name:	Vince Rothenberg
course:	csci 10
assignment:	homework 12
prepared:	Tue, Apr 28, 2020 // 9:30 am

1. Show the signature of a MUL instruction, and describe the operation of the MUL and MULS instructions.

```
MUL{S}{<c>}  <Rd>, <Rn>{, <Rm>}  // {optional}

Examples:

// r0 = r1 * r2
MUL r0, r1, r2

// r0 *= r1
MUL r0, r1

MULS r0, r1, r2

CMP r0, r1
it eq
MULEQ r0, r2
```

Multiplies two 32-bit values and produces a 32-bit result. The values in the two source operand registers are multiplied and the low-order 32 bits of the result are stored in <Rd>. Higher order bits are lost. If <Rm> is not present, <Rd> is used as the second source operand. The as assembler prefers that <Rm> be used. The 32-bit result does not depend on whether the source operands are considered signed or not.

MULS: If â€˜Sâ€™ is present the N and Z condition flags are updated according to the result, but the other condition flags are not changed. If absent, the condition flags are not changed.

2. Show the signature of a SMULL instruction, and describe the operation of the SMULL and SMULLS instructions.

SMULL{S}{<c>} <RdLo>, <RdHi>, <Rn>, <Rm> // 2 destination and 2 source registers, not optional

Multiplies two signed 32-bit values and produces a 64-bit result in two registers. <RdHi> specifies the high-order 32-bit destination register, <RdLo> the low-order 32-bit destination register, and <Rm> and <Rn> are the source registers. The values in the two source operand registers are multiplied, the high-order 32 bits of the result are stored in <RdHi>, and the low-order 32 bits of the result are stored in <RdLo>. <Rm> and <Rn> are treated as holding signed values.

SMULLS: If â€˜Sâ€™ is present the N and Z condition flags are updated according to the result, but the other condition flags are not changed. If absent, the condition flags are not changed.

3. Show the signature of a UMULL instruction, and describe the operation of the UMULL and UMULLS instructions.

UMULL{S}{<c>} <RdLo>, <RdHi>, <Rn>, <Rm>

Multiplies two unsigned 32-bit values and produces a 64-bit result. The values in the two source operand registers are multiplied, the high-order 32 bits of the result are stored in <RdHi>, and the low-order 32 bits of the result are stored in <RdLo>. <Rm> and <Rn> are treated as holding unsigned values.

UMULLS: If â€˜Sâ€™ is present the N and Z condition flags are updated according to the result, but the other condition flags are not changed. If absent, the condition flags are not changed.

4. When is it "safe" to use MUL, versus SMULL or UMULL?

If you are certain that the result of your multiplication will be within the range âˆ’2,147,483,648 â‰¤ x â‰¤ +2,147,483,647 for signed integers or x â‰¤ 4,294,967,295 for unsigned, then you can use the MUL instruction. Otherwise, you must use the SMULL or UMULL instruction, respectively. Notice that the carry and overflow condition flags are not affected by the multiplication instructions.

5. Show the signature of a SDIV instruction, and describe the operation of the SDIV instruction. Does this instruction set condition flags?

```
SDIV{<c>}  <Rd>, <Rn>, <Rm> // Registers not optional

// Rd = Rn / Rm
```

Divides a signed 32-bit value into another signed 32-bit value, producing a 32-bit signed result. The value in <Rn> is divided by the value in <Rm> and the result is stored in <Rd>. All values are treated as signed values. The remainder is lost.

The condition flags are not changed.

6. Show the signature of a UDIV instruction, and describe the operation of the UDIV instruction. Does this instruction set condition flags?

```
UDIV{<c>}  <Rd>, <Rn>, <Rm>
```

Divides an unsigned 32-bit value into another unsigned 32-bit value, producing a 32-bit unsigned result. The value in <Rn> is divided by the value in <Rm> and the result is stored in <Rd>. All values are treated as unsigned values. The remainder is lost.

The condition flags are not changed.

7. Do SDIV and UDIV compute remainders?

No, since the divide instructions in the ARM ignore the remainder, we will need to compute it on our own in order to use . The sequence of instructions:

```
udiv    r0, r6, r7      @ no, div to get quotient
mul     r1, r0, r7      @ need for computing remainder
sub     r2, r6, r1      @ the mod (remainder)
```

permalinkuses the udiv instruction to compute the quotient. The quotient is then multiplied by the divisor. Subtracting this result from the original dividend yields the remainder. shows how this can be done.

8. Show the sequence of instructions (not using MLS) needed to compute the remainder when r1 is divided by r2. Store the result in r4. The values are SIGNED.

```
// dozens = totalDonuts / donutsPerDozen
// leftover = totalDonuts - (dozens * donutsPerDozen)
// dozens: r0, total: r1, perDozen: r2, leftover: r3, dozens*perDozen: r4

// I have 27 donuts -- many dozen, and how many left over
// dozens = 27 / 12 = 2
// leftover = 27 - (2 * 12) = 3

SDIV      r0, r1, r2
SMULL r4, r0, r2
SUB       r4, r1, r4

// What is r4? leftover is remainder of dividing r1 by r2
r4 = 3

This is how mod is performed.
```

9. Show the sequence of instructions (not using MLS) needed to compute the remainder when r1 is divided by r2. Store the result in r4. The values are UNSIGNED.

```
UDIV      r0, r1, r2
UMULL r4, r0, r2
SUB       r4, r1, r4
```

10. Show the sequence of instructions (using MLS) needed to compute the remainder when r1 is divided by r2. Store the result in r4. The values are UNSIGNED.

// MLS{<c>} <Rd>, <Rn>, <Rm>, <Ra>
// The values in <Rm> and <Rn> are multiplied, the result is subtracted from the value in <Ra>, and the result is stored in <Rd>. Only the low-order 32 bits are retained.
// leftover = totalDonuts - (dozens * donutsPerDozen)
// Rd = Ra - (Rn * Rm)
 	
UDIV r0, r1, r2
MLS r4, r0, r2, r1