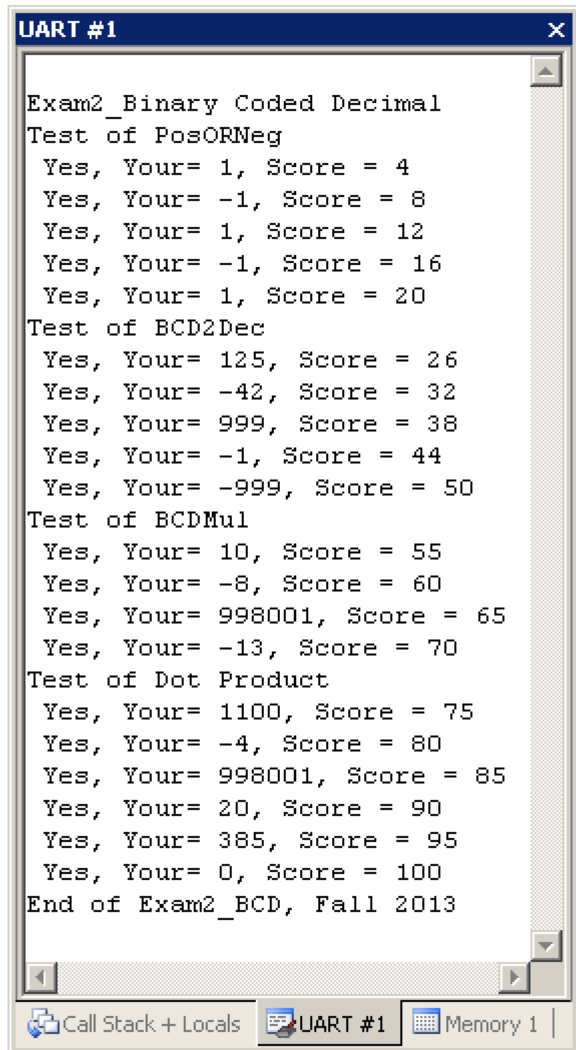
First:________________ Last:______________________ Circle: **MW12** **MW1:30** **TTh3:30**
V+Y **Gerst** **V+Y**

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= Run by TA at the checkout		TA:

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 70 minutes, so allocate your time accordingly. You must bring a laptop and are allowed to bring only some pens and pencils (no books, 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 Friday afternoon.



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 Friday,
3) using material/equipment other than a pen/pencil.

Students caught cheating will be turned to the Dean of Students.

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam:

Signed: ______________________________________ November 7, 2013

Procedure

First, you will log onto the computer and download files from the web as instructed by the TAs.

Web site http://users.ece.utexas.edu/~valvano/Exam2V
User: zxcv
Password: 2ww3

UNZIP the folder placing it in a temporary folder **ON THE DESKTOP**. You are not allowed to archive this exam. Within **Keil uVision** open the project, put your name on the first comment line of the file **Exam2.s**. Before writing any code, please build 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.* Each time you add a block of code, 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 numbers we will be dealing with in this exam are encoded in a format called **Binary Coded Decimal**, or BCD for short. In BCD-encoding, each decimal digit is encoded as a group of 4 bits (called a *nibble*) that can only take values between 0 and 9 (other values between 0xA and 0xF are not allowed and will not appear). We will work with 3-digit signed BCD numbers represented in 16-bits. This means the representation can be viewed as four nibbles. The least significant 3 nibbles are BCD-encoded decimal digits representing magnitude. The most significant nibble represents the sign: 0x0 means positive, 0xF means negative. This makes the range of valid numbers to be between -999 (BCD-format: 0xF999) to 999 (BCD-format: 0x0999). For example,

- The decimal number 43 will be represented as 0x0043 in 16-bit BCD-format.
- The decimal number -125 will be represented as 0xF125 in 16-bit BCD-format.
- The decimal number -1 will be represented as 0xF001 in 16-bit BCD-format.
- The decimal number 0 will be represented as 0x0000 in 16-bit BCD-format.

← 16-bit →

sign	hundreds	tens	ones

The exam has four parts a) through d), details of which are given in the starter code (Exam2.s):

Part a) The first subroutine you will write, called **`PosOrNeg`** determines if a given number in BCD-format is a positive or negative number. It returns a +1 or -1 accordingly.

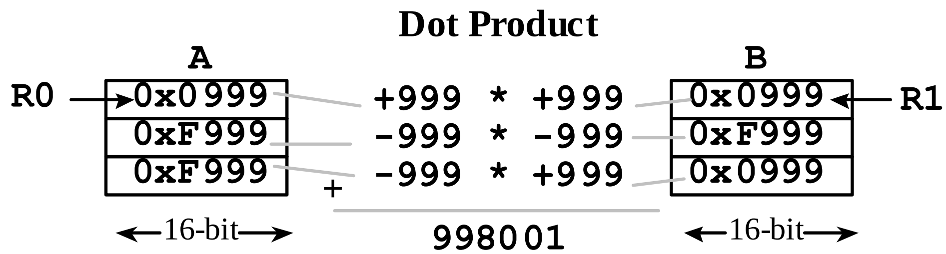
Part b) The second subroutine you will write, called **`BCD2Dec`** determines the decimal value of the given BCD-format input number.

Part c) The third subroutine you will write, called **`BCDMul`** computes the product of two given BCD-format input numbers and returns the value of the result.

Part d) The fourth subroutine you will write, called **`DotProduct`** computes the dot product of two equal-sized arrays of BCD-format input numbers and returns the value of the result. The dot product of two arrays **`A`** and **`B`** of size **`n`** is defined as the sum,

`A[0]*B[0] + A[1]*B[1] + A[2]*B[2] + … + A[n-1]*B[n-1]`.

That is, corresponding elements from two arrays are multiplied and the accumulated sum is the dot product. The arrays are passed to your subroutine as pointers in R0 and R1 and their common size is passed in R2.



Important Notes:

- Your subroutines should work for all cases shown in the starter file.
- Handle the simple cases first and the special cases last.

Note that calling your subroutine in part (a) in (b) and (b) in (c) and (c) in (d) 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.

Scratch Paper

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 or 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    {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`

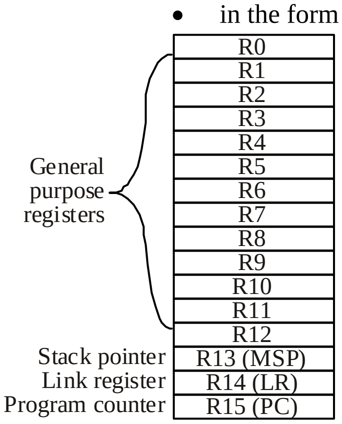
```
value   any 32-bit value: signed, unsigned, or address
{S}     if S is present, instruction will set condition codes
#im12   any value from 0 to 4095
#im16   any value from 0 to 65535
{Rd,}   if Rd is present Rd is destination, otherwise Rn
#n      any value from 0 to 31
#off    any value from -255 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
```

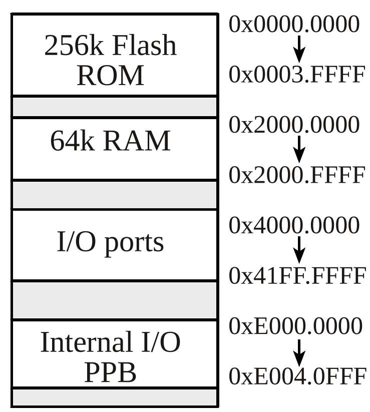
, where **X** and **Y** are hexadecimal digits:

- 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**



Condition code bits

N negative
Z zero
V signed overflow
C carry or unsigned overflow




```
SPACE  4        ; allocates 4 bytes
DCB    1,2,3,4  ; defines 4 bytes with initial values
DCW    1,2,3,4  ; defines 4 halfwords with initial values
DCD    1,2,3,4  ; defines 4 words with initial values
```