Assignment 2
COEN 317

Adrian PATTERSON
40048841

December 7, 2020

Date Due:	October 9, 2020
Instructor:	Professor Gomar
Section:	U

I certify that this submission is my original work and meets the Faculty's Expectations of Originality.

UNIVERSITÉ Concordia UNIVERSITY
GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

1. **Consider the execution of an addition instruction. Before the execution of this instruction both C and V flags are cleared. If the instruction uses the S suffix, then give two examples (one example of operands which results in setting c, one example of operands which results in setting v flag) of operands that can result in setting of C and V flags?**

   Suppose we have the following assembly instruction.

   ```
   ADDS Rd, Rs
   ```

   (a) **Example where C bit is set.**
   If `Rd = FFFF FFFF` and `Rs = 0000 0001`, then the result of this operation is as follows.

   ```
        ADDS Rd, Rs
   Rd = 0000   Rs = 0001
   N = 0, Z = 1, C = 1, V = 0, Q = 0
   ```

   In this example, the carry bit is set because a carry occurred during the addition operation. When adding one to a 32-bit number that is full of ones, the result is a carry of the most significant bit. Thus, the C bit was set to 1 in this example.

   (b) **Example where V bit is set.**
   If `Rd = 7000` and `Rs = 1001`, then the result of this operation is as follows.

   ```
        ADDS Rd, Rs
   Rd = 8001   Rs = 1001
   N = 1, Z = 0, C = 0, V = 1, Q = 0
   ```

   In this example, the overflow bit is set to one as an overflow occurred. Assuming signed numbers, when adding 7000 with 1001, the result is 8001. This would be correct, if we were dealing with unsigned integers. However, since it is signed, our result in Rd is thus a negative number which is incorrect. This error is caught when the overflow signal is set to 1. Therefore, the V bit was set to 1 in this example.

2. **Consider the addition instruction with flexible second operand Operand2, given below.**

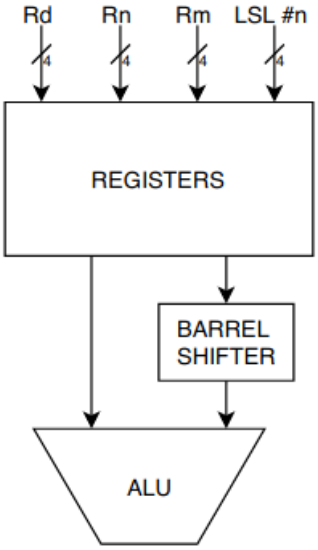
   ```
   ADD Rd, Rn, Rm, LSL #n
   ```

   It is desired to modify this instruction to the following new syntax.

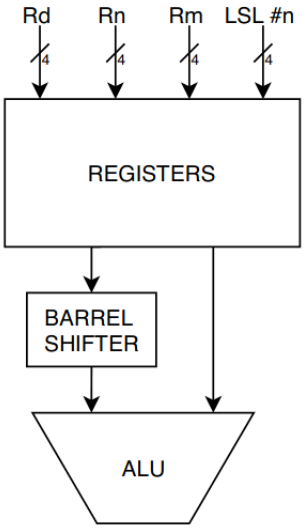
   ```
   ADD Rd, Rm, LSL #n, Rn
   ```

   This new syntax can be called addition instruction with flexible first operand Operand1. What minimum change in the hardware architecture is required for the implementation of this instruction? (Draw your hardware modifications)

   As per the course slides, we know there is a barrel shifter between the register bank and the ALU within the processor. This barrel shifter allows a user to execute instructions such as ADD Rd, Rn, Rm, LSL #n. The current hardware architecture for an instruction like this would look like the following diagram.



These different fields for registers in the encoding define the hardware of our processor as well. For instance, Rs and Rd correspond to specific register buses, as seen in the image above. For the first instruction presented (ADD Rd, Rn, Rm, LSL #n), the hardware above is correct and the barrel shifter would have allowed for the LSL #n. However, if we wanted to modify the language's syntax to be able to execute instructions such as ADD Rd, Rm, LSL #n, Rn, the architecture of the processor would have to change. The order of the two operands would have to swtich, in order to allow the second to last field of "ADD" to have a LSL. The diagram below depicts the hardware change that must occur.



Thus, the minimum hardware architecture change for the change in syntax here is to **swap the operand busses going to the ALU**. By switching the operand order going to the ALU, we could allow the second-to-last passed operand to be shifted by a barrel shifter.

3. **Write the equivalent assembly instructions for each of the following C instructions. Assume x is R1 register, y is R2, z is R3.**

   (a) `y = y + x *32;`

   ```
   LSL R1, R1, #5
   ADD R2, R2, R1
   ```

   (b) `z = y + x*x;`

   ```
   MLA R3, R1, R1, R2
   ```

4. **Assume you want to design the instruction fields for a simple processor. This simple processor has only 128 simple instructions. The register bank is made of 32-8 bit registers. How would you design the instruction fields for the following cases**

   Given: 128 instructions, 32 (8-bit) registers.
   From this information, we can deduce the amount bits needed for op-codes and register addressing.

   If we have 128 instructions, then we know there are $2^n = 128$ bits dedicated for the op-code. Thus, $n = \frac{\ln(128)}{\ln(2)} =$ **7 bits for op-code**.

   We also know that
   $Instruction\ Field = OpCode\ Field + Immediate\ Field + Register\ Fields$ (1)

   (a) **The instruction field is 16 bits fixed and its made of Rds (one register for both source and destination), Rs, immediate and opcode fields and only the first 9 registers can be used as Rs and Rds in the instruction (what would be the range of bits for each fields?)**

   With an instruction field of 16 bits, and two registers (Rs and Rds), we thus have $2^n = 16$ bits dedicated for the register addressing. Thus, $n = \frac{\ln(16)}{\ln(2)} =$ **4 bits for register addressing**. Therefore, using eq. (1),

   $Immediate\ Field = Instruction\ Field - OpCode\ Field - Register\ Fields$
   $Immediate\ Field = 16 - 7 - (4 + 4) =$ **1 bit**

   Thus, Opcode field would be bits 16-9, Rds field would be bits 9-5, Rs field would be bits 5-1, and the **immediate field would be bit 1**.

   (b) **The instruction field is 32 bits fixed and it is made of Rd, Rs1 and Rs2 (separate registers for source and destination), immediate and opcode fields (what would be the range of bits for each fields?)**

   With an instruction field of 32 bits, and three registers (Rd, Rs1, and Rs2), we thus have $2^n = 32$ bits dedicated for the register addressing. Thus, $n = \frac{\ln(32)}{\ln(2)} =$ **5 bits for register addressing**. Therefore, using eq. (1),

   $Immediate\ Field = Instruction\ Field - OpCode\ Field - Register\ Fields$
   $Immediate\ Field = 32 - 7 - (5 + 5 + 5) =$ **10 bits**

   Thus, Opcode field would be bits 32-25, Rd field would be bits 25-20, Rs1 field would be bits 20-15, Rs2 field would be bits 15-10, and the **immediate field would be bits 10-1**.

(c) **The instruction field is 32 bits fixed and it is made of Rds, Rs and (one register for source and destination), immediate and opcode fields (what would be the range of bits for each fields?)**

With an instruction field of 32 bits, and two registers (Rs and Rds), we thus have $2^n = 32$ bits dedicated for the register addressing. Thus, $n = \frac{\ln(32)}{\ln(2)} =$ **5 bits for register addressing**. Therefore, using eq. (1),

$$Immediate\ Field = Instruction\ Field - OpCode\ Field - Register\ Fields$$
$$Immediate\ Field = 32 - 7 - (5 + 5) = \textbf{15 bits}$$

Thus, Opcode field would be bits 32-25, Rd field would be bits 25-20, Rs1 field would be bits 20-15, and the **immediate field would be bits 15-1**.

5. **Assume the content of the registers are as follows before any instruction execution:**

   (a) R1 = C3AB H

   (b) R2 = A0B2 H

   (c) R3 = 1108 H

   Also assume the flags are: N=0, Z= 1, C= 1, V=0, Q=0

   Indicate the content of each register and flag after each instruction execution bellow:

   (a) ADD R1, R2

R1	R2	R3	N	Z	C	V	Q
645D	A0B2	1108	0	1	1	0	0

   (b) ADD R3, R1, R2

R1	R2	R3	N	Z	C	V	Q
C3AB	A0B2	645D	0	1	1	0	0

   (c) ADDS R1, R2

R1	R2	R3	N	Z	C	V	Q
645D	A0B2	1108	0	0	1	1	0

   (d) ADDS R3, R1, R2

R1	R2	R3	N	Z	C	V	Q
C3AB	A0B2	645D	0	0	1	1	0

   (e) ADCS R3, R1, R2

R1	R2	R3	N	Z	C	V	Q
C3AB	A0B2	645E	0	0	1	1	0

   (f) ADDNE R1, R2

R1	R2	R3	N	Z	C	V	Q
C3AB	A0B2	1108	0	1	1	0	0