The University of Alabama in Huntsville
Electrical and Computer Engineering Department
CPE 221 01
Test 1 Solution
Spring 2018

This test is closed book, closed notes. You may not use a calculator. You should have the 6 page ARM Instruction Reference. You must show your work to receive full credit.

1. (1 point) An _embedded_ computer is one that is dedicated to doing one task, such as the one found in washing machines.
2. (1 point) _Registers_ are used to hold data when faster access than main memory is needed.
3. (1 point) _Branch_ instructions can alter the normal flow of control from executing the next instruction in sequence.
4. (1 point) In RTL, the symbol _←_ is used to indicate a data transfer.
5. (1 point) _Direct, indexed, immediate, indirect_ is an example of an addressing mode found in processors.
6. (10 points) Represent 528 and -719 as signed (2s complement) 16-bit numbers

```
16     0   r 2              16     0   r  2
16     2   r 1              16     2   r 12
16    33   r 0              16    44   r 15
16   528                    16   719

+528 = 0000 0010 0001 0000  = 0x0210
+719 = 0000 0010 1100 1111
-719 = 1111 1101 0011 0001  = 0XFD31
```

7. (10 points) If `r1 = 0x0000 0FFF` and `r2` = 7, what is the value of `r0` after each of the following instructions has been executed (assume that each instruction uses the same data)?

(a) `ADD r0, r1, r1, LSL #2_111`

```
R1 =             0000 0000 0000 0000 0000 1111 1111 1111
R1 LSL #2_111 =  0000 0000 0000 0111 1111 1111 1000 0000
Sum =            0000 0000 0000 1000 0000 1111 0111 1111
Sum = 0x0008 0F7F
```

(b) `ADD r0, r1, r1, ROR #21`

```
R1 =             0000 0000 0000 0000 0000 1111 1111 1111
R1 ROR 21     =  0000 0000 0111 1111 1111 1000 0000 0000
Sum =            0000 0000 1000 0000 0000 0111 1111 1111
Sum = 0x0080 07FF
```

(c) `ADD r0, r1, r1, LSR r2`

```
R1 =             0000 0000 0000 0000 0000 1111 1111 1111
R1 LSR r2 =      0000 0000 0000 0000 0000 0000 0001 1111
Sum =            0000 0000 0000 0000 0001 0000 0001 1110
Sum = 0x0000 101E
```

8. (10 points) For each of the following operations on 6 bit signed numbers, calculate the values of the C, Z, V, and N flags

(a) 101011 + 001101

```
001111   C
 101011
 001101
 111000  S
```

CZVN = 0001

(b) 111111 + 001001

```
111111
 111111
 001001
 001000
```

CZVN = 1000

9. (15 points) Assume that `r2` contains the initial value `0xFF001000` and that `r0` contains `0xFFFF 8700`. Explain the effect of each of the following instructions, and give the value in `r2` after each instruction executes. Use register transfer notation.

(a) `LDR  r1, [r2]`

r1 ← M[r2], r2 = 0xFF00 1000

(b) `STR  r1, [r2, #2_110101]`

M[r2 + 0x35] ← r1, r2 = 0xFF00 1000

(c) `STR  r1, [r2, #0x2C]!`

r2 ← r2 + 0x2C, M[r2] ← r1, r2 = 0xFF00 102C

(d) `STR  r1, [r2], #16`

M[r2 + 16] ← r1, r2 ← r2 + 16 = 0xFF00 1010

(e) `STR  r1, [r2, r0, ASR #7]`

M[r2+ r0>>7] ← r1, r2 = 0xFF00 1000

```
r0 =         1111 1111 1111 1111 1000 0111 0000 0000
ASR 7 r0 =   1111 1111 1111 1111 1111 1111 0000 1110
r2 =         1111 1111 0000 0000 0001 0000 0000 0000
Sum =        1111 1111 0000 0000 0000 1111 0000 1110
```

10. (25 points) Consider the following ARM program. Trace the values of the registers shown as they change during program execution. Also, trace the writes to memory by the `STR` instructions. There may be unused columns or rows in the tables. If you need to add columns or rows, you may do so. `DCD 1` reserves one word of storage and sets it equal to 1. `SPACE 3` reserves 3 bytes of memory but does not give those bytes a value.

```
            AREA   COUNT_ONES_BIT0_MASK, CODE, READONLY
            ENTRY
0           LDR    r1, num
4           LDR    r3, mask
8           SUB    r2, r2, r2        ; initialize r2 to 0, it will hold the
                                     ; number of ones in num
12          SUB    r5, r5, r5        ; initialize r5 to 0, it will hold the
                                     ; number of bits examined before all the
                                     ; ones have been shifted out
16          MOVS   r1, r1            ; set the status bits based on the value
                                     ; of num
20          BEQ    store             ; if (Z = 1) or num = 0, stop
24 next     ANDS   r4, r1, r3        ; By anding with r3 which has a value of
                                     ; 1 and setting the status bits, the
                                     ; value of register r4 is either 0 or 1
                                     ; based on the least significant bit of
                                     ; r1 being 0 or 1
28          ADD    r5, r5, #1        ; r5 is counting the number of bits
```

```
                                               ; examined, increment it each time
32          BEQ    shift                       ; if (Z = 1) or r4 = 0 the LSB of r1 was
                                               ; 0 and the ADD r2 is skipped
36          ADD    r2, r2, #1                  ; we only get here if (Z = 0) or r4 = 1
                                               ; add one to the count of the number of
                                               ; ones
40 shift    MOVS   r1, r1, LSR #1              ; Shift the value of num right by one
                                               ; bit. Since this is a logical shift, a
                                               ; 0 comes in at the MSB and the previous
                                               ; LSB is lost to history. Also set
                                               ; status bits
44          BNE    next                        ; if (Z = 0) or num != 0, there are more
                                               ; ones to count
48 store    STR    r2, count                   ; we only get here if (Z = 1) or num = 0
                                               ; and there are no more ones left to
                                               ; count. Store the number of ones in
                                               ; count
52          STR    r5, bits                    ; Store the number of bits examined in
                                               ; bits
56 done     B      done
60 count    SPACE  4                           ; reserve 4 bytes of space for count,
                                               ; no value is given to count
64 bits     SPACE  4                           ; reserve 4 bytes of space for bits,
                                               ; no value is given to bits.
68 num      DCD    2341
72 mask     DCD    1
            END
```

r1	2341	2341	1170	585	292	146	73	36	18	9	4	2	1	0			
r2	0	1	1	2	2	2	3	3	3	4	4	4	5				
r3	1																
r4		1	0	1	0	0	1	0	0	1	0	0	1				
r5	0	1	2	3	4	5	6	7	8	9	10	11	12				

Results of the STR instructions.

Memory Address	Contents
60	5
64	12

Contents of r1	r4
0000 0000 0000 0000 0000 1001 0010 0101	1
0000 0000 0000 0000 0000 0100 1001 0010	0
0000 0000 0000 0000 0000 0010 0100 1001	1
0000 0000 0000 0000 0000 0001 0010 0100	0
0000 0000 0000 0000 0000 0000 1001 0010	0
0000 0000 0000 0000 0000 0000 0100 1001	1
0000 0000 0000 0000 0000 0000 0010 0100	0
0000 0000 0000 0000 0000 0000 0001 0010	0
0000 0000 0000 0000 0000 0000 0000 1001	1
0000 0000 0000 0000 0000 0000 0000 0100	0
0000 0000 0000 0000 0000 0000 0000 0010	0
0000 0000 0000 0000 0000 0000 0000 0001	1
0000 0000 0000 0000 0000 0000 0000 0000	

11. (25 points) Complete the ARM assembly language program below so that it implements the following C++ statements.

```
int neg = 0;
int pos = 0;
int size = 10;
int nums[10] = {5, 3, -1, 2, 4, 37, -100, 13, -5, 0};
for (i = 0; i < size; i++)
   if (nums[i] < 0)
      neg++;
   else
      pos++;
```

```
        AREA   PROB_11, CODE, READONLY
        ENTRY
;       CPE 221 Assembly Example
;
;       This program counts the number of negative and positive values in an
;       array and stores them in neg_count and pos_count, respectively
;
        AREA   COUNT_NEG_POS, CODE, READONLY
        ENTRY
        ADR    r10, nums              ; pointer to first element of array
        LDR    r1, size               ; holds size of array
        LDR    r2, i                  ; i = 0
        LDR    r3, =0                 ; neg = 0
        MOV    r4, #0                 ; pos = 0
loop    CMP    r2, r1                 ; Check to see whether i < size.
        BPL    store                  ; If not, done.
        ADD    r5, r10, r2, LSL #2    ; Add index to base array pointer.
        LDR    r5, [r5]               ; Load nums[i] into r5.
        ADD    r2, r2, #1             ; i++
        CMP    r5, #0                 ; Set status bits
        ADDPL  r4, r4, #1             ; pos++
        ADDMI  r3, r3, #1             ; neg++
        B      loop                   ; Go back to loop.
store   STR    r3, neg
        STR    r4, pos
done    B      done
nums    DCD    5, 3, -1, 2, 4, 37, -100, 13, -5, 0
neg     SPACE  4
pos     SPACE  4
i       DCD    0
size    DCD    10
        END
```