# CAO HW3

Remy Jaspers - 4499336

March 1, 2017

## 1a

The steps for multiplying -7 and 3 are found by closely following the algorithm as stated in $3.3, p. 187. First 7 is negated using two's complement. One of both of the signs is negative, hence the result must be negative. After using two's complement again on the end result after 4 iterations, the value 1110 1011 (-21) is found.

| Iteration | Description | MR | MD | Product |
|---|---|---|---|---|
| 0 | Initial values | 0011 | 1111 1001 | 0000 0000 |
| | 1. Convert MR and MD to positive using 2s complement | 0011 | 0000 0111 | 0000 0000 |
| 1 | 1. Prod = Prod + MD | 0011 | 0000 0111 | 0000 0111 |
| | 2. SLL MD | 0011 | 0000 1110 | 0000 0111 |
| | 3. SRL MR | 0001 | 0000 1110 | 0000 0111 |
| 2 | 1. Prod = Prod + MD | 0001 | 0000 1110 | 0001 0101 |
| | 2. SLL MD | 0001 | 0001 1100 | 0001 0101 |
| | 3. SRL MR | 0000 | 0001 1100 | 0001 0101 |
| 3 | 1. NOP | 0000 | 0001 1100 | 0001 0101 |
| | 2. SLL MD | 0000 | 0011 1000 | 0001 0101 |
| | 3. SRL MR | 0000 | 0011 1000 | 0001 0101 |
| 4 | 1. NOP | 0000 | 0001 1100 | 0001 0101 |
| | 2. SLL MD | 0000 | 0111 0000 | 0001 0101 |
| | 3. SRL MR | 0000 | 0111 0000 | 0001 0101 |
| After | Convert product to negative using 2s complement | 0000 | 0111 0000 | 1110 1011 |

## 1b

This is possible because multiplication is commutative. In all cases we can begin by converting the multiplicand and multiplier to positive values, but we have to remember if either one of them was a negative value. If this is the case, the product will also be negative. We can then use two's complement on the end result to obtain a negative product.

## 1c

The multiplier is 8 bits, hence we need as many iterations as there are bits in the multiplier, which is 8. We need an addition and two shifts in each iteration, and the shifts of the multiplicand and multiplier are done simultaneously. We need (8 * 2) * 3u = 48u time units to perform this calculation.

## 1d

By using a parallel tree of ALU's we can reduce the number of iterations from 8 to $log_2(8) = 3$. The time needed is then (3 * 3u) for the adders, and additionally (3 * 3u) for the shifts = 27u . However, we need $\sum_{n=1}^{b} n$ adders, where b is the number of bits in the multiplier. In the case of 8 bits, we need 36 adders.

## 2a

C463000 represents **1100 0100 0110 0011 0000 0000 0000 0000** in binary. This is from left to right: $(2^{31} + 2^{30} + 2^{26} + 2^{22} + 2^{21} + 2^{17} + 2^{16}) = 3294822400_{10}$

## 2b

First the 2s complement is found by inverting all bits and adding 1 to the result.
**1100 0100 0110 0011 0000 0000 0000 0000**
**0011 1011 1001 1100 1111 1111 1111 1111**
**0011 1011 1001 1101 0000 0000 0000 0000**
Then we calculate the value to be $-1000144896_{10}$, by remembering that this is a negative value, because the sign bit in the uncomplemented value is 1.

## 2c

Looking at the first 6 bits, we find the opcode to be **110001** or 49 in decimal. This corresponds to the **lwc1** or Load Floating Point Single. We extract all bits in I-format:

110001 | 00011 | 00011 | 0000000000000000
49 | 3 | 3 | 0
lwc1 $v1, 0($v1)

Which loads a single precision floating point number into the $v1 result register, with offset 0.