Exam 1

Date: October 2, 2014

UT EID: ________________

Printed Name: ________________________________ ________________________________
Last, First

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:

Signature: ____

Instructions:

- Closed book and closed notes. No books, no papers, no data sheets (other than the last two pages of this Exam)
- No devices other than pencil, pen, eraser (no calculators, no electronic devices), please turn cell phones off.
- Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided. *Anything outside the boxes/blanks will be ignored in grading*. You may use the back of the sheets for scratch work.
- You have 75 minutes, so allocate your time accordingly.
- For all questions, unless otherwise stated, find the most efficient (time, resources) solution.
- Unless otherwise stated, make all I/O accesses friendly.
- *Please read the entire exam before starting.*

Problem 1	10	
Problem 2	6	
Problem 3	4	
Problem 4	10	
Problem 5	20	
Problem 6	10	
Problem 7	10	
Problem 8	15	
Problem 9	15	
Total	100	

(10) Question 1. State the term, symbol, or expression that is best described by each definition.

Part a) A property of memory that describes the fact that when power is removed and subsequently restored, the contents of the memory is lost.

volatile

Part b) A debugging instrument or tool that measures voltage versus time for multiple digital signals.

Logic analyzer or oscilloscope

Part c) A drawing that describes how information is passed from one module to another in a system. An arrow from circle A to circle B means information is passed from software module A to software module B.

Data flow graph

Part d) A collection of wires in a computer that allows data to travel from one module to another within the computer.

Bus

Part e) A processor in which the operands to ALU instructions are never a memory location uses what type of generic architecture? (Hint: the answer to this question is not ARM, THUMB, or Cortex-M, but rather the general architecture type.)

Load/store or RISC

Part f) The electrical property that specifies the number of electrons per second that are traveling down a wire.

Current or amps

Part g) This C operator will perform the exclusive or of two numbers in a bit-wise fashion.

^

Part h) A C program calls an assembly subroutine. When the assembly subroutine returns, where can the return value be found? (Hint: AAPCS)

In register R0

Part i) This declaration is used to create a variable in C that can take on the values from -20 to +200. Pick the most efficient format.

`int16_t`

Part j) A debugging feature that causes execution to halt, and control returns to the debugger, when your software executes an instruction at a specific location in your code.

Breakpoint

(6) Question 2. Octal means base 8 in the same way binary means base 2, decimal means base 10, and hexadecimal means base 16. This means each octal digit can be 0, 1, 2, 3, 4, 5, 6, or 7. What is the value of the unsigned four-digit octal number 1036? Give your answer as a decimal number. Show your work.

3 points for the basis: Basis is 8^0, 8^1, 8^2, 8^3 = 2^0, 2^3, 2^6, 2^9 = 1, 8, 64, 512
3 points for the value Value = 512*1+64*0+8*3+6*1 = 512+24+6 = 542

(4) Question 3 Consider the following 8-bit addition (assume registers are 8 bits wide, and assume the condition code bits are set in a way similar to the Cortex M4)

```
Load   0x80 into R1
Load   0x20 into R2
Adds   R3 = R1+R2, setting the condition codes
```

a. What is the 8-bit result in Register R3 (as an unsigned decimal)?

0x80 is 128 and 0x20 is 32, 128+32 = 160
Another way: 0x80+0x20 = 0xA0, which is 10*16 = 160
1 point, no partial credit

b. What is the 8-bit result in Register R3 (as a signed decimal)?

0x80 is -128 and 0x20 is 32, -128+32 = -96
Another way: -0x80+0x20 = -0x60 which is -6*16 = -96
1 point, no partial credit

c. What will be the value of the carry (C) bit?

Carry bit is clear (C=0) because 160 is the correct answer
1 point, no partial credit

d. What will be the value of the overflow (V) bit?

Overflow bit is clear (V=0) because -96 is the correct answer
1 point, no partial credit

(10) Question 4. Write an assembly subroutine that initializes Port D, making PD4 an output, and making PD3, PD2, PD1 PD0 inputs. This subroutine is called once at the start of execution of the system. **All accesses to I/O registers must be friendly.** Your subroutine will set the clock, direction, and enable registers (in this question do not worry about AFSEL, PUR, PDR, AMSEL, or PCTL). Basically fill in the instruction or instructions for the following five boxes. Boxes may contain 0, 1, or 2 instructions. Do not assume DIR, DEN or DATA registers have been cleared by the reset operation. Comments are not needed.

```
GPIO_PORTD_DATA_R  EQU 0x400073FC
GPIO_PORTD_DIR_R   EQU 0x40007400
GPIO_PORTD_DEN_R   EQU 0x4000751C
SYSCTL_RCGCGPIO_R  EQU 0x400FE608
PortD_Init
    LDR R1, =SYSCTL_RCGCGPIO_R
    LDR R0, [R1]
```

```
  ORR R0,R0,#0x08 ; enable Port D
```
1 point for ORR, 1 point for #0x08
e.g., MOV R0,#0x08 is 1 point out of 2 possible

```
    STR R0, [R1]
```

```
   NOP
   NOP     ; wait for clock to stabilize
```
1 point for any wait, no partial credit

```
    LDR R1, =GPIO_PORTD_DIR_R
    LDR R0, [R1]
```

```
   ORR R0,R0,#0x10 ; PD4 output
   BIC R0,R0,#0x0F ; PD3-0 inputs
```
1 point for ORR #0x10, 1 point for BIC, 1 point for #0x0F
e.g., MOV R0,#0x10 is 1 point out of 3 possible

```
    STR R0, [R1]
    LDR R1, =GPIO_PORTD_DEN_R
    LDR R0, [R1]
```

```
   ORR R0,R0,#0x1F ; enable PD4-0
```
1 point for ORR, 1 point for #0x1F (ok if 0xFF)

```
    STR R0, [R1]
```

```
   BX LR     ; return from subroutine
```
2 points, no partial credit for other or nothing

(20) Question 5. The inputs are on Port D pins 3,2,1,0. The output is PD4. Design a detector that reads a 4-bit number on PD3 – PD0 and activates a detection light on PD4. First, read the input and count the number of input pins, PD3 – PD0, that are high. If the count is odd, set PD4 high; if the count is even, clear PD4 low. For example, if PD3 – PD0 is 1011 then there are an odd number of pins that are high, the pattern is detected, and the PD4 should be set high. When such a pattern is detected turn ON the light otherwise turn it off. You will design pieces of the solution in two parts. You may assume the subroutine in Question 4 has been called making PD4 an output and making PD3 – PD0 inputs.

Part a) Write an assembly subroutine called *Detect* that takes a 4-bit input in a register (the remaining bits are zero). Returns a 1 if pattern is detected, 0 otherwise. *Detect* must be AAPCS compliant.

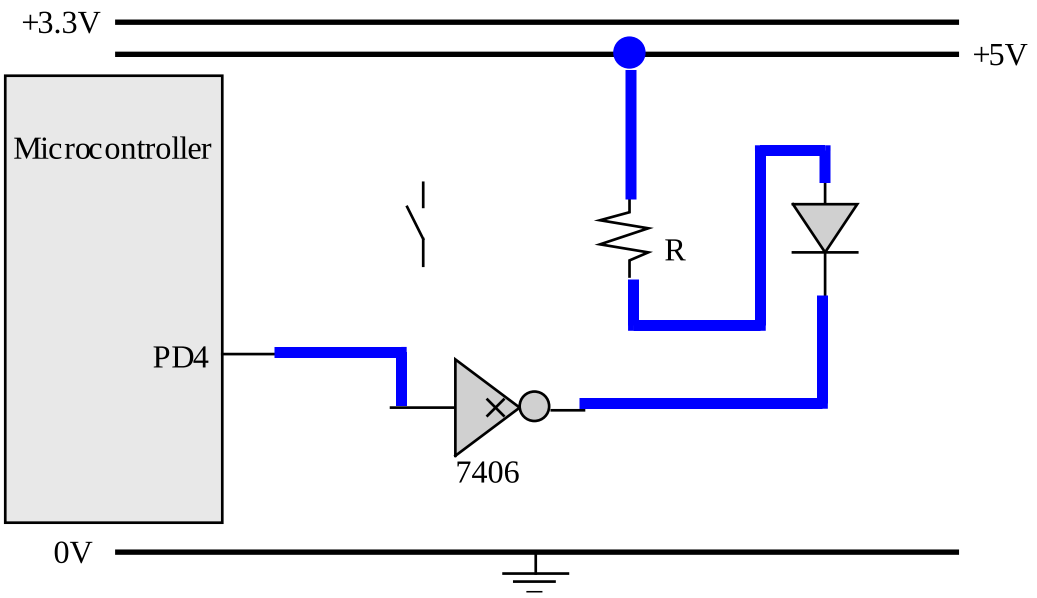
```
Detect    PUSH {R4,LR} ; R4 saved for compliance
          MOV R4,#0
          LSR R0,#1    ; R0 has input (Compliant)
          BCC Bit1     ; C bit has bit 0 which is 0
          ADD R4,#1
Bit1      LSR R0,#1
          BCC Bit2     ; C bit has bit 1 which is 0
          ADD R4,#1
Bit2      LSR R0,#1
          BCC Bit3     ; C bit has bit 2 which is 0
          ADD R4,#1
Bit3      LSR R0,#1
          BCC Done     ; C bit has bit 3 which is 0
          ADD R4,#1
Done      MOV R0,R4    ; Compliant by returning in R0
          AND R0,R0,#1  ;0 or 1
          POP {R4,PC}  ; R4 restored for compliance
```

```
Detect2  MOV R1,R0      ;copy
         LSR R1,#1
         XOR R0,R1,R0   ;Bit1^Bit0
         LSR R1,#1
         XOR R0,R1,R0   ;Bit2^Bit1^Bit0
         LSR R1,#1
         XOR R0,R1,R0   ;Bit3^Bit2^Bit1^Bit0
         AND R0,R0,#1   ;0 or 1
         BX LR
```

Part b) Complete the caller code loop in assembly that repeatedly reads the 4-bit number, calls Detect and appropriately manipulates the light. Execute these steps over and over.

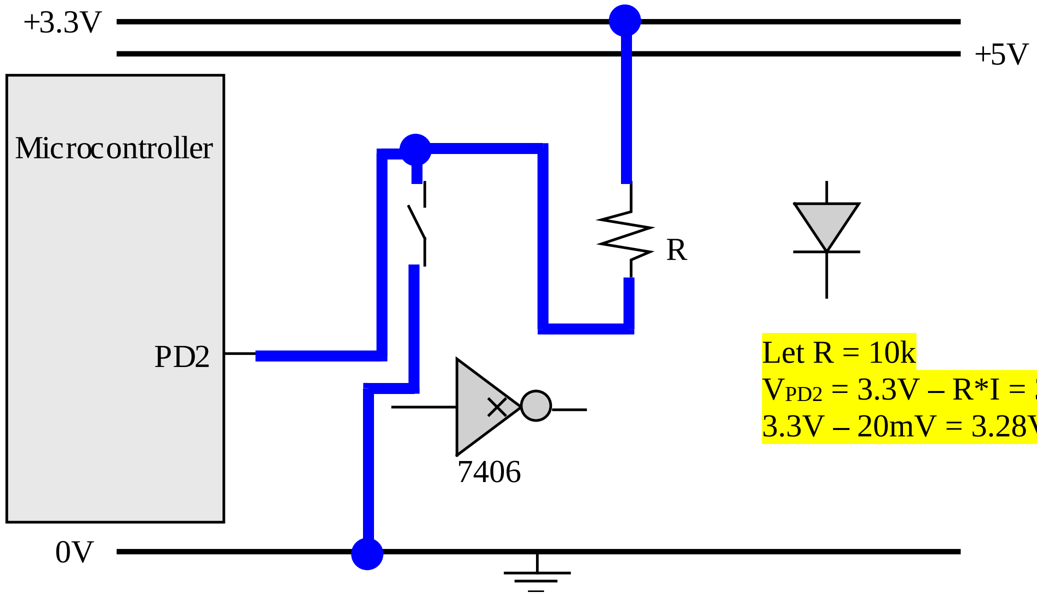
```
         LDR   R4,=GPIO_PORTD_DATA_R
Loop     LDR   R0,[R4]  ;input Port D
         AND   R0,#0x0F ;mask (could skip this)
         BL    Detect   ;1 if odd,0 if even
         LSL   R0,R0,#4 ;16 if odd, 0 if even
         STR   R0,[R4]  ;output to LED
         B     Loop
```

(10) Question 6. You are to interface an external LED on Port D pin 4 that operates using positive logic. You have an LED whose desired brightness requires an operating point of (V_d , I_d) = (1.5V, 15mA). Given the TM4C microcontroller output low V_{OL} ranges between (0V,0.5V) and output high V_{OH} ranges between (2.4V,3.3V). The 7406 driver's V_{OL} is 0.5V. Show the calculation used to find the resistor value needed and draw the circuit below by connecting the needed elements: R = (5-1.5-0.5V)/15mA = (5-2V)/15mA= (3V)/15mA = 200 ohms



(10) Question 7. You are to interface an external Switch on Port D pin 2 that operates using negative logic by using the needed elements in the following figure.

(8) Part a) Given the TM4C microcontroller limits the current flow into it to 2 μA calculate the voltage at Port D pin 2 when the switch is open? R = any resistance from 1k to 1M



Let R = 10k
V_{PD2} = 3.3V – R*I = 3.3V-10k*2uA = 3.3V – 20mV = 3.28V

(2) Part b) If you were using an internal resistor instead (of an external one) what extra line(s) would you add to the initialization for port D. (C or Assembly is okay)

```
GPIO_PORTD_PUR_R |= 0x04; // need pullup in PD2
```

(15) Question 8. The right column shows Cortex M assembly for a function called **`Calc`**. You will write the corresponding C code in the left column. Think of the assembly as code generated by the C compiler. You must write the C code that corresponds to the functionality defined in the assembly code. Do not optimize, just translate the assembly into C.

```c
#include <stdint.h> // C99
uint16_t Num;
uint16_t Cnt;
// 2 points, uint16_t variables



uint16_t Calc(uint16_t input){
// 1 points, uint16_t type
// 1 points, function
// 1 point, input parameter
  Num = input;
// 1 point

  for(Cnt=0; Cnt<10; Cnt++){
// 4 points, while{} or for(){}

    if(Num < 100){
// 1 point, if

      Num = Num + Cnt;
// 1 point, Num = Num+Cnt

    }else{
// 1 point, else
      Num = Num + 1;
// 1 point, Num = Num+1

  }




  return Num;
// 1 points, return
}
```

```
      AREA  Data,ALIGN=2
Num   SPACE 2
Cnt   SPACE 2
      AREA |.text|, CODE, ALIGN=2
      THUMB
      EXPORT  Calc
;Input is 16-bit unsigned in R0
;Output is 16-bit unsigned in R0
Calc    LDR   R1, =Num
        STRH R0, [R1] ;R0 is input

        LDR   R2, =Cnt
        MOV   R3, #0
        STRH R3, [R2]
        B     labelD

labelA LDRH R0, [R1]
       CMP   R0, #0x64
       BHS   labelB
       LDRH R3, [R2]
       ADD   R0, R0, R3
       STRH R0, [R1]
       B     labelC

labelB ADD   R0, R0, #1
       STRH R0, [R1]

labelC LDRH R3, [R2]
       ADD   R3, R3, #1
       STRH R3, [R2]

labelD LDRH R3, [R2]
       CMP   R3, #0x0A
       BLS   labelA

       LDRH R0, [R1]
;R0 is the 16-bit return value
       BX   LR
```

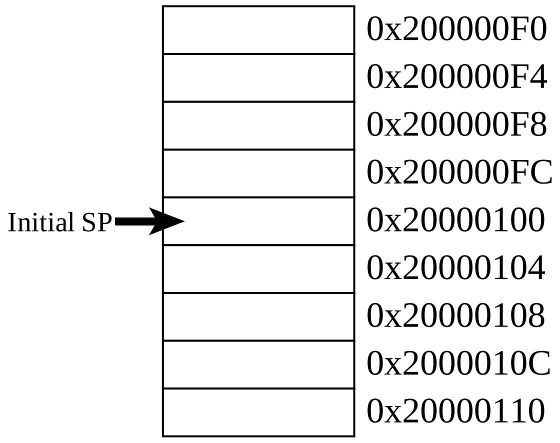
(15) Question 9. Consider the following assembly code. Execution begins at line 127 in **`main`**, and the initial SP equals 0x20000100.

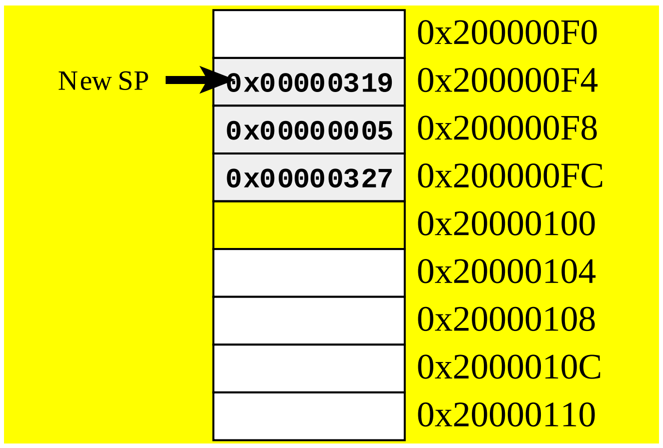
```
111:                    ;R0 dividend
112:                    ;R1 divisor
113:                    ;R0 is returned with remainder
114: 0x000002F8 B500      mod  PUSH {LR}
115: 0x000002FA FBB0F3F1       UDIV R3, R0, R1
116: 0x000002FE FB03F301       MUL  R3, R3, R1
117: 0x00000302 EBA00003       SUB  R0, R0, R3
118: 0x00000306 BD00           POP  {PC}
119: 0x00000308 B510      fun  PUSH {R4,LR}
120: 0x0000030A F04F040A       MOV  R4, #10
121: 0x0000030E F04F0010  loop MOV  R0, #16
122: 0x00000312 4621           MOV  R1, R4
123: 0x00000314 F7FFFFF0       BL   mod
124: 0x00000318 3C01           SUBS R4, #1
125: 0x0000031A D1F8           BNE  loop
126: 0x0000031C BD10           POP  {R4,PC}
127: 0x0000031E F04F0405  main MOV  R4, #5
128: 0x00000322 F7FFFFF1       BL   fun
129: 0x00000326 E7FE      done B    done
```

Part a) What is the SP when execution reaches line 115 for the first time?

SP = 0x2000.00F4 because three items are pushed, each push decrements SP by 4. (4 points)

Part b) What are all the values stored on the stack as it executes from line 127 to line 115? Show each value as a 32-bit hexadecimal number into the appropriate place on the stack picture. The addresses and machine code are included for each line. (-3 each wrong answer, -3 for swap)





Notice the LR is always odd, so when the function returns, we remain in Thumb mode. Do not take off points if LR values are given as even numbers

Part c) What addressing mode does the BL instruction in line 123 use?

This instruction uses PC-relative addressing. The number is a signed 24-bit value, meaning -16. The target address from 318 to 2F8 is -32. So the branch location is PC+2*value. (1 points)

Part d) What does **`B500`** at line 114 represent?

`B500` is the machine code or object code representing the instruction **`PUSH {LR}`** (1 points)

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{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`

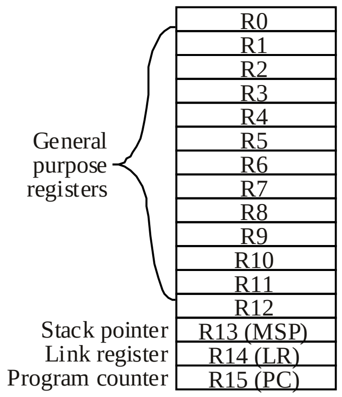
```
     value   any 32-bit value: signed, unsigned, or address
     {S}     if S is present, instruction will set condition codes
     #im12   any value from 0 to 4095
     #im16   any value from 0 to 65535
     {Rd,}   if Rd is present Rd is destination, otherwise Rn
     #n      any value from 0 to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

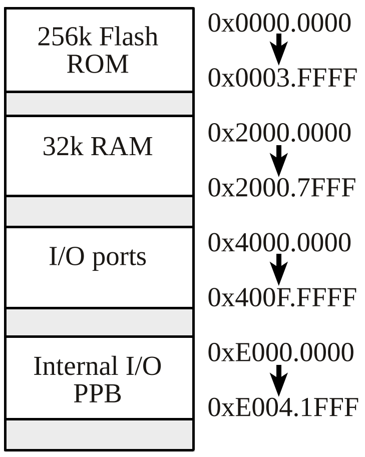
, where **`X`** and **`Y`** are hexadecimal digits:

- 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`**



Condition code bits

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


256k Flash ROM
32k RAM
I/O ports
Internal I/O PPB
0x0000.0000
0x0003.FFFF
0x2000.0000
0x2000.7FFF
0x4000.0000
0x400F.FFFF
0xE000.0000
0xE004.1FFF