First:_____   Last:_____

Scoring The correct output values are shown in the figure on the right. Your grade will be based both on the numerical results returned by your program and on your programming style. In particular, write code that is easy to understand, easy to debug, easy to change. Please employ good labels, pretty structure, and good comments.

Performance Score=		TA:
Run by TA at the checkout		

```
UART #1                              ×
Exam2_Quad
Test of Pow4
Your=1   Yes, Score = 2
Your=4   Yes, Score = 4
Your=16  Yes, Score = 6
Your=64  Yes, Score = 8
Your=2   Yes, Score = 11
Your=8   Yes, Score = 14
Your=32  Yes, Score = 17
Your=128  Yes, Score = 20
Your=3   Yes, Score = 23
Your=12  Yes, Score = 26
Your=48  Yes, Score = 29
Your=192  Yes, Score = 32
Your=0   Yes, Score = 35
Your=0   Yes, Score = 38
Your=0   Yes, Score = 41
Your=0   Yes, Score = 44
Test of QuadDec
Your = 27  Yes, Score = 53
Your = 3   Yes, Score = 62
Your = 85  Yes, Score = 71
Your = 255  Yes, Score = 80
Your = 240  Yes, Score = 90
Your = 0   Yes, Score = 100
End of Exam2_Quad
```

I promise to follow these rules
        This is a closed book exam. You must develop the software solution using the **Keil uVision** simulator. You have 35 minutes, so allocate your time accordingly. You are allowed to bring only some pencils (no books, laptops, cell phones, hats, disks, CDs, or notes). You will have to leave other materials up front. Each person works alone (no groups). You have full access to **Keil uVision**, with the **Keil uVision** help. You may use the Window's calculator. You sit in front of a computer and edit/assemble/run/debug the programming assignment. You do NOT have access the book, internet or manuals. You may not take this paper, scratch paper, or rough drafts out of the room. You may not access your network drive or the internet. You are not allowed to discuss this exam with other EE319K students until Thursday.

The following activities occurring during the exam will be considered scholastic dishonesty:
   1) running any program from the PC other than **Keil uVision**, or a calculator,
   2) communicating with other students by any means about this exam until Thursday,
   3) using material/equipment other than a pen/pencil.
Students caught cheating will be turned to the Dean of Students.

                                        Signed: _____   October 31, 2012

Procedure
        First, you will log onto the computer and download files from the web as instructed by the TAs.
Web site   xxxxxx
user: xxxxxx
password: xxxxxx

Unzip the folder placing it in a temporary folder. You are not allowed to archive this exam. Within **Keil uVision** open these files, put your name on the first comment line of the file **Exam2.s**. Before writing any code, please assemble and run the system. You should get output like the figure above (but a much lower score). You may wish create backup versions of your program. If you wish to roll back to a previous version, simply open one of the backup versions.
        My main program will call your subroutines multiple times, and will give your solution a performance score of 0 to 100. *You should not modify my main program or my example data.* When you have written your subroutines, you should run my main program, which will output the results to

the **UART#1** window. After you are finished, raise your hand and wait for a TA. The TA will direct you on how to complete the submission formalities. The TA will run your program in front of you and record your performance score on your exam cover sheet. The scoring page will not be returned to you. The submission guidelines are at the end of the exam handout.

Quad representation is like binary or hexadecimal representation, except the digits are not 0-1 or 0-F but 0,1,2,3. That is a quad representation is made of numbers 0, 1, 2 and 3 with place values 0, 4, 16 and 64.

Part a) The first subroutine, called **Pow4**, computes the product, $R0 \times 4^{R1}$, that is multiply contents of R0 with 4 raised to the power of the contents of R1.

>   **Input parameters:** *Input1*: R0 contains a quad number (range: 0-3).
>   *Input2*: R1 contains a number between 0 and 3.
>   **Output:**   R0 contains the result of multiplying *Input1* with 4^{Input2}

Think of this routine as computing the contribution of one quad digit (R0) based on its place value (4^{R1})

A typical calling sequence is
```
  MOV  R0,#2
  MOV  R1,#3
  BL   Pow4     ; must compute 2*4³ = 128
```

Part b) The second subroutine, called **QuadDec**, takes as input a 4-digit quad number and converts it to its decimal equivalent. The input is passed using the address of an array containing the four digits. Being quad digits the numbers in this array are restricted to the range 0 to 3.

>   **Input parameters:** R0 contains the address of a 4-element Array of quad numbers (range: 0-3)
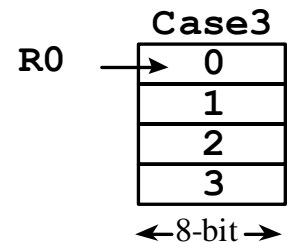>   **Output:**   R0 contains the decimal value corresponding to the quad number in input

```
Case3 DCB 0,1,2,3   ; QuadDec=27
```
A typical calling sequence is
```
  LDR  R0,=Case3   ; pointer to Case3 [0,1,2,3]
  BL   QuadDec     ; QuadDec computes and returns 27:
                   ;    0*4³ + 1*4² + 2*4¹ + 3*4⁰
```

Note that calling your subroutine in part (a) in part (b) will greatly reduce the amount of code you will need to write.

Submission Guidelines:

*   Log onto Blackboard and submit your Exam2.s source file into the Exam2 field. Be careful because only one submission will be allowed.

Memory access instructions
```
    LDR    Rd, [Rn]        ; load 32-bit number at [Rn] to Rd
    LDR    Rd, [Rn,#off]   ; load 32-bit number at [Rn+off] to Rd
    LDR    Rd, =value      ; set Rd equal to any 32-bit value (PC rel)
    LDRH   Rd, [Rn]        ; load unsigned 16-bit at [Rn] to Rd
    LDRH   Rd, [Rn,#off]   ; load unsigned 16-bit at [Rn+off] to Rd
    LDRSH  Rd, [Rn]        ; load signed 16-bit at [Rn] to Rd
    LDRSH  Rd, [Rn,#off]   ; load signed 16-bit at [Rn+off] to Rd
    LDRB   Rd, [Rn]        ; load unsigned 8-bit at [Rn] to Rd
    LDRB   Rd, [Rn,#off]   ; load unsigned 8-bit at [Rn+off] to Rd
    LDRSB  Rd, [Rn]        ; load signed 8-bit at [Rn] to Rd
    LDRSB  Rd, [Rn,#off]   ; load signed 8-bit at [Rn+off] to Rd
    STR    Rt, [Rn]        ; store 32-bit Rt to [Rn]
    STR    Rt, [Rn,#off]   ; store 32-bit Rt to [Rn+off]
    STRH   Rt, [Rn]        ; store least sig. 16-bit Rt to [Rn]
    STRH   Rt, [Rn,#off]   ; store least sig. 16-bit Rt to [Rn+off]
    STRB   Rt, [Rn]        ; store least sig. 8-bit Rt to [Rn]
    STRB   Rt, [Rn,#off]   ; store least sig. 8-bit Rt to [Rn+off]
    PUSH   {Rt}            ; push 32-bit Rt onto stack
    POP    {Rd}            ; pop 32-bit number from stack into Rd
    ADR    Rd, label       ; set Rd equal to the address at label
    MOV{S} Rd, <op2>       ; set Rd equal to op2
    MOV    Rd, #im16       ; set Rd equal to im16, im16 is 0 to 65535
    MVN{S} Rd, <op2>       ; set Rd equal to -op2
```
Branch instructions
```
    B    label   ; branch to label    Always
    BEQ  label   ; branch if Z == 1    Equal
    BNE  label   ; branch if Z == 0    Not equal
    BCS  label   ; branch if C == 1    Higher or same, unsigned ≥
    BHS  label   ; branch if C == 1    Higher or same, unsigned ≥
    BCC  label   ; branch if C == 0    Lower, unsigned <
    BLO  label   ; branch if C == 0    Lower, unsigned <
    BMI  label   ; branch if N == 1    Negative
    BPL  label   ; branch if N == 0    Positive or zero
    BVS  label   ; branch if V == 1    Overflow
    BVC  label   ; branch if V == 0    No overflow
    BHI  label   ; branch if C==1 and Z==0  Higher, unsigned >
    BLS  label   ; branch if C==0 or  Z==1  Lower or same, unsigned ≤
    BGE  label   ; branch if N == V    Greater than or equal, signed ≥
    BLT  label   ; branch if N != V    Less than, signed <
    BGT  label   ; branch if Z==0 and N==V  Greater than, signed >
    BLE  label   ; branch if Z==1 and N!=V  Less than or equal, signed ≤
    BX   Rm      ; branch indirect to location specified by Rm
    BL   label   ; branch to subroutine at label
    BLX  Rm      ; branch to subroutine indirect specified by Rm
```
Interrupt instructions
```
    CPSIE  I                ; enable interrupts  (I=0)
    CPSID  I                ; disable interrupts (I=1)
```

Logical instructions
```
    AND{S} {Rd,} Rn, <op2> ; Rd=Rn&op2     (op2 is 32 bits)
    ORR{S} {Rd,} Rn, <op2> ; Rd=Rn|op2     (op2 is 32 bits)
    EOR{S} {Rd,} Rn, <op2> ; Rd=Rn^op2     (op2 is 32 bits)
    BIC{S} {Rd,} Rn, <op2> ; Rd=Rn&(~op2) (op2 is 32 bits)
    ORN{S} {Rd,} Rn, <op2> ; Rd=Rn|(~op2) (op2 is 32 bits)
    LSR{S} Rd, Rm, Rs       ; logical shift right Rd=Rm>>Rs  (unsigned)
    LSR{S} Rd, Rm, #n       ; logical shift right Rd=Rm>>n   (unsigned)
    ASR{S} Rd, Rm, Rs       ; arithmetic shift right Rd=Rm>>Rs (signed)
```

```
    ASR{S} Rd, Rm, #n       ; arithmetic shift right Rd=Rm>>n  (signed)
    LSL{S} Rd, Rm, Rs       ; shift left Rd=Rm<<Rs (signed, unsigned)
    LSL{S} Rd, Rm, #n       ; shift left Rd=Rm<<n  (signed, unsigned)
```
Arithmetic instructions
```
    ADD{S} {Rd,} Rn, <op2> ; Rd = Rn + op2
    ADD{S} {Rd,} Rn, #im12 ; Rd = Rn + im12, im12 is 0 to 4095
    SUB{S} {Rd,} Rn, <op2> ; Rd = Rn - op2
    SUB{S} {Rd,} Rn, #im12 ; Rd = Rn - im12, im12 is 0 to 4095
    RSB{S} {Rd,} Rn, <op2> ; Rd = op2 - Rn
    RSB{S} {Rd,} Rn, #im12 ; Rd = im12 – Rn
    CMP    Rn, <op2>        ; Rn - op2      sets the NZVC bits
    CMN    Rn, <op2>        ; Rn - (-op2)   sets the NZVC bits
    MUL{S} {Rd,} Rn, Rm     ; Rd = Rn * Rm        signed or unsigned
    MLA    Rd, Rn, Rm, Ra   ; Rd = Ra + Rn*Rm     signed or unsigned
    MLS    Rd, Rn, Rm, Ra   ; Rd = Ra - Rn*Rm     signed or unsigned
    UDIV   {Rd,} Rn, Rm     ; Rd = Rn/Rm          unsigned
    SDIV   {Rd,} Rn, Rm     ; Rd = Rn/Rm          signed
```
Notes Ra Rd Rm Rn Rt represent 32-bit registers
```
    value    any 32-bit value: signed, unsigned, or address
    {S}      if S is present, instruction will set condition codes
    #im12    any value from 0 to 4095
    #im16    any value from 0 to 65535
    {Rd,}    if Rd is present Rd is destination, otherwise Rn
    #n       any value from 0 to 31
    #off     any value from -255 to 4095
    label    any address within the ROM of the microcontroller
    op2      the value generated by <op2>
```
Examples of flexible operand **<op2>** creating the 32-bit number. E.g., **Rd = Rn+op2**
```
    ADD Rd, Rn, Rm          ; op2 = Rm
    ADD Rd, Rn, Rm, LSL #n ; op2 = Rm<<n  Rm is signed, unsigned
    ADD Rd, Rn, Rm, LSR #n ; op2 = Rm>>n  Rm is unsigned
    ADD Rd, Rn, Rm, ASR #n ; op2 = Rm>>n  Rm is signed
    ADD Rd, Rn, #constant  ; op2 = constant, where X and Y are hexadecimal digits:
```
- produced by shifting an 8-bit unsigned value left by any number of bits
- in the form **0x00XY00XY**
- in the form **0xXY00XY00**
- in the form **0xXYXYXYXY**

R0	
R1	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	

General purpose registers

Stack pointer — R13 (MSP)
Link register — R14 (LR)
Program counter — R15 (PC)

Condition code bits
N negative
Z zero
V signed overflow
C carry or
   unsigned overflow

256k Flash ROM	0x0000.0000 ↓ 0x0003.FFFF
64k RAM	0x2000.0000 ↓ 0x2000.FFFF
I/O ports	0x4000.0000 ↓ 0x41FF.FFFF
Internal I/O PPB	0xE000.0000 ↓ 0xE004.0FFF