COMS 321
Written 3 & 4

Aryan Rao
aryanrao@iastate.edu
264954748

3.1

5ED4 − 07A4 (16 bit unsigned numbers)
0101111011010100
-0000011110100100
Ans: 0101011100110000 : 5730 (hexadecimal) (ANS)

3.2

5ED4 − 07A4 (16 bit signed numbers)
0101111011010100 + (- 0000011110100100)
0101111011010100 + (1111100001011100) (2's complement)
Ans: 0101011100110000 : 5730 (hexadecimal) (ANS) (Same as above)

3.3

5ED4 in binary is : 0101 1110 1101 0100

Each hex digit contains one of 16 different characters (0-9, A-E). Since with 4 binary bits you can represent 16 different patterns, in hex each digit requires exactly 4 binary bits. And bytes are by definition 8 bits long, so two hex digits are all that are required to represent the contents of 1 byte.

3.7

185 in Binary: 10111001
122 in Binary: 01111010
Answer: 00110011 = 51 (decimal) No Overflow/Underflow (Ans)

3.17

Numbers to multiply: 0x33 and 0x55 0x33 in binary 8 bit: 00110011
0x55 in binary 8 bit: 01010101

0x33=00110011=(2^5 +2^4 +2^1 +2^0) and,

0x55=01010101=(2^6 +2^4 +2^2 +2^0) = (2^6 +2^4 +2^2 + 1) Now the best way to multiply these numbers would be to:

1. Initialize our result to 0
2. Shift 0x33, 6 times to the left and add it to the result.
3. Shift 0x33, 4 times to the left and add it to the result.
4. Shift 0x33, 2 times to the left and add it to the result.
5. Add 0x33 to the result

Step 1: Shift 0x33, 6 times to the left (Shifting x times to the left is adding x zeroes to the end)

Result = 0 + 00110011000000 = 00110011000000

Step 2: Shift 0x33, 4 times to the left and add it to the result.

Result = 00110011000000 + 001100110000 = 0111111110000

Step 3: Shift 0x33, 2 times to the left and add it to the result. Result = 0111111110000 + 0011001100 = 01000010111100 Step 4: Add 0x33 to the result

Result = 01000010111100 + 00110011 = 01000011101111 (4335 in binary)

3.39

(A) 1.666015625 * 10^0 = 1.1010101010 * 2^0

(B) 1.9760* 10^4 = 1.0011010011 * 2^14

(C) -1.9744* 10^4 = - 1.0011010010 * 2^14

We need to calculate : A*B + (A*C) -----> Distributive Property A*B :
Exponent : 14 + 0
Sign : + (A positive * positive is a positive)

```
                    1.1010101010 ( A )
x                   1.0011010011 ( B )
                   11010101010
                  11010101010X
                 11010101010XXXX
                11010101010XXXXXX
               11010101010XXXXXXX
             11010101010XXXXXXXXX
= 10.00000010011000011111
= 1.00000010011000011111 * 2^15
```

1.0000000100 11 00001111 Guard = 1, Round = 1 Sticky = 1: Round
= 1.0000000101 * 2^15

```
                    1.1010101010 ( A )
x                  -1.0011010010 ( C )
                     11010101010X
                   11010101010XXXX
                  11010101010XXXXXX
                 11010101010XXXXXXX
              11010101010XXXXXXXXX
          10.00000001111101110 10
```

= 1.0000000011 11 101110100 (Normalize exponent and then adding 1 to it)

A * C = −1.0000000100 × 2^15
Now, consolidating the two

```
  (A * B ) 1.0000000101 * 2^15
+ (A * C) −1.0000000100 × 2^15
  .0000000001 × 2^15
```

Answer = 1.0000000000 × 2^5 (Ans)

4.1
4.1.1

Control signal for this instruction
RegWrite = 1 (true)(Register file written)
ALUSrc = 0 (Second operand for ALU is register value)
ALUOp = 10 (“AND”)(This is R-type instruction)(AND operation)
MemRead = 0 (Data Memory is not read)
MemWrite = 0 (Data Memory is not written)
MemToReg = 0 (ALU output is written to register file)
Branch = 0 (This is not a branch instruction)

4.1.2

Following resources (blocks) perform a useful function for this instruction Registers, ALUsrc mux, ALU, and the MemToReg mux.

4.1.3

All blocks produce some output. The outputs of DataMemory and Imm Gen are not used.

4.2

For the STUR and the CBZ rows: Notice that the MemtoReg field is irrelevant when the RegWrite signal is 0: since the register is not being written, the value of the data on the register data write port is not used. Thus, the entry MemtoReg in the last two rows of the table is replaced with X for don’t care.

For the LDUR Row: A don't care can also be added to Reg2Loc for LDUR, which doesn't use a second register because the ALUSrc mux ignores the resulting “Read data 2” output and selects the sign extended immediate value instead.

4.9
4.9.1

Without Improvement: 950
With Improvement: 1250

4.9.2

Speed up = Old Clock Cycle Time/New Clock Cycle Time
= (950 * 1 * n)/(1250*0.15*n) = 0.8 (Ans) [Where n is the number of instructions]

4.9.3

Adding a multiply will result in the removal of 5% of instructions, the cycle time can grow to as much as 950/0.95 = 1000. Thus, the time for the ALU can increase by 50 (200 ---> 250 now).

4.18

Initialisation: X1 = 11, X2 = 22
X3 = 33 and X4 = 26 (Ans)

4.19

Initialisation: X1 = 11, X2 = 22
X5 = 54 (The code will run correctly because the result of the first instruction is written back to the register file at the beginning of the 5th cycle, whereas the final instruction reads the updated value of X1 during the second half of this cycle.)

4.20

```
ADDI X1, X2, #5
NOP
NOP
ADD X3, X1, X2
ADDI X4, X1, #15
NOP
ADD X5, X3, X2
```