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

This test is closed book, closed notes. You may use a calculator. You should have the ARM reference packet. You must show your work to receive full credit. Before you begin, please make sure that you have all nine pages of the exam.

1. (1 point) Pipelining is a technique for increasing the _**throughput**_ of instruction execution.
2. (1 point) _**SRAM**_ uses cross-coupled transistors to store one bit of memory.
3. (1 point) The _**CS**_ signal enables the operation of a particular memory module.
4. (1 point) The principle of _**temporal**_ locality states that items that have been recently accessed will be accessed again soon.
5. (1 point) (True or False) _**True**_ A fully associative cache has one set.
6. (4 points) In an ARM computer, `r2` contains a value of -4263 in decimal. What is the binary value of `r1` after this instruction is executed?

```
ROR   r1, r2, #7
```

```
16      0    1
16      1    0
16     16   10
16    266    7
16   4263
```

```
 4263 = 0000 0000 0000 0000 0001 0000 1010 0111
-4263 = 1111 1111 1111 1111 1110 1111 0101 1001
Rotating right 7
r1 = 1011 0011 1111 1111 1111 1111 1101 1110  or 0xB3FF FFDE
```

7. (4 points) In an ARM computer, r2 contains a value of 5971 in decimal. What is the binary value of `r2` after this instruction is executed?

```
MOVT   r2, #5971
```

```
16      0    1
16      1    7
16     23    5
16    373    3
16   5971
```

```
5971 = 0000 0000 0000 0000 0001 0111 0101 0011

r2 = 0001 0111 0101 0011 0001 0111 0101 0011   or ox1753 1753
```

8. (2 points) In an ARM computer, r2 contains a value of -4263 in decimal while r3 contains a value of 5971 in decimal. What is the binary value of r1 after this instruction is executed?

```
BIC   r1, r2, r3

r2 = 1111 1111 1111 1111 1110 1111 0101 1001
r3 = 0000 0000 0000 0000 0001 0111 0101 0011
r1 = 1111 1111 1111 1111 1110 1000 0000 1000  or 0xFFFF E808
```

9. (13 points) (a) (9 points) What are the values of the following registers when the program executes "B loop" for the sixth time? Answer in decimal.

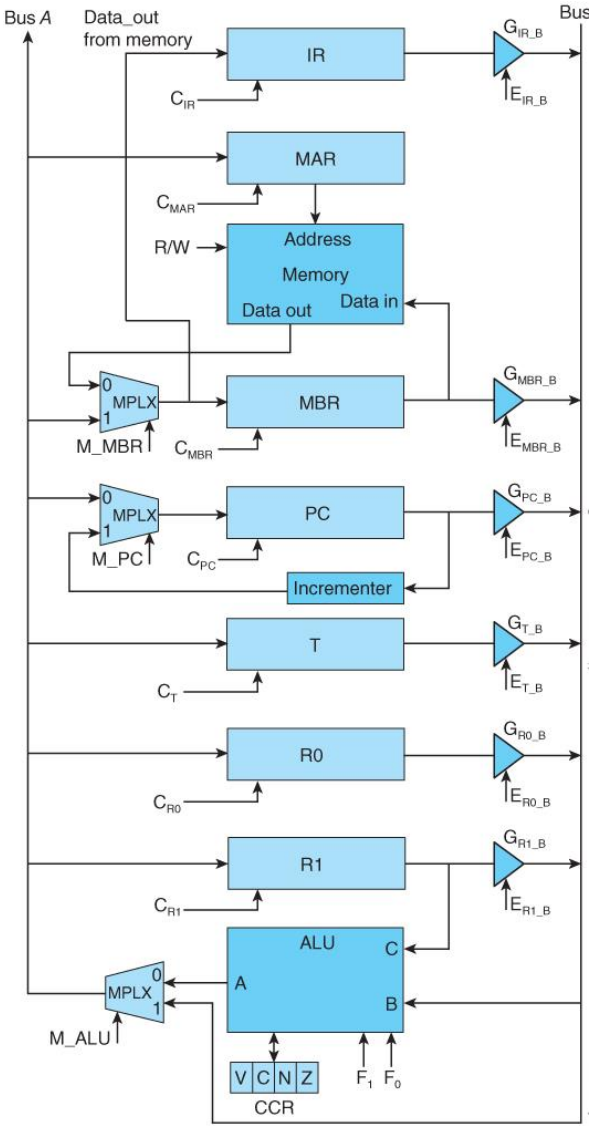
r3: _1_ r4_5_ r5: _37_

(b) (4 points) What values are written by the 56 STR r3, neg and 60 STR r4, pos instructions? Answer in decimal.

56 STR r3, neg _3_ 60 STR r4, pos _7_

```
            AREA   COUNT_NEG_POS, CODE, READONLY
            ENTRY
0           ADR    r10, nums
4           LDR    r1, size
8           LDR    r2, i
12          LDR    r3, =0
16          MOV    r4, #0
20 loop     CMP    r2, r1
24          BPL    store
28          ADD    r5, r10, r2, LSL #2
32          LDR    r5, [r5]
36          ADD    r2, r2, #1
40          CMP    r5, #0
44          ADDPL  r4, r4, #1
48          ADDMI  r3, r3, #1
52          B      loop
56 store    STR    r3, neg
60          STR    r4, pos
64 done     B      done
68 size     DCD    10
72 neg      SPACE  4
76 pos      SPACE  4
80 i        DCD    0
84 nums     DCD    5, 3, -1, 2, 4, 37, -100, 13, -5, 0
            END
```

10. (15 points) For the architecture shown, except that the ALU has 8 possible operations, write the concrete RTL and the sequence of signals and control actions necessary to execute the instruction `XOR R1, R0`, that stores the exclusive-OR of R0 and R1 in R1.
Abstract RTL: R1 ← R0 ⊕ R1



F_2	F_1	F_0	Operation
0	0	0	A = B′
0	0	1	A = C′
0	1	0	A = B OR C
0	1	1	A = B AND C
1	0	0	A = B + C
1	0	1	A = B - C
1	1	0	A = B + 1
1	1	1	A = C + 1

Cycle	Concrete RTL	Signals
1	**T ← R0′**	**E_{R0_B}, F = 000, M_ALU = 0, C_T**
2	**T ← T AND R1**	**E_{T_B}, F = 011, M_ALU = 0, C_T**
3	**R1 ← R1′**	**F = 001, M_ALU = 0, C_{R1}**
4	**R1 ← R0 AND R1**	**E_{R0_B}, F = 011, M_ALU = 0, C_{R1}**
5	**R1 ← R1 OR T**	**E_{T_B}, F = 010, M_ALU = 0, C_{R1}**

11. (10 points) A certain memory system has a 4 GB main memory and a 64 MB cache. Blocks are 8 words and each word is 32 bits. Show the fields in a memory address if the cache is 8-way set associative. This memory system is byte addressable.

A 4 GB memory has 32 bits of address.
64 MB x 8 bits/1 byte x 1 word/32 bits x 1 block/8 words x 1 set/8 blocks = 2^{18} sets
4 bytes/word → 2 bits of byte offset
8 words/block → 3 bits of block offset
2^{18} set → 18 bits of index
Tag = 32 – (2 + 3 + 18) = 32 – 23 = 9 bits

12. (6 points) If you want to build a 2^{31} word, 128-bits-per-word memory and the only parts you have available to you are static RAM chips that contain 2^{18} 8 bit words each. (a) (2 points) How many rows are required? (b) (2 points) How many columns are required? (c) (2 points) How many chips in all?

(a) $2^{31}/2^{18} = 2^{13}$ rows
(b) 128/8 = 16 columns
(c) 2^{13} x 16 = 2^{17} chips

13. (6 points) A RISC processor executes the following code. There are data dependencies. A source operand cannot be used until it has been written.

```
LDR   r2, [r4]
MOV   r3, r5
STR   r6, [r2]
```

Assuming a five-stage pipeline (fetch (IF), operand fetch (OF), execute (E), memory access (M), and register write (W)), how many extra cycles are required to ensure that the correct value of r2 is available for the STR instruction? **Two extra cycles are required as the STR instruction spends three cycles in the OF stage because it has to wait on the LDR to be written in stage 5.**

	1	2	3	4	5	6	7	8	9
LDR r2, [r4]	IF	OF	E	M	W				
MOV r3, r5		IF	OF	E	M	W			
STR r6, [r2]			IF	OF	OF	OF	E	M	W

14. (20 points) Complete the ARM assembly language program below so that it implements the following C++ statements.

```
;
;       This program examines two arrays, element by element and copies
;       the larger number of each pair into a third array. It also
;       writes a 0 into an array called which if the x value was
;       selected in that position or a 1 if the y value was selected.
;
;      const int size = 10;
;       int x[size] = {100, 3, -1, 2, 4, 4, 2, -1, 3, 100};
;       int y[size] = {-53, 247, 95, -7, 481, 91, -33, 1500, 29, -83};
;       int z[size];
;       int which[size];
;       int i;
;       for (i = 0; i < size; i++)
;         if (x[i] > y[i]) {
;           z[i] = x[i];
;           which[i] = 0; }
;         else {
;           z[i] = y[i];
;           which[i] = 1; }

        AREA    PROB_11, CODE, READONLY
        ENTRY
        ADR     r0, x
        ADR     r1, y
        ADR     r2, z
        LDR     r3, size
        LDR     r4, i
        ADR     r5, which
loop    CMP     r4, r3          ; i < size
        BPL     done            ; if i == 10, exit for
        LDR     r6, [r0], #4    ; r6 ← x[i] w/post index
        LDR     r7, [r1], #4    ; r7 ← y[i] w/post index
        CMP     r6, r7          ; x[i] > y[i]
        STRPL   r6, [r2], #4    ; z[i] ← x[i] if x[i] >= y[i]
        MOVPL   r10, #0         ; prepare 0 for which[i]
        STRPL   r10, [r5], #4   ; store 0 in which[i]
        STRMI   r7, [r2], #4    ; z[i] ← y[i] if x[i] < y[i]
        MOVMI   r10, #1         ; prepare 1 for which[i]
        STRMI   r10, [r5], #4   ; store in in which[i]
        ADD     r4, r4, #1      ; i++
        B       loop
done    B       done
x       DCD     100, 3, -1, 2, 4, 4, 2, -1, 3, 100
y       DCD     -53, 247, 95, -7, 481, 91, -33, -1500, 29, -83
z       SPACE   40
which   SPACE   40
i       DCD     0
size    DCD     10
        END
```

15. (15 points) Consider the following ARM program. Trace the stack activity, including all changes to the stack pointer and to the contents of the stack. Clearly indicate the value of the sp.

```
;   int main() {
;     int P = 3;
;     int Q = -1;
;     P = Func1(P);
;     Q = Func1(Q);}
;   int Func1(int x) {
;     if (x > 0) x = Times16(x) + 1;
;     else x = 32*x;
;     return(x);}
;   int Times16(int x) {
;     x = 16*x;
;     return(x);}

            AREA NESTED_SUBROUTINE_STACK, CODE, READWRITE
            ENTRY
0           ADR     r4, P
4           ADR     r5, Q
8           MOV     sp, #0
12          MOV     fp, #0x0000C000
16          LDR     r0, [r4]
20          BL      Func1
24          STR     r1, [r4]
28          LDR     r0, [r5]
32          BL      Func1
36          STR     r1, [r5]
40  done    B       done
44  Func1   PUSH    {fp}
48          CMP     r0, #0
52          PUSHGT  {lr}
56          PUSHGT  {r0}
60          BLGT    Times16
64          POPGT   {r1}
68          POPGT   {lr}
72          ADDGT   r1, r1, #1
76          MOVLE   r1, r0, LSL #5
80          POP     {fp}
84          MOV     pc, lr
88  Times16 POP     {r7}
92          MOV     r7, r7, LSL #4
96          PUSH    {r7}
100         MOV     pc, lr
            AREA  NESTED_SUBROUTINE_STACK, DATA, READWRITE
104 P       DCD   3
108 Q       DCD   -1
            END
```

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	0x0000C000

Instruction: _44 PUSH {fp}_

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: _52 PUSHGT {lr}_

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	3
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: _56 PUSHGT {r0}_

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	3
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: _88 POP {r7}_

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	48
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: _96 PUSH {r7}_

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	48
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: _64 POPGT {r1}_

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	48
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: __68 POPGT {lr}__

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	48
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: _80 POP {fp},sp = 0

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	48
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: __44 PUSH {fp}__

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	48
FFFF FFF8	24
FFFF FFFC	0x0000C000

Instruction: _80 POP {fp}, sp=0

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction: _____

Address	Value
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction: _____

Shading indicates value of sp, except where sp = 0 which is noted.