Exam 1

Date: Feb 26, 2015

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	14	
Problem 4	10	
Problem 5	20	
Problem 6	5	
Problem 7	5	
Problem 8	15	
Problem 9	15	
Total	100	

(10) Question 1. State the term, symbol, instruction or expression that best answers the question.

(1) Part a) The drawing with circles representing software modules. An arrow from circle A to circle B means software in A invokes a function in module B.

Call graph

(1) Part b) This declaration is used to create a variable in C that can take on the values from 20 to +40000. Pick the most efficient format.

uint16_t *or* **unsigned short**

(1) Part c) You are writing a function with exactly three input parameters. According to ARM Architecture Procedure Call Standard, how should you pass the three parameters?

R0, R1, R2

(1) Part d) According to ARM Architecture Procedure Call Standard, which registers must be preserved?

R4-R11 *or* **R4-R11,LR,SP**

(1) Part e) The term used to define the amount of work that can be done. Units are Joules.

Energy

(1) Part f) The term that defines the subset of a number system from which all elements of that set can be derived.

Basis

(1) Part g) This C operator will perform the logic or of two Booleans (the inputs and outputs are True or False).

||

(3) Part h) Write the assembly code to create a 16-bit signed variable called **Num**. Include the details that will place the variable in RAM

```
      AREA  data
Num   SPACE 2
```

(6) Question 2. Complete the following table. Each row in the table contains an equal value expressed in binary, hexadecimal, unsigned decimal, and signed decimal. Assume each value is 8 bits.

Binary	Hexadecimal	Unsigned Decimal	Signed Decimal
10000001	0x81	129	-127
11111110	0xFE	254	-2
11010100	0xD4	212	-44
01000000	0x40	64	64

(14) Question 3

a. **(4)** 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). What are the condition code bits?

```
Load   0x88 into R1
Load   0xC8 into R2
Adds   R3 = R1+R2, setting the condition codes
```

N	Z	V	C
0	0	1	1

b. **(10)** Complete the table below by marking with an X which branches will be taken or not taken as a result of instructions below. Assume registers are 8 bits wide, and assume the branch instructions and condition code bits are similar to the Cortex M4.

```
Load   #100 into R1
Load   #200 into R2
Subs   R3 = R1-R2  ; setting the condition codes
```

Branch Instruction	Taken	Not Taken
B target	x	
BL target	x	
BEQ target		x
BNE target	x	
BCS target		x
BCC target	x	
BVS target	x	
BVC target		x
BLO target	x	
BLT target		x

(10) Question 4. Complete the assembly subroutine that initializes **Port B**. You should make **PB7 PB0** outputs, and make **PB4 PB1** 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. You must fill in the instruction or instructions for the following four boxes. Each box 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_PORTB_DATA_R  EQU 0x400053FC
GPIO_PORTB_DIR_R   EQU 0x40005400
GPIO_PORTB_DEN_R   EQU 0x4000551C
SYSCTL_RCGCGPIO_R  EQU 0x400FE608
PortB_Init
    PUSH {R4,R5}
    LDR R5, =SYSCTL_RCGCGPIO_R
    LDR R4, [R5]
```

```
ORR R4,R4,#0x02 ; enable Port B
```
1 point for ORR, 1 point for #0x02
e.g., MOV R0,#0x02 is 1 point out of 2 possible, comments not needed

```
    STR R4, [R5]
    NOP
    NOP
    LDR R5, =GPIO_PORTB_DIR_R
    LDR R4, [R5]
```

```
ORR R4,R4,#0x81 ; PB7,PB0 outputs
BIC R4,R4,#0x12 ; PB4,PB1 inputs
AND R4,R2,#0xED ; instead of BIC
```
2 points for ORR #0x81, 1 point for BIC, 1 point for #0x12

```
    STR R4, [R5]
    LDR R5, =GPIO_PORTB_DEN_R
    LDR R4, [R5]
```

```
ORR R4,R4,#0x93 ; enable PB7,PB4,PB1,PB0
```
1 point for ORR, 1 point for #0x93
1 point for works, but unfriendly

```
    STR R4, [R5]
```

```
POP {R4,R5}
BX LR    ; return from subroutine
```
1 point for POP, 1 point for BX LR

(20) Question 5. You have been hired to build a Morse code distress signal detector. The **detector monitors for a special SOS bit pattern "000111000" on Port B, Pin 7 (i.e., PB7).** Upon detecting the pattern you must **activate a LED using negative logic on PB2.** PB7 is input and PB2 is output. Assume from Question 4 that the port has been activated, and that PB7 and PB2 are configured to be input and output, respectively.

Part a) Write a subroutine in C or assembly called SOS_Detector that first reads PB7 nine times very quickly. If the nine consecutive inputs match the pattern "**000111000**", then return a 1, otherwise return a 0. SOS_Detector **must be AAPCS compliant**.

```
int32_t SOS_Detector(void){ uint32_t i,data=0;
  for(i=0; i<9; i++){
    data = (data<<1)|(GPIO_PORTB_DATA_R&0x80);
  }
  if(data == (0x38<<7)){
    return 1;
  }
  return 0;
}
```

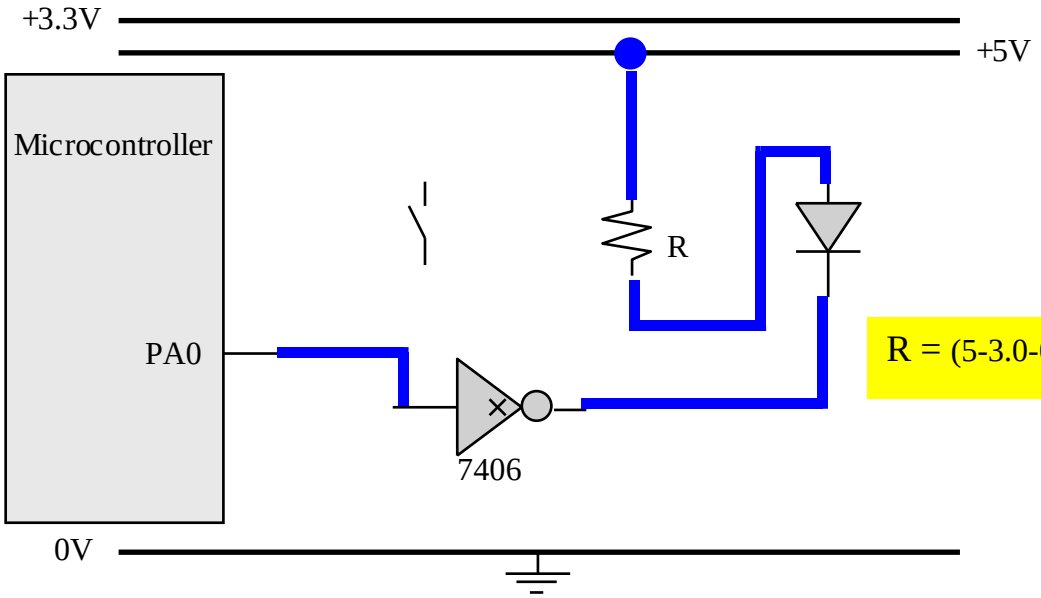
```
SOS_Detector
     MOV R0,#0
     LDR  R1,=GPIO_PORTB_DATA_R
     MOV  R2,#9
loop LDR  R3,[R1] ; PORTB
     AND  R3,#0x80 ; PB7
     LSL  R0,#1
     ORR  R0,R0,R3 ;combine bits
     SUBS R2,#1
     BNE  loop     ;9 times
     CMP  R0,#(0x38<<7)
     BEQ  yes
no   MOV  R0,#0
     B    done
yes  MOV  R0,#1
done BX   LR
```

Part b) Integrate the SOS_Detector from above into a loop that turns on the LED (using the logic specified above) whenever the SOS pattern is detected, **waits two cycles** and turns off the LED and continues to check for the SOS signal. Execute these steps over and over. Be friendly.

```
while(1){
  GPIO_PORTB_DATA_R &= ~(SOS_Detector()<<2);
  delay = GPIO_PORTB_DATA_R;
  GPIO_PORTB_DATA_R |= 0x04;
}
while(1){
  if(SOS_Detector()){
    GPIO_PORTB_DATA_R &= ~0x04;
    delay = GPIO_PORTB_DATA_R;
    GPIO_PORTB_DATA_R |= 0x04;
  }
}
```

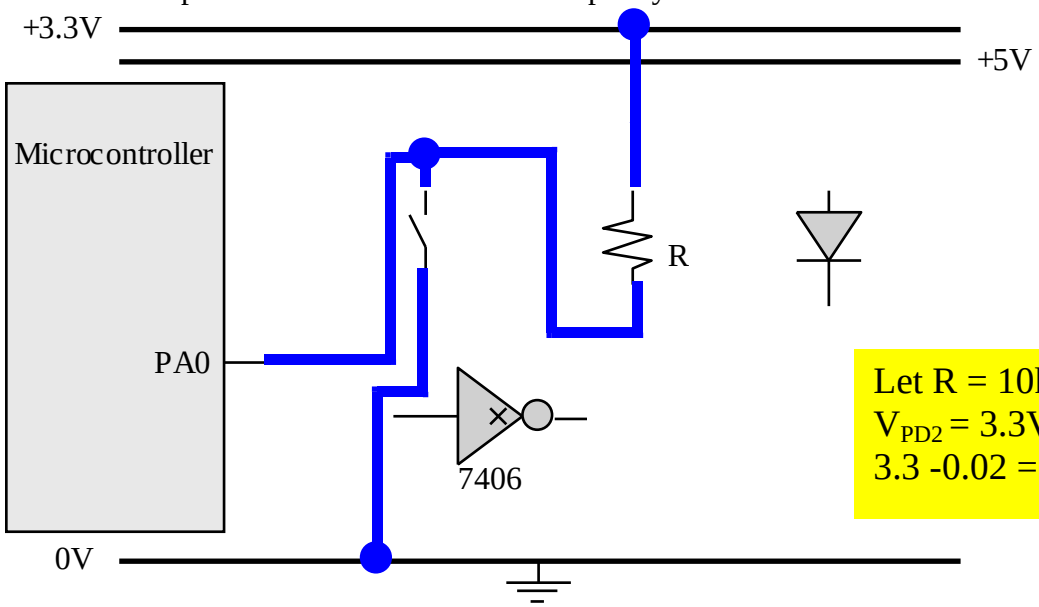
```
loop2 BL   SOS_Detector
      CMP  R0,#0   ;0 means no
      BEQ  loop2
      LDR  R1,=GPIO_PORTB_DATA_R
      BIC  #0x04   ; negative
      STR  R0,[R1] ; turn on
      ORR  R0,#4   ;
      NOP
      STR  R0,[R1]  ;turn off
      B    loop2
```

(5) Question 6. You are to interface an **external LED** on Port A pin 0 that operates using positive logic. You have an LED whose desired brightness requires an operating point of (V_d, I_d) = (3V, 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 = (5-3.0-0.5V)/15mA = 100 ohms

(5) Question 7. You are to interface an **external Switch** on Port A pin 0 that operates using negative logic by using the needed elements in the following figure. Given the TM4C microcontroller limits the current flow into it to 2 µA calculate the voltage at Port A pin 0 when the switch is open. Choose a value for R and specify its value.



Let R = 10kohms
V_{PD2} = 3.3V – 10k * 2uA =
3.3 -0.02 = 3.28V

(15) Question 8. Given below is the assembly code for an AAPCS compliant subroutine called **`func`**.

(8) Part a) Give an equivalent C function that achieves the same purpose as the given assembly code in **`func`**.
Hint: *Do not try to translate line-by-line, you have no access to the stack in C*

Assembly code

```
func CMP  R0, R1
     BLT  L1
     PUSH {R1,LR}
     POP  {R0,LR}
L1   BX   LR
```

C code

```
int32_t func(int32_t x, int32_t y){
   if(x >= y){
      return(y);
   } else {
      return(x);
}
```

(2) Part b) Briefly explain what the **`func`** subroutine does

Returns the smaller of the two signed 32-bit inputs

(If they say it orders the values in the two registers then they are wrong – AAPCS compliant means only one output)

(5) Part c) Write a C function '*g*' to perform the following functionality:
Step1: Read the value from global variable **`input`**, divide it by two;
Step2: Call the function '`func`' from (a) and pass it two inputs: the result of step 1 and the constant value 50;
Step3: Update the value of the global variable **`output`** by adding the value returned by the call in step 2 to it;

```
int32_t input;
int32_t output;
void g(void){
   output += func(input/2,50);




// Hint: Can be done in 1 line with no extra variables,
// however extra variables are allowed if implemented properly
}
```

(15) Question 9. This question deals with the stack.

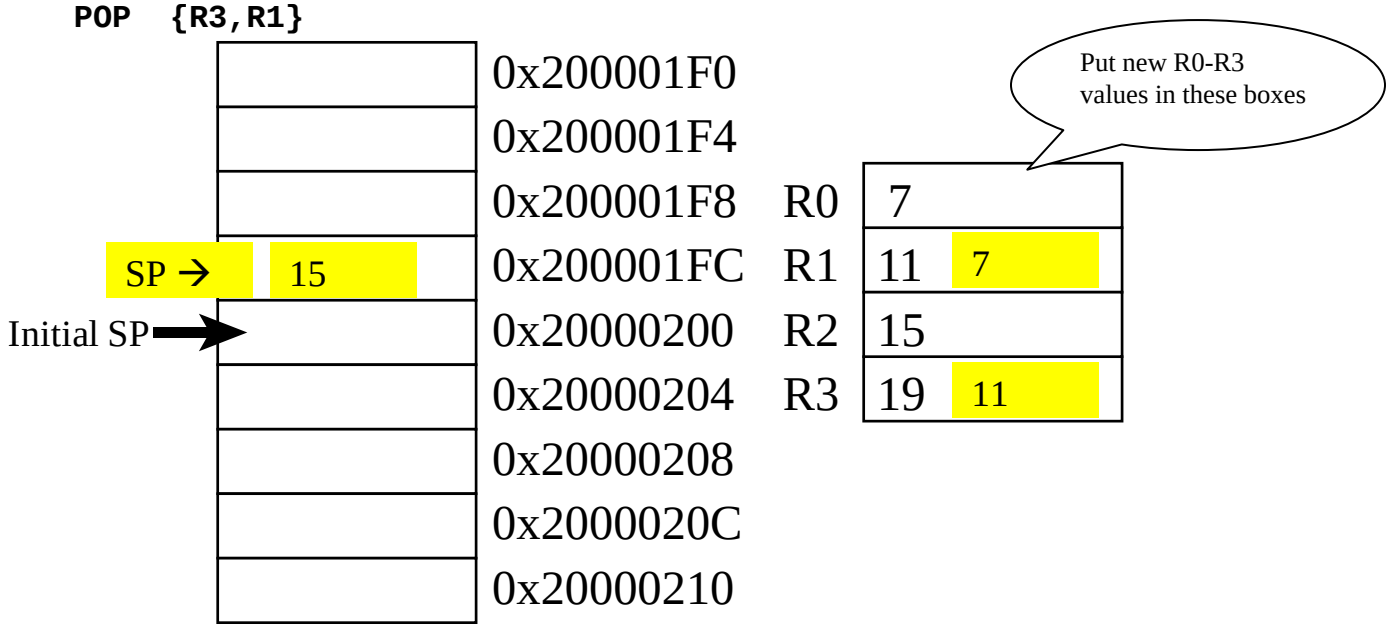
Part a) In AAPCS, the fifth parameter is located on top of the stack. Please write the following function in assembly using AAPCS. Your subroutine must balance the stack.

```
int32_t add_5(int32_t n1, int32_t n2, int32_t n3,
              int32_t n4, int32_t n5){
    return n1 + n2 + n3 + n4 + n5;}
```

```
add_5 POP  {R12}  // get n5
      PUSH {R12}  // balance the stack
      ADD  R0,R1  // n1+n2
      ADD  R0,R2  // n1+n2+n3
      ADD  R0,R3  // n1+n2+n3+n4
      ADD  R0,R12 // return n1+n2+n3+n4+n5
      BX   LR
add_5 ADD  R0,R1  // n1+n2
      ADD  R0,R2  // n1+n2+n3
      ADD  R0,R3  // n1+n2+n3+n4
      LDR  R12,[SP] // R12 is n5
      ADD  R0,R12 // return n1+n2+n3+n4+n5
      BX   L
```

Part b) Assume the stack pointer (SP) is initially equal to 0x20000200, and registers R0, R1, R2, R3 are 7, 11, 15, and 19 respectively. Draw the contents of the stack and the values in registers R0, R1, R2, R3 after these two instructions are executed. Also, label the new SP on the figure.

```
PUSH {R0,R1,R2}
POP  {R3,R1}
```



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, return address in LR
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
```

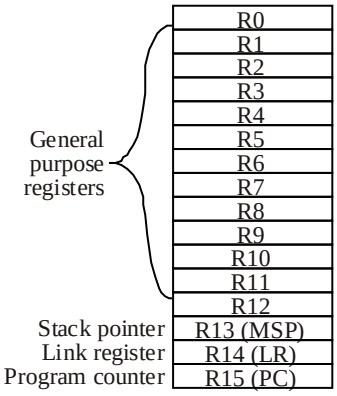
Notes `Ra Rd Rm Rn Rt represent 32-bit registers`

```
     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