z[i+1] = z[i]

SUB r4, r4, #2; i = i - 2

B for-loop

return: MOV r0, r7

POP {r4-r11, lr}

Bx lr

Name: _____

Problem 6: (40 points) NO DISASSEMBLE! REASSEMBLE!

What kind of pants does Mario wear? DENIM, denim, denim

Consider the following code:

```
main:      MOV r0, #16
           SUB r0, r0, #1
           MOV r1, #5
           ADD r1, r1, r1, LSL #2
           BL mario
           BX lr

mario:     CMP r0, r1
           BEQ yoshi
           BLT luigi
           SUB r0, r0, r1
           B mario

luigi:     SUB r1, r1, r0
           B mario

yoshi:     BX lr
```

- a) (5 points) How many functions does the above code contain? What is the return value of the mario function?

Two: main & mario

Return value = 5 (in r0)

Name: _____

- b) (10 points) Assume the processor that runs the above code implements a five-stage pipeline. Are there any data hazards in the above code, if no special mechanisms were in place to deal with data hazards? If yes, identify them and justify your answer.

Yes, there are two data hazards.

1st instance: `mov r0, #16 ;` `sub r0, r0, #1 ;` `r0` updated in the 5th cycle but is being consumed in the third cycle

2nd instance: `mov r1, #5 ;` `add r1, r1, r1, lsl #2 ;` Similar situation with `r1`

- c) (5 points) List two mechanisms to alleviate data hazards. If the processor implemented each of these mechanisms, how many cycles does the execution of the main function take until (but excluding) the call to `mario`?

To alleviate the data hazard we could: (5+3+1+3)

1. Stall: execution of 1st 4 lines of main will take 12 cycles

2. Forward result of arithmetic operations to next ~~stage~~ ^{instruction} right after the execute stage.

Execution time = 8 cycles (5+1+1+1)

3. Reordering: `mov r0, #16`
`mov r1, #5`
`sub r0, r0, #1`
`add r1, r1, r1, lsl #2`

Execution time: 5+1+2+1 = 9 cycles

Name: _____

- d) (5 points) How many bytes of code does the `mario` function use? Assume the normal ARM instruction set as we discussed in class (i.e., no Thumb instructions).

Size in bytes of each instruction
↑
 $8 \times 4 = 32$ bytes
↓
No. of instructions

- e) (10 points) Rewrite the function using the skeleton code below. You only need to add three instructions. How many bytes of code does your new implementation of `mario` use? $5 \times 4 = 20$ bytes

```
mario      CMP r0,r1  
  
           SUBGT  r0,r0,r1  
  
           SUBLT  r1,r1,r0  
  
           BNE   mario  
  
yoshi      BX lr
```

- f) (5 points) Translate the `mario` function into C. Your function header should list the types of any arguments and return values. Also, your code should be as concise as possible, without any `gotos`.

```
int mario (int x, int y)      x: r0  
                                y: r1  
{  
    while (x != y)  
    {  
        if (x < y)  
            y = y - x;  
        else  
            x = x - y;  
    }  
    return x  
}
```

Name: _____

Problem 3. Iterative C implementation

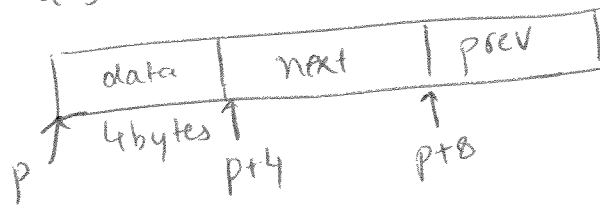
```
ListNode * reverse_iter ( ListNode * aNode )
```

```
{  
    ListNode * tmp;  
    while ( aNode != NULL )  
    {  
        tmp = aNode → prev;  
        aNode → prev = aNode → next;  
        aNode → next = tmp;  
        if ( aNode → prev == NULL )  
            return aNode;
```

```
        aNode = aNode → prev;  
    }
```

```
}
```

For the ARM implementation we need to know the representation of ~~data~~ an instance of struct in memory, size of pointer & size of int. Assuming a 32-bit machine, an instance of ListNode would be:



See next page for the ARM code.

reverse iter- ARM:

aNode: r0

```
loop: CMP r0, #0
      BEQ fin
      LDR r1, [r0, #8]; r1 = aNode → prev
      LDR r2, [r0, #4]; r2 = aNode → next
      STR r1, [r0, #4]; r1 aNode → next = r1
      STR r2, [r0, #8]; aNode → prev = r2
      CMP r2, #0
      BEQ fin
      MOV r0, r2; set r0 = aNode → prev;
      B loop
```

fin:

Bx lr