**The University of Alabama in Huntsville**
**Electrical and Computer Engineering Department**
**CPE 221 01**
**Final Exam Solution**
**Fall 2019**

**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. (2 points) List the RTL for the PUSH {r9} instruction: _**sp ← sp – 4, M[sp] ← r9**_.
2. (1 point) _**Forwarding**_ is a method for resolving a data hazard by retrieving the missing data from internal buffers rather than waiting for it to arrive from registers or memory.
3. (1 point) The principle of _**temporal**_ locality states that items recently accessed will be accessed again soon.
4. (1 point) (True/False) _**False**_ Caches with 8-way set associativity have 8 sets.
5. (4 points) In an ARM computer, r2 contains a value of --145 in decimal. What is the binary value of r1 after this instruction is executed?

```
LSL r1, r2, #5
-r2 = 145 = 0000 0000 0000 0000 0000 0000 1001 0001
r2 = -145 = 1111 1111 1111 1111 1111 1111 0110 1111
r1 = -145 shifted left 5 places = 1111 1111 1111 1111 1110 1101 1110
0000
```

6. (4 points) What is the binary value of r2 after this instruction is executed if r2 is initially -145 decimal?

```
ORR   r2, r2, #6284
r2 = -145    = 1111 1111 1111 1111 1111 1111 0110 1111
6284         = 0000 0000 0000 0000 0001 1000 1000 1100
-145 OR 6284 = 1111 1111 1111 1111 1111 1111 1110 1111

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

7. (3 points) In an ARM computer, r2 contains a value of -0xFB97 CE1A while r3 contains a value of 0x5932 CD07 What is the binary value of r2 after this instruction is executed?

```
REV   r2, r3
r2       = 1111 1011 1001 0111 1100 1110 0001 1010
r3       = 0101 1001 0011 0010 1100 1101 0000 0111
r2       = 0000 0111 1100 1101 0011 0010 0101 1001
```

8. (13 points) (a) (9 points) What are the values of the following registers when the program executes "`B loopa`" for the sixth time? Answer in decimal.

R8: _4_ r4_6_ r9: _64_

(b) (4 points) What values are written by the `32 STR r9, [r2, r4, LSL #2]` instruction the third and ninth time it is executed? Answer in decimal.

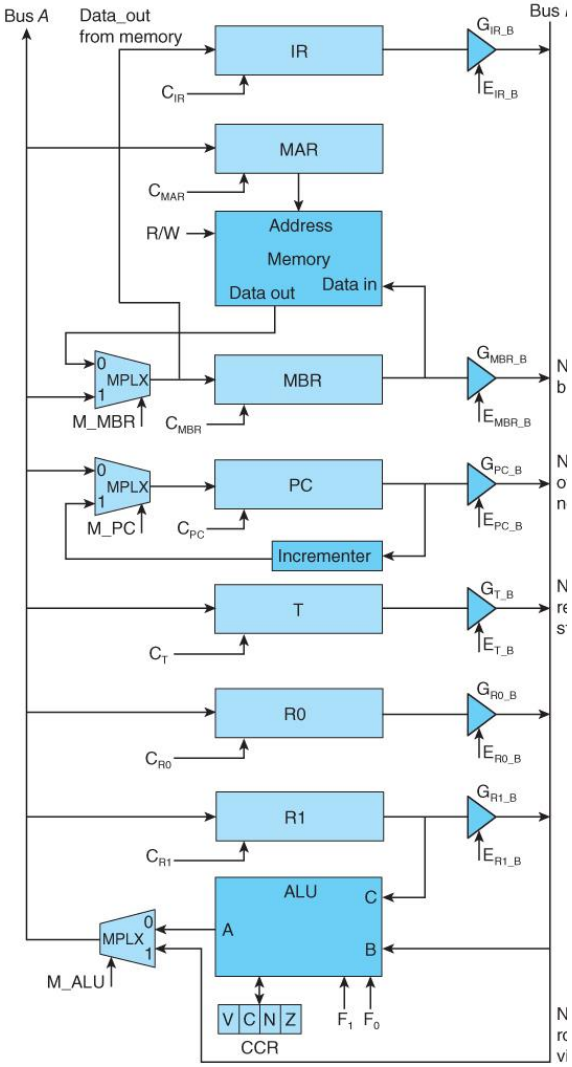
`STR r9, [r2, r4, LSL #2] ;(32)` (third time) _-1_

`STR r9, [r2, r4, LSL #2] ;(32)` (ninth time) _27_

```
        AREA   PROB_8, CODE, READ
        ENTRY
        ADR    r0, x                                  ;(0)
        ADR    r2, z                                  ;(4)
        LDR    r3, size                               ;(8)
        LDR    r4, i                                 ;(12)
loopa   CMP    r4, r3                                ;(16)
        BGE    done                                  ;(20)
        LDR    r8, [r0, r4, LSL #2]                  ;(24)
        BL     sub                                   ;(28)
        STR    r9, [r2, r4, LSL #2]                  ;(32)
        ADD    r4, r4, #1                            ;(36)
        B      loopa                                 ;(40)
done    B      done                                  ;(44)
sub     MOV    r9, #1                                ;(48)
        MOV    r1, #0                                ;(52)
loopb   CMP    r1, #3                                ;(56)
        BGE    back                                  ;(60)
        MUL    r9, r8, r9                            ;(64)
        ADD    r1, r1, #1                            ;(68)
        B      loopb                                 ;(72)
back    MOV    pc, lr                               ;(76)
done    B      done                                 ;(80)
x       DCD    100, 3, -1, 2, 4, 4, 2, -1, 3, 100    ;(84)
z       SPACE   40                                  ;(124)
i       DCD     0                                   ;(128)
size    DCD     10                                  ;(132)
        END
```

9. (15 points) For the architecture shown, write the concrete RTL and the sequence of signals and control actions necessary to execute the instruction `ADD [P, R1], R0` which is described by the abstract RTL given. Assume that `P` is stored in the instruction register (IR). Use as few cycles as possible.

Abstract RTL: `R0 ← M[P + R1] + R0`




| $F_1$ | $F_0$ | 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 + R1** | **$E_{IR\_B}$, F = 10, M_ALU = 0, $C_{MAR}$** |
| 2 | **MBR ← M[MAR], T ← R1** | **$E_{R1\_B}$, M_ALU = 1, $C_T$, R = 1, $C_{MBR}$, M_MBR = 0** |
| 3 | **R1 ← MBR** | **$E_{MBR\_B}$, M_ALU = 1, $C_{R1}$** |
| 4 | **R0 ← R1 + R0** | **$E_{R0\_B}$, F = 10, M_ALU = 0, $C_{R0}$** |
| 5 | **R1 ← T** | **$E_{T\_B}$, M_ALU = 1, $C_{R1}$** |

10. (10 points) A certain memory system has a 4 GB main memory and a 32 MB cache. Blocks are 4 words and each word is 64 bits. Show the fields in a memory address if the cache is 2-way set associative. This memory system is byte addressable.

**32 MB x 1 word/64 bits x 8 bits/1 byte x 1 block/4 words x 1 set/2 blocks = $2^{(25-6+3-2-1)} = 2^{19}$ sets**
**Byte offset = 3, since 8 or 23 bytes in a word**
**Block offset = 2, since 4 or 22 words in a block**
**Address = 32 bits, since 232 = 4 GB**
**Tag = 32 – 19 – 3 – 2 = 8 bits**

| Tag (8 bits) | Index (19 bits) | Block Offset (2 bits) | Byte Offset (3 bits) |
|---|---|---|---|

11. (6 points) A RISC processor executes the following code. There are data dependencies. A source operand cannot be used until it has been written. The register file cannot do a write to and a read from the same register in a single clock cycle.

```
LDR   r2, [r4]
MOV   r6, r2
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 values of are available for the `STR` instruction? **Without dependences, pipeline time is m + n – 1, where m is the number of instructions and n is the number of pipeline stages, in this case 3 + 5 -1 = 7 cycles. Since the instructions complete in 13 cycles, that is 6 extra cycles.**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `LDR r2, [r4]` | IF | OF | E | M | W | | | | | | | | |
| `MOV r6, r2` | | IF | OF | OF | OF | OF | E | M | W | | | | |
| `STR r6, [r2]` | | | IF | IF | IF | IF | OF | OF | OF | OF | E | M | W |

12. (20 points) Complete the ARM assembly language program below so that it implements the following C++ statements and stores the correct values in `num_even` and `num_odd`.

```
;       const int size = 10
;       int x[size] = {1, 2, 3, 5, 7, 11, 13, 17, 19, 23};
;       int y[size] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};
;       int z[size];
;       int num_even, num_odd = 0;
;       int i;
;       for (i = 0; i < size; i++)
;       {
;         if ((x[i] % 2) == 0)
;         {
;           num_even++;
;           z[i] = x[i];
;         }
;         else
;         {
;           num_odd++;
;           z[i] = y[i];
;         }
;       }
```

```
            AREA    PROB_12, CODE, READONLY
            ENTRY
            LDR     r3, size
            LDR     r4, i
            ADR     r10, x
            ADR     r11, y
            ADR     r12, z
            MOV     r0, #0
            MOV     r1, #0
loop        CMP     r4, r3
            BGE     save
            LDR     r5, [r10, r4, #LSL 2]
            AND     r2, r5, #1
            CMP     r2, #0
            ADDEQ   r0, r0, #1
            LDRNE   r5, [r11, r4, LSL #2]
            ADDNE   r1, r1, #1
            STR     r5, [r12, r4, LSL #2]
            ADD     r4, r4 #1
            B       loop
save        STR     r0, num_even
            STR     r1, num_odd
done        B       done
x           DCD     1, 2, 3, 5, 7, 11, 13, 17, 19, 23
y           DCD     2, 4, 6, 8, 10, 12, 14, 16, 18, 20
z           SPACE   40
num_even    SPACE   4
num_odd     SPACE   4
i           DCD     0
size        DCD     10
            END
```

13. (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.

```
//This program calculates the cube of each of the elements of an
//array using a subroutine and stores them in another array.

int cube(int);
int main()
{
  int x = 4;
  int z;
  z = cube(x);
  return(0);
}
int cube(int val)
{
  int i, result;
  result = 1;
  for (i = 0; i < 3; i++)
    result = result * val;
  return result;
}
```

```
        AREA PROB14, CODE, READONLY
        ENTRY
main    mov   sp, #0              ;(0)
        sub   sp, sp, #4          ;(4)
        add   fp, sp, #0          ;(8)
        mov   r0, #4              ;(12)
        bl    cube                ;(16)
        str   r0, [fp, #-12]      ;(20)
        mov   r3, #0              ;(24)
        mov   r0, r3              ;(28)
        add   sp, fp, #4          ;(32)
done    b     done                ;(36)
cube    str   fp, [sp, #-4]!      ;(40)
        add   fp, sp, #0          ;(44)
        sub   sp, sp, #20         ;(48)
        str   r0, [fp, #-16]      ;(52)
        mov   r3, #1              ;(56)
        str   r3, [fp, #-12]      ;(60)
        mov   r3, #0              ;(64)
        str   r3, [fp, #-8]       ;(68)
        b     L4                  ;(72)
L5      ldr   r3, [fp, #-12]      ;(76)
        ldr   r2, [fp, #-16]      ;(80)
        mul   r3, r2, r3          ;(84)
        str   r3, [fp, #-12]      ;(88)
        ldr   r3, [fp, #-8]       ;(92)
        add   r3, r3, #1          ;(96)
        str   r3, [fp, #-8]       ;(100)
L4      ldr   r3, [fp, #-8]       ;(104)
        cmp   r3, #2              ;(108)
        ble   L5                  ;(112)
        ldr   r3, [fp, #-12]      ;(116)
        mov   r0, r3              ;(120)
        add   sp, fp, #0          ;(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, sp=0 (0)
```

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | |
| FFFF FFEC | |
| FFFF FFF0 | |
| FFFF FFF4 | |
| FFFF FFF8 | **FFFF FFFC** |
| FFFF FFFC | |

Instruction:

```
str  fp, [sp, #-4]!
(40)
```

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | **4** |
| FFFF FFEC | |
| FFFF FFF0 | |
| FFFF FFF4 | |
| FFFF FFF8 | FFFF FFFC |
| FFFF FFFC | |

Instruction:

```
str  r0, [fp, #-16](52)
```

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | **4** |
| FFFF FFEC | **4** |
| FFFF FFF0 | **0** |
| FFFF FFF4 | |
| FFFF FFF8 | **FFFF FFFC** |
| FFFF FFFC | |

Instruction:

```
str  r3, [fp, #-12](88)
```

| 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** |
| FFFF FFFC | |

Instruction:

```
add fp, sp, #0 (44)
```

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | **4** |
| FFFF FFEC | **1** |
| FFFF FFF0 | |
| FFFF FFF4 | |
| FFFF FFF8 | **FFFF FFFC** |
| FFFF FFFC | |

Instruction:

```
str  r3, [fp, #-12](60)
```

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | **4** |
| FFFF FFEC | **4** |
| FFFF FFF0 | **1** |
| FFFF FFF4 | |
| FFFF FFF8 | **FFFF FFFC** |
| FFFF FFFC | |

Instruction:

```
str  r3, [fp, #-8](100)
```

| 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:

```
sub  sp, sp, #20 (48)
```

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | **4** |
| FFFF FFEC | **1** |
| FFFF FFF0 | **0** |
| FFFF FFF4 | |
| FFFF FFF8 | **FFFF FFFC** |
| FFFF FFFC | |

Instruction:

```
str  r3, [fp, #-8](68)
```

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | **4** |
| FFFF FFEC | **16** |
| FFFF FFF0 | **1** |
| FFFF FFF4 | |
| FFFF FFF8 | **FFFF FFFC** |
| FFFF FFFC | |

Instruction:

```
str  r3, [fp, #-12](88)
```

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | 4 |
| FFFF FFEC | 16 |
| FFFF FFF0 | 2 |
| FFFF FFF4 | |
| FFFF FFF8 | FFFF FFFC |
| FFFF FFFC | |

Instruction:
_str r3, [fp, #-8](100)

0

| 0Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | 4 |
| FFFF FFEC | 64 |
| FFFF FFF0 | 3 |
| FFFF FFF4 | |
| FFFF FFF8 | FFFF FFFC |
| FFFF FFFC | |

Instruction:
__add sp, fp, #0_(124)_

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | 4 |
| FFFF FFEC | 64 |
| FFFF FFF0 | 64 |
| FFFF FFF4 | |
| FFFF FFF8 | FFFF FFFC |
| FFFF FFFC | |

Instruction:
_add sp, fp, #4,sp=0(32)

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

Instruction:
_____

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | 4 |
| FFFF FFEC | 64 |
| FFFF FFF0 | 2 |
| FFFF FFF4 | |
| FFFF FFF8 | FFFF FFFC |
| FFFF FFFC | |

Instruction:
_str r3, [fp, #-12](88)

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | 4 |
| FFFF FFEC | 64 |
| FFFF FFF0 | 3 |
| FFFF FFF4 | |
| FFFF FFF8 | FFFF FFFC |
| FFFF FFFC | |

Instruction:
_ldr fp, [sp], #4 (128)

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | |
| FFFF FFEC | sp |
| FFFF FFF0 | fp |
| FFFF FFF4 | sp, fp |
| FFFF FFF8 | |
| FFFF FFFC | |

Instruction:

___Key to colors_____

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

Instruction:
_____

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | 4 |
| FFFF FFEC | 64 |
| FFFF FFF0 | 3 |
| FFFF FFF4 | |
| FFFF FFF8 | FFFF FFFC |
| FFFF FFFC | |

Instruction:
_str r3, [fp, #-8](100

| Address | Value |
|---|---|
| FFFF FFE4 | |
| FFFF FFE8 | 4 |
| FFFF FFEC | 64 |
| FFFF FFF0 | 64 |
| FFFF FFF4 | |
| FFFF FFF8 | FFFF FFFC |
| FFFF FFFC | |

Instruction:
_str r0, [fp, #-12](20)

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

Instruction:
_____

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

Instruction:
_____