Tony Hoss

W0335303, CMPS 375-01

HW#2

Chapter 2 (pg. 117) #’s: 2, 8, 16, 37, 39, 43, 47, 69, 78

Chapter 4 (pg. 282) #’s: 5, 11, 13, 22, 29

Chapter 5 (pg. 333) #’s: 2, 12, 14, 17, 22, 23, 26, 28

CHAPTER 2:

2.) a.) 588­10 ­= 210210­­3

­­ b.) 2254­10 = 33004­5

c.) 65210 = 16217

d.) 310410 = 22289

8.) a.) 25.84375 = 11001.110112

b.) 57.55 = 111001.1000112

c.) 80.90625 = 1010000.111012

d.) 84.874023 = 1010100.1101112

16.) a.) 77

= Signed Magnitude = 01001101

= One’s complement = 01001101

= Two’s complement = 01001101

= Excess-127 = 11001100

b.) -42

= Signed Magnitude = 10101010

= One’s complement = 11010101

= Two’s complement = 11010110

= Excess-127 = 1010101

c.) 119

= Signed Magnitude = 01110111

= One’s complement = 01110111

= Two’s complement = 01110111

= Excess-127 =

d.) -107

= Signed Magnitude = 11101011

= One’s complement = 10010100

= Two’s complement = 10010101

= Excess-127 =

37.) a.) 1011 x 101

= 1000010

b.) 10011 x 1011

= 1111001

c.) 11010 x 1011

= 100011110

39.) a.) 11111101 ÷ 101

= 10111

b.) 110010101 ÷ 1001

= 101101

c.) 1001111100 ÷ 1100

= 110101

43.) Booth’s algorithm (assuming signed two’s complement integers)

a.) 1011 b.) 0011 c.) 1011

x 0101 x 1011 x 1100

= 0110111 = 110001 = (1)0000100

|  |  |  |
| --- | --- | --- |
| 0 | 10111 | 11001000 |

47.) a.) 100.0 =

0.25 =

|  |  |  |
| --- | --- | --- |
| 0 | 01111 | 10000000 |

b.) Adding numbers from part a:

= .11001 x 27  = .11001 x 27

(+) .1 x 2-1 = (+) .000000001 x 27

= .110010001 x 27

c.) The computer would represent the sum from part b, using the simple model for floating-point notation would be:

|  |  |  |
| --- | --- | --- |
| 0 | 10111 | 11001000 |

which in decimal value is = .11001 x 27 = 1100100 = 100­­10.

69.) 0111 1010 101

= 0 1 1 1 1 0 1 0 1 1 1

= Errors occur in bit positions 1, 2, and 8, the error is in bit number 1+2+8 = 11.

78.) The codeword is 01001101100. Division at the receiver should yield a zero remainder.

CHAPTER 4:

5.) a.) 4M x 2 bytes = 22 x 220 x 2 = 223 total bytes, so 23 bits are needed for an address.

b.) 4M words = 22 x 220 = 222 , so 22 bit would be needed for an address.

11.) 16M x 16 memory built using 512K x 8 RAM chips.

a.) 64 RAM chips are needed

b.) 2

c.) 512K = 219 , so 19 bits would be needed.

d.) 32

e.) 16M = 224 , so 24 bits would ne needed.

f.) Bank 0 (000)

g.) Bank 14 if counting from 0, Bank 15 if counting from 1.

13.) a.) 8

b.) 16

c.) 216

d.) 216 – 1

22.) Hexadecimal code:

1108

3109

9106

3109

2108

7000

3108

9103

0023

0001

29.) MARIE assembly language code segment:

Hex Address Label Instruction

100 If, LOAD X / Load X

101 SUBT One / Subtract 1, store result in AC

102 SKIPCOND 800 / If AC>0 (X>1), skip the next instruction

103 JUMP Endif / Jump to Endif if X is not greater than 1

104 Then, LOAD X / Reload X so it can be doubled

105 ADD X / Double X

106 STORE Y / Y = X + x

107 CLEAR / Move 0 into AC

108 STORE X / Set X to 0

109 Endif, LOAD Y / Load Y into AC

10A ADD One / Add 1 to Y

10B STORE Y / Y = Y + 1

10C HALT / Terminate program

10D X, DEC ? / X has starting value, not given in problem

10E Y, DEC ? / Y has starting value, not given in problem

110 One, DEC 1 / Use as a constant

CHAPTER 5:

2.) a.) 0x456789A1

Address 10­­­16­ ­­ 11­­­­16 1216 1316

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Big Endian | 45 | 67 | 89 | A1 |
| Little Endian | A1 | 89 | 67 | 45 |

b.) 0x0000058A

Address 1016 1116 1216 1316

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Big Endian | 00 | 00 | 05 | 8A |
| Little Endian | 8A | 05 | 00 | 00 |

c.) 0x14148888

Address 1016 1116 1216 1316

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Big Endian | 14 | 14 | 88 | 88 |
| Little Endian | 88 | 88 | 14 | 14 |

12.) a.) X \* Y + W \* Z + V \* U

= X Y \* W Z \* V U \* + +

b.) W \* X + W \* (U \* V + Z)

= W X \* W U V \* Z + \* +

c.) (W \* (X + Y \* (U \* V)))/ (U \* (X + Y))

= X Y Z + V W - \* Z + +

14.) a.) W X Y Z - + \*

= W \* (X + Y – Z)

b.) U V W X Y Z + \* + \* +

= U + (V \* (W + (X \*(Y + Z))))

c.) X Y Z + V W - \* Z + +

= X + ((Y + Z) \* (V – W) + Z)

22.) a.) Yes, it is possible to have all those types of address instructions.

= 5 2-address instructions:

000 xxxx xxxx

100 xxxx xxxx // 000xxxxxxxx to 100xxxxxxxx make 5 2-address instructions

= 45 1-address instructions:

1010000 xxxx

1011111 xxxx // 1010000 through 1011111 make 16 1-address instructions

1100000 xxxx

1101111 xxxx // 1100000 through 1101111 make 16 1-address instructions

1110000 xxxx

1111100 xxxx // 1110000 through 1111100 make 13 1-address instructions

= 32 0-address instructions

111 1110 0000

111 1110 1111 // 11111100000 to 11111101111 make 16 instructions

111 1111 0000

111 1111 1111 // 11111110000 to 11111111111 make 16 instructions for a

// total of 32 0-address instructions

b.) A maximum of 30 one-address instructions could be added to the instruction set.

22.)

|  |  |
| --- | --- |
| Mode | Value loaded into AC |
| Immediate | 500 |
| Direct | 100 |
| Indirect | 600 |
| Indexed | 800 |

23.) Speedup S = (200ns x 200) / ((5 + 200 – 1)(40ns))

= 40000 / 8160 = 4.91 (speedup ratio)

Maximum Speedup = (5 x 40ns) / (40ns) = 5 (which is the number of stages in the pipeline)

26.) A = (B + C) \* (D + E)

a.) 3-address machine

|  |
| --- |
| Add R1, B, C |
| Add R2, D, E |
| Mult A, R1, R2 |

b.) 2-address machine

|  |
| --- |
| Load R1, B |
| Add R1, C |
| Load R2, E |
| Add R2, E |
| Mult R2, R1 |
| Store A, R2 |

c.) 1-address machine

|  |
| --- |
| Load B |
| Add C |
| Store Temp |
| Load D |
| Add E |
| Mult Temp |
| Store A |

d.) 0-address machine

|  |
| --- |
| Push B |
| Push C |
| Add |
| Push D |
| Push E |
| Add |
| Mult |
| Store A |

28.) a.) Mode field = 3 bits (23, or 3 bits)

b.) Register field = 6 bits (26, or 6 bits)

c.) Address field = 18 bits (218, or 18 bits)

d.) Opcode field = 5 bits (32 bit instruction – (3 + 6 + 18) = 32 – 27 = 5 bits)