CZ1106 Tutorial ONE

Data and Instructions

# Question 1

a)

(8421)

0111 1111

7F

15\*1 + 7\*16 = 127 ✅

2’s complement

0111 1111 is positive.

111 1111

001 111 111

1 7 7

1\*8^2 + 7\*8^1 + 7 = 127 ✅

b)

1111 1111

FF

15 + 15\*16 = 255 ✅

2’s complement

1111 1111 is negative.

Invert all the numbers (1’s complement): 0000 0000

Add 1: 0000 0001

-1 ✅

c) 0 ✅

d)

1000 0000

2^7 = 128 ✅

2’s complement:

Invert: 0111 1111

Add 1: 1000 0000

-128 ✅

e)

(8421)

1111 1110

F E

15\*16 + 14 = 254 ✅

2’s complement:

Invert: 0000 0001

Add 1: 0000 0010

-2. ✅

# Question 2

Largest: 1111 1111 unsigned = 0 to 255 ✅

Smallest: 1000 0000 2’s complement = -128 to 127 ✅

# Question 3

a) char is 1 byte = 8 bits = 0000 0000 – 1111 1111 = 0 to 255 ✅

b) short int is 2 bytes = 16 bits = (2^16-1)/2 = 32767 --> -32768 to 32767 ✅

c) unsigned = 2^16-1 = 0 to 65535 ✅

d) long int is 4 bytes = (2^(4\*8)-1)/2 = - 2147483648 to 2147483647 ✅-2^31 to 2^31 -1

# Question 4

a) signed char ✅

b) unsigned short int ✅

c) unsigned long long int ✅

d) signed char ✅\_Bool from the stdbool.h header

# Question 1.2

1) MSB>7. a, d ✅

2) c,e ✅b is not alphanumeric ASCII

3)

0xFF is negative.

F F

1111 1111

Invert: 0000 0000

Add 1: 0000 0001

-1 ✅

0x0F is positive

15 ✅

unsigned char c;

c = 77;

c = c-5;

What is the ascii character of c?

c is originally 77, i.e. 0x4D. Minus 5, go up ascii table to 0x48.

4)

0xFFFFFFFF: no. ❌0xF and 0xFF and 0xFFFFFFFF are the same value of (-1). Remember in 2’s complement, these numbers are negative, so if you invert and +1 they are actually –1.

0x0000000F: yes, as 0x0F ✅

5) Add 5 zeroes to the front, then add the 2nd last digit to the front ❌ Just extend the sign.

# Question 1.3

1) 511 / 16^2 = 1 R 255

255 / 16 = 1 R 15

1FF

0x003. ❌0x002 Little Endian. ✅511 = 01FF. 01 is the big end, FF is the little end. Storing the little end first means little endian.

2) 636261 (0x000F to 0x000C) ❌Little Endian only applies to integers. 0x4C6F67696E3A 00 = “\*LOGIN”

3)

unsigned char is 1 byte. r.i = 0x67 ✅

long int is 4 bytes. r.j = 696E3A00 ✅

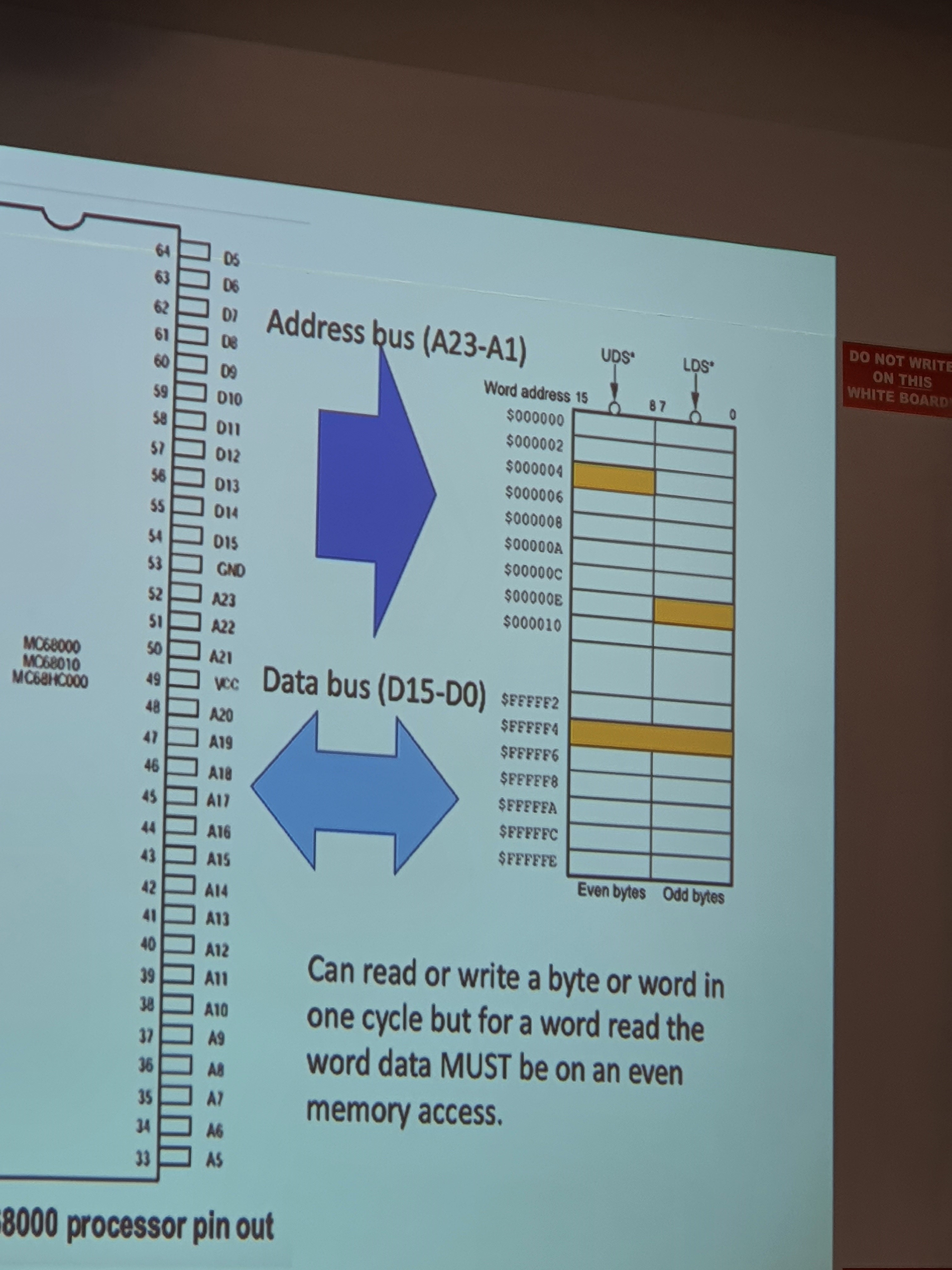
r.a[0] = 0x61 ✅

r.a[2] = 0x63 ✅

# Question 1.4

1) A0-A23.

Max = 2^24-1 = 16777215 bytes = 16Mbytes ✅each permutation of ‘0’s and ‘1’s provided by the address buses A0-A23 points to a single memory address which contains a byte. Hence 2^24 is the total number of bytes addressable.



2) D0-D15

16 bits = 2 bytes ✅

3)

struct rec {

unsigned char i;

long int j;

char a[3];

};

struct rec r;

char a[3] is 3 bytes, unsigned char j is 1 byte. Swap long int j (4 bytes) and unsigned char i. ✅

# Question 1.5

1) 32 bits ✅

2)

unsigned magnitude: 0x0000 0000 to 0xFFFF FFFF ✅

2’s complement:

smallest: -2^(n-1) = first number 1, the rest 0 = 0x800 000 ✅

largest: 2^(n-1)-1 = first number 0, the rest 1 = 0x7FFF FFFF ✅

3) R0-R13 and LR and PC and CSPR ❌CSPR is not a register. Yes. ✅R13 is the stack pointer, R14 is the link register, R15 is the program counter

4)

MOVS R0, #0x00000001

Move the 32-bit value of 1 into the R0 register and set the CSPR status bits if applicable (yes: Z) ✅

MOVS R1, #0xFFFFFFFF

Move the value of –1 into the R1 register and set the CSPR status bits if applicable (yes: N) ✅

MOVS R2, #0x7FFFFFFF

Move the value of 2^31-1 (i.e. the largest possible value) into the R2 register and set the CSPR status bits if applicable (no) ✅

ADDS R3, R0, R1

Sum the values of R0 (1) and R1 (-1) and put the result into R3. Also set the CSPR status bits if applicable (yes: Z and C). The value 0 is stored into R3. ✅

ADDS R4, R0, R2

Sum the values of R0 (1) and R1 (2^31-1) and put the result into R3. Also set the CSPR status bits if applicable (yes: V and N). The value of –2^31/0x8000 0000 is stored into R4. ✅

5) Yes. 0xFFFF FFFF is –1, which is a negative number, hence change N to 1. ✅

6) Yes. 0x7FFF FFFF is 2^32-1 which is a positive number, hence change N to 0.

7) ADDS R3, R0, R1 (0x0000 0001 and 0xFFFF FFFF)

a) 0x0000 0000

b) Yes if they are 2’s complement numbers

c) 0 1 0 1.

d) The number is not negative but zero (0 1). There is no overflow if they are signed numbers but there will be overflow if they are unsigned numbers (0 1).

8) ADDS R4, R0, R2 (0x0000 0001 and 0x7FFF FFFF)

a) 0x8000 0000

b) Yes if they are unsigned numbers

c) 0 1 1 0.

d) The number is not negative but zero (0 1). There is overflow if they are signed numbers but no overflow if they are unsigned numbers (1 0).