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

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 ten pages of the exam.

1. (1 point) A 1 address instruction has a register, called the _**accumulator**_ to hold one operand and the result..
2. (1 point) A _**tristate gate**_ is used to control what is allowed to drive a bus.
3. (1 point) _**DRAM**_ requires refreshing.
4. (1 point) The principle of _**spatial**_ locality states that items close to items recently accessed will be accessed 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, #13

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

  4263 = 0000 0000 0000 0000 0001 0000 1010 0111
 -4263 = 1111 1111 1111 1111 1110 1111 0101 1001
    r1 = 0111 1010 1100 1111 1111 1111 1111 1111 = 0x7ACF FFFF
```

7. (4 points) What is the binary value of r2 after this instruction is executed?

```
MVN   r2, #6284

            0
16          1     1
16         24     8
16        392     8
16       6284    12

  6284 = 0000 0000 0000 0000 0001 1000 1000 1100
    r2 = 1111 1111 1111 1111 1110 0111 0111 0011 = 0xFFFF E773
```

8. (3 points) In an ARM computer, r2 contains a value of -4263 in decimal while r3 contains a value of 6284 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 1000 1000 1100
r3′ = 1111 1111 1111 1111 1110 0111 0111 0011
r1 = 1111 1111 1111 1111 1110 0111 0101 0001 = 0xFFFF E751

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: 137

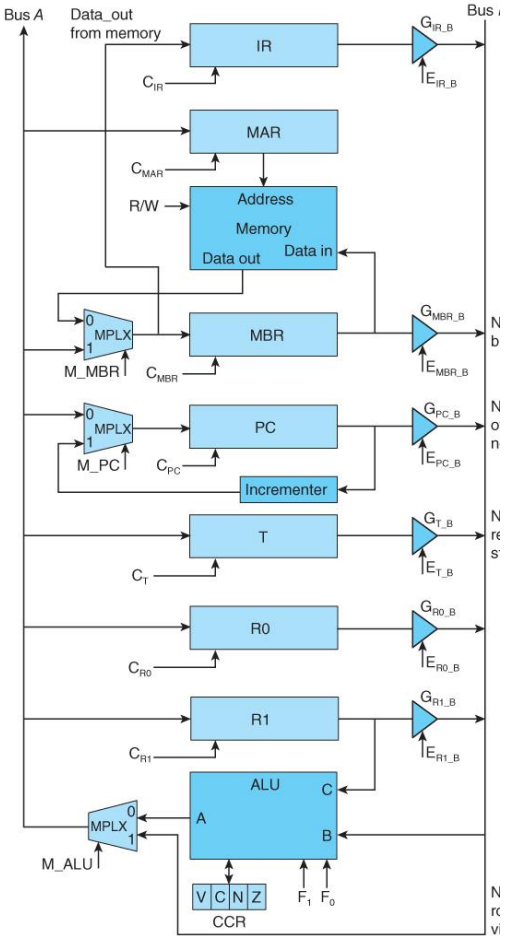
(b) (4 points) What values are written by the `56 STR r3, neg` and `60 STR r4, pos` instructions? Answer in decimal.

`56 STR r3, neg` **4** `60 STR r4, pos` **6**

```
            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, 22, 2, 4, 137, -100, -13, -5, 0
            END
```

10. (15 points) For the architecture shown, write the concrete RTL and the sequence of signals and control actions necessary to execute the instruction `SSUB P, R1, R0`, that stores `R0 – R1` in the memory location pointed to by P. Assume that `P` is stored in IR. Use as few cycles as possible.

Abstract RTL: `M[P] ← R0 – R1`



F_1	F_0	Operation
0	0	A = B′
0	1	A = B
1	0	A = B + C
1	1	A = B + 1

Cycle	Concrete RTL	Signals
1	**MAR ← IR**	**E_{IR_B}, M_ALU = 1, C_{MAR}**
2	**T ← R1**	**E_{R1_B}, M_ALU = 1, C_T**
3	**R1 ← R1′**	**E_{R1_B}, F = 00, M_ALU = 0, C_{R1}**
4	**R1 ← R1 + 1**	**E_{R1_B}, F = 11, MALU = 0, C_{R1}**
5	**MBR ← R0 + R1**	**E_{R0_B}, F = 10, M_ALU = 0, M_MBR = 1, C_{MBR}**
6	**M[MAR] ← MBR, R1 ← T**	**WRITE, E_{T_B}, M_ALU = 1, C_{R1}**

11. (10 points) A certain memory system has a 4 GB main memory and a 64 MB cache. Blocks are 16 words and each word is 32 bits. Show the fields in a memory address if the cache is 4-way set associative. This memory system is byte addressable.

Total Address: It takes 32 bits to address 4 GB.
Block offset: It takes 4 bits to address the 16 words in a block.
Byte offset: It takes 2 bits to address the 4 bytes in a word.
Index: 64 MB x 1 word/4 bytes x 1 block/16 words x 1 set/4 blocks = $2^{(26-2-4-2)} = 2^{18}$
Tag: Total Address – Index – Block Offset – Byte Offset = 32 – 18 – 4 – 2 = 8

12. (6 points) If you want to build a 2^{48} word, 256-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? **$2^{48}/2^{18} = 2^{30}$** (b) (2 points) How many columns are required? **256/8 = 32** (c) (2 points) How many chips in all? **2^{30} x 32 = 2^{35}**

13. (20 points) Complete the ARM assembly language program below so that it implements the following C++ statements and stores the correct values in `pal` and `loc`.

```
;       This program determines whether the elements in an array
;       are the same forwards as backwards. If they are not, the first
;       location at which a difference is found is stored in loc.
;       const int size = 10;
;       int x[size] = {100, 3, -1, 2, 4, 4, 2, -1, 3, 100};
;       int pal, i, first;
;       pal = 1;
;       first = 1;
;       loc = 10;
;       for (i = 0; i < size; i++)
;         if (x[i] != x[size - i - 1]) {
;            pal = 0;
;            if (first == 1) {
;              loc = i;
;              first = 0;
;            }
;         }

        AREA    PROB_13, CODE, READONLY
        ENTRY
        LDR     r3, size
        LDR     r4, i
        MOV     r0, #1
        STR     r0, pal
        STR     r0, first
        MOV     r0, #10
        STR     r0, loc
        ADR     r10, x
loop    CMP     r4, r3
        BGE     done
        SUB     r2, r3, r4
        SUB     r2, r2, #1
        LDR     r5, [r10, r4, LSL #2]
        LDR     r6, [r10, r2, LSL #2]
        CMP     r5, r6
        BEQ     next
        MOV     r0, #0
        STR     r0, pal
        LDR     r7, first
        CMP     r7, #1
        BNE     next
        STR     r4, loc
        STR     r0, first
next    ADD     r4, r4, #1
        B       loop
done    B       done
x       DCD     100, 3, -1, 2, 4, 4, 2, -1, 3, 100
loc     SPACE   4
pal     SPACE   4
i       DCD     0
first   DCD     1
size    DCD     10
        END
```

14. (20 points) Consider the following ARM program. Trace the stack activity, including all changes to the stack pointer, the frame pointer and to the contents of the stack. Clearly indicate the value of the sp and the fp. Include any instruction that changes the sp, the fp or the contents of the stack.

```
int main ()
{
  int base5power1;
       base5power1 = Power(5, 1);
}
int Power(int number, // Base number
          int n)      // Power to raise base to
  int result = 1;     // Holds intermediate powers of x
  while (n > 0)
  {
    result = result * number;
    n--;
  }
  return result;
}
```

```
        AREA POWER_STACK, CODE, READONLY
        ENTRY
main    mov     sp, #0                  ; (0)
        sub     sp, sp, #4              ; (4)
        add     fp, sp, #0              ; (8)
        movs    r1, #1                  ; (12)
        movs    r0, #5                  ; (16)
        bl      Power                   ; (20)
        str     r0, [fp]                ; (24)
        adds    fp, fp, #4              ; (28)
        mov     sp, fp                  ; (32)
done    B       done                    ; (36)
Power   push    {fp}                    ; (40)
        sub     sp, sp, #20             ; (44)
        add     fp, sp, #0              ; (48)
        str     r0, [fp, #4]            ; (52)
        str     r1, [fp]                ; (56)
        movs    r3, #1                  ; (60)
        str     r3, [fp, #12]           ; (64)
        b       L4                      ; (68)
L5      ldr     r3, [fp, #12]           ; (72)
        ldr     r2, [fp, #4]            ; (76)
        mul     r3, r2, r3              ; (80)
        str     r3, [fp, #12]           ; (84)
        ldr     r3, [fp]                ; (88)
        subs    r3, r3, #1              ; (92)
        str     r3, [fp]                ; (96)
L4      ldr     r3,  [fp]               ; (100)
        cmp     r3, #0                  ; (104)
        bgt     L5                      ; (108)
        ldr     r3, [fp, #12]           ; (112)
        mov     r0, r3                  ; (116)
        adds    fp, fp, #20             ; (120)
        mov     sp, fp                  ; (124)
        ldr     fp, [sp], #4            ; (128)
        bx      lr                      ; (132)
        END
```

Address	Value
FFFF FFE4	
FFFF FFE8	
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

`mov sp, #0 (0) sp = 0`

Address	Value
FFFF FFE4	
FFFF FFE8	
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction

`sub sp, sp, #4 (4)`

Address	Value
FFFF FFE4	
FFFF FFE8	
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

`add fp, sp, #0 (8)`

Address	Value
FFFF FFE4	
FFFF FFE8	
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`push {fp} (40)`

Address	Value
FFFF FFE4	
FFFF FFE8	
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`sub sp, sp, #20 (44)`

Address	Value
FFFF FFE4	
FFFF FFE8	
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`add fp, sp, #0 (48)`

Address	Value
FFFF FFE4	
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`str r0, [fp, #4] (52)`

Address	Value
FFFF FFE4	1
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`str r1, [fp] (56)`

Address	Value
FFFF FFE4	1
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`str r3, [fp, #12] (64)`

Address	Value
FFFF FFE4	1
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	5
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`str r3, [fp, #12] (84)`

Address	Value
FFFF FFE4	0
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	5
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`str r3, [fp] (96)`

Address	Value
FFFF FFE4	0
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	5
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`adds fp, fp, #20 (120)`

Address	Value
FFFF FFE4	0
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	5
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`_mov sp, fp (124)_`

Address	Value
FFFF FFE4	0
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	5
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	

Instruction:

`_ldr fp, [sp], #4 (128)_`

Address	Value
FFFF FFE4	0
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	5
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	5

Instruction:

`_str r0, [fp] (24)_`

Address	Value
FFFF FFE4	0
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	5
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	5

Instruction:

`_adds fp, fp, #4 (28), fp = 0_`

Address	Value
FFFF FFE4	0
FFFF FFE8	5
FFFF FFEC	
FFFF FFF0	5
FFFF FFF4	
FFFF FFF8	FFFF FFFC
FFFF FFFC	5

Instruction:

`_mov sp, fp (32), fp = 0, 0, sp = 0_`

Address	Value
FFFF FFE4	
FFFF FFE8	
FFFF FFEC	
FFFF FFF0	
FFFF FFF4	
FFFF FFF8	
FFFF FFFC	

Instruction:

_____