(d) Consider the following snippet of C code

```
int i, a[10];
for (i = 0, a[0] = 0; i < 9; i++)
   a[i+1] = a[i]+1;
```

You are to translate this code into ARM assembly language. Include comments that state clealy how you have mapped the C variables to registers. Do not provide a complete program; write only enough assembly code to be equivalent to the C code.

Put your answer in the space below

```
          MOV     R0, #0          // i = 0
          MOV     R1, #A          // R1 -> A[0]
          STR     R0, [R1], #4    // A[i] = 0, point to A[i+1]
FOR:      CMP     R0, #9
          BGE     ENDFOR
          ADD     R0, #1
          STR     R0, [R1], #4    // A[i+1] = i
          B       FOR

ENDFOR:   B       ENDFOR

A:       .space   40
```

[12 marks] 3. This question involves writing both assembly-language and C code.

In the space below you are to write a C-language program for the ARM processor. The program should perform the following task: in an infinite loop read from the SW port and show on HEX1-0 the number of SW switches that are set to 1. Your result has to be displayed in *decimal*. Hence if the SW switches were set to 1001110011, then you would display `6`. If no switches are in the upward position you would display `0`, and if all ten switches are set to 1 you would display `10`. Note that an array of 7-segment patterns is provided for convenience. (The SW port address is `0xFF200040`. A diagram of the 7-segment display port is shown at the end of the exam.)

```
char seg7[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};

int main(void) {
   volatile int *SW_ptr = 0xFF200040;
   volatile int *HEX3_1_ptr = 0xFF200020;
   int switches, count;

   while (1) {
      switches = *SW_ptr;
      count = 0;
      while (switches) {
         if (switches & 0b1)
            ++count;
         switches = switches >> 1;
      }
      if (count < 10)
         *HEX3_1_ptr = seg7[count];
      else
         *HEX3_1_ptr = (seg7[1] << 8) | seg7[0];
   }
}
```

On the next page, rewrite your code from part (a) of this question, but using ARM assembly language. Your program should produce exactly the same output as the C-language version. Note that an array of 7-segment patterns is provided for convenience.

```
        .text
        .global _start
_start:
        LDR     R0, =0xFF200000
        LDR     R7, =SEG7

LOOP:   LDR     R1, [R0, #0x40]     // read SW
        MOV     R2, #0              // reset count
WHILE:  CMP     R1, #0              // any SW set?
        BEQ     DISPLAY

        ANDS    R3, R1, #0b1        // check lsb
        ADDNE   R2, #1              // ++count
        LSR     R1, #1              // shift SW and continue
        B       WHILE

DISPLAY: CMP    R2, #10             // count == 10?
        BEQ     TEN
        LDRB    R2, [R7, R2]        // get 7-seg pattern for 0-9
        STR     R2, [R0, #0x20]     // write to HEX3-0
        B       LOOP
TEN:    LDRB    R2, [R7, #0]        // get 7-seg pattern for 1
        LDRB    R1, [R7, #1]        // get 7-seg pattern for 0
        ORR     R2, R1, LSL #8      // form final 7-seg pattern
        STR     R2, [R0, #0x20]     // write to HEX3-0
        B       LOOP

SEG7:   .byte   0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67
        .end
```

[15 marks] 4. The program shown below has to enable interrupts every 0.25 seconds for the FPGA interval timer (interrupt ID = 72), which has a 100 MHz clock. A diagram of this timer's registers is given at the end of the exam. The main program displays a counter on the red LEDs, and the interrupt service routine for the timer increments this counter. You are to fill in the missing code so that the program will function properly. Assume that the exception vector table has been set up properly (code not shown).

```
        .text
        .global  _start
_start:
        /* Set up stack pointers, etc. */

        MOV     R1, #0b11010010   // ints disabled, mode = IRQ
        MSR     CPSR_c, R1        // change to IRQ mode
        MOV     SP, #0x40000

        MOV     R1, #0b11010011   // ints diabled, SVC mode
        MSR     CPSR, R1          // change to supervisor mode
        MOV     SP, #0x20000



        // Enable timer interrupts in the GIC
        BL      CONFIG_GIC        // this code is not shown (ignore)
        BL      CONFIG_TIMER      // configure the FPGA timer



        /* enable IRQ interrupts in the processor */
        MOV     R1, #0b01010011 // IRQ unmasked, mode = SVC
        MSR     CPSR_c, R1



        LDR     R5, =0xFF200000   // LEDR base address
LOOP:   LDR     R3, COUNT         // global variable
        STR     R3, [R5]          // light up the red lights
        B       LOOP

COUNT: .word    0x0               // used by timer
```

```
CONFIG_TIMER:   // Configure the interval timer
       LDR      R0, =0xFF202000



       LDR      R1, =25000000    // 1/(100 MHz) x 25x10^6 = 0.25 sec
       STR      R1, [R0, #0x8]   // store low halfword of start value
       LSR      R1, R1, #16
       STR      R1, [R0, #0xC]   // high halfword of start value

       // start the interval timer, enable its interrupts
       MOV      R1, #0x7         // START = 1, CONT = 1, ITO = 1
       STR      R1, [R0, #0x4]



       MOV      PC, LR

SERVICE_IRQ:    // IRQ Exception service routine
        PUSH    {R0-R7, LR}
        LDR     R4, =0xFFFEC100
        LDR     R5, [R4, #0x0C]  // read the interrupt ID


        CMP     R5, #72          // check for FPGA timer interrupt
BAD:    BNE     BAD              // if not recognized, stop here

        BL      TIMER_ISR
        STR     R5, [R4, #0x10]  // clear interrupt from GIC

        POP     {R0-R7, LR}
        SUBS    PC, LR, #4
```

```
TIMER_ISR:      // Interval timer interrupt service routine
        LDR     R0, =0xFF202000 // interval timer base address


        MOV     R1, #0
        STR     R1, [R0]        // clear the interrupt

        LDR     R2, =COUNT      // pointer to the COUNT variable
        LDR     R3, [R2]        // load the current value of COUNT
        ADD     R3, #1          // increment by 1
        STR     R3, [R2]        // save to COUNT variable

        MOV     PC, LR
```

[10 marks] 5. Trace an ARM Program:

Consider the ARM code shown below. Note that the address that each instruction would have in the memory is shown to the left of the code.

```
                      .text
                      .global _start
            _start:
00000000              LDR      SP, =0x20000
00000004              LDR      R4, =0xFF200000
00000008              LDR      R0, =LIST

0000000C              BL       SEEIT
00000010              STR      R0, [R4]
00000014    GOTIT:    B        GOTIT
00000018    LIST:     .word    0xA3, 88, 105, 0x89, 221, 0x100
00000030              .word    0

00000034    SEEIT:    LDR      R1, [R0]
00000038              ADD      R0, #4
0000003C              LDR      R3, [R0]
00000040              CMP      R3, #0
00000044              BNE      WANTIT
00000048              MOV      R0, #0
0000004C              B        LIKEIT
00000050    WANTIT:   PUSH     {R1, LR}
00000054              BL       SEEIT
00000058              POP      {R1, LR}

0000005C    LIKEIT:   CMP      R1, R0
00000060              MOVGT    R0, R1
00000064              MOV      PC, LR

                     .end
```

(a) State in one or two sentences what this code "does". That is, given a list of data what does the program produce as an output?

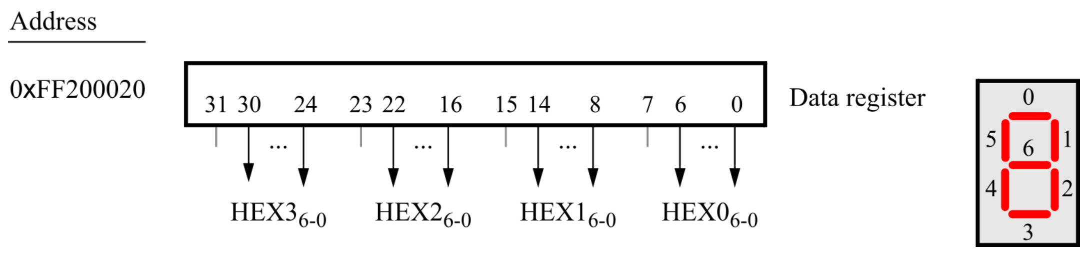
Answer: It finds the largest number in LIST, and displays it on LEDR

(b) If this program is executed on the ARM processor, what would be the values of the ARM registers shown below the **first** time the code reaches, but has not yet executed, the instruction at address 0x58. Also, show in the space below the contents of the stack in memory at this point in time (fill in the memory addresses on the left, and show the data stored in each location). For memory values that are not known, if any, write N/A in the corresponding box.

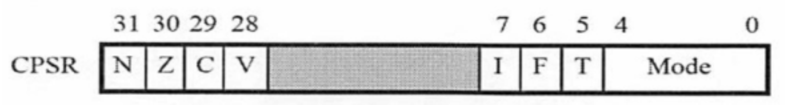
R0 0x100	R1 0x100	R3 0
R13 0x0001FFD8	R14 0x00000058	R15 0x00000058

Memory Address	Content
1FFD8	0xDD
1FFDC	0x58
1FFE0	0x89
1FFE4	0x58
1FFE8	0x69
1FFEC	0x58
1FFF0	0x58
1FFF4	0x58
1FFF8	0xA3
1FFFC	0x10
20000	N/A

Seven-segment Display Port



Processor Modes



User Mode:	10000
FIQ Mode:	10001
IRQ Mode:	10010
Supervisor Mode:	10011
Abort Mode:	10111
Undefined Mode:	11011

FPGA Interval Timer Port

Address	31 . . . 17 16	15 . . . 4	3	2	1	0	
0xFF202000	Not present (interval timer has 16-bit registers)	Unused			RUN	TO	Status register
0xFF202004		Unused	STOP	START	CONT	ITO	Control register
0xFF202008		Counter start value (low)					
0xFF20200C		Counter start value (high)					
0xFF202010		Counter snapshot (low)					
0xFF202014		Counter snapshot (high)					

GIC Interface

Address	31 . . . 10	9 8	7 . . . 1	0	Register name
0xFFFEC100	Unused			E	ICCICR
0xFFFEC104	Unused		Priority		ICCPMR
0xFFFEC10C	Unused	Interrupt ID			ICCIAR
0xFFFEC110	Unused	Interrupt ID			ICCEOIR