EEL4742C: Embedded Systems

Homework 1

Part a) A memory is byte addressable and has a 17-bit address. All the addresses are valid. What is the total size of the memory

$2^7 \cdot 2^{10}$

$128 \cdot 1024 = 131,072$ bytes or 128 kb

Part b) A memory is byte addressible and has a total size of 18432 bytes (18kb). What is the smallest address size that can be used for this memory?

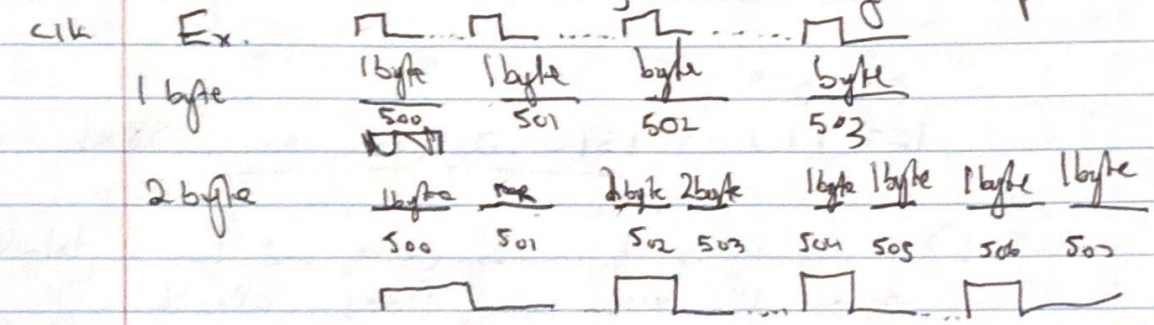
$18 \cdot 1024 = 18432$

$\times 2^4$ or $2^5 \cdot 2^{10} = 2^{\boxed{15}} =$ 15 bit address

b

$\times 16$ $\quad 32 \cdot 1024 =$

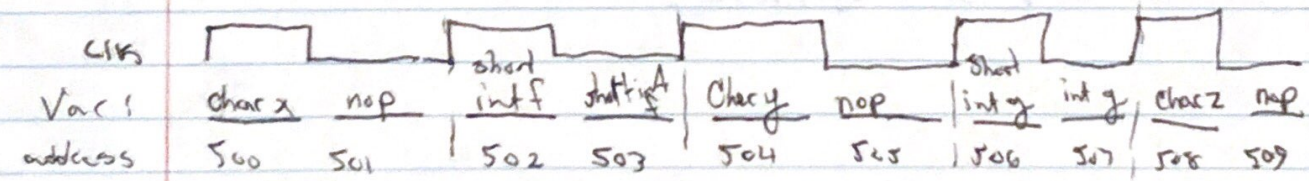
2. Part A) Depending on the architecture, a single byte variable (8 bit) can store in each consecutive spot in a byte addressable configuration, In a 2 byte accessible, than only multiples of 2.



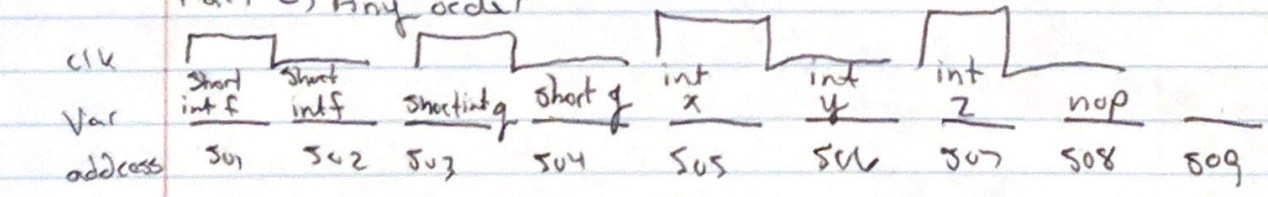
Part b)

unsigned char x 8bit
short int f 16bit
unsigned char y
short int g
unsigned char z

A 16 bit requires 2 8 bit memory slots filling a 2 byte addressable configuration



Part c) Any order



3. Part A; explain the endians

Little endian is when an address in hexadecimal is arranged where the most significant byte is placed in ~~the~~ highest address.

Big endian is when the address in hexadecimal is arranged where the most significant byte is placed in the lowest address

Part B; 0x12EF

Big	12	EF
address	500	501
Little	EF	12

Part C; MSP430?

Little endian

4. A) Flash code space 0x4400 to 0BFF

0x0400

0000 0400 → 1024 bytes

1011 1111 1111 → 3071 bytes

3071 − 1024 = 2047 + 1 = 2048 bytes

$2^1 \cdot 2^{10}$ bytes

2^{11} bytes

B) Vector table (address)

16 bit ranging 0xFFC0 to 0xFFFF

1100 0000 → 192 → 48+15

65535 − 63 = 65472 = 0xFFC0

3 bytes

64 bytes free

$16\overline{)512}$ → 32 Vectors at 16 bits a piece

48↓

32

32

0

5. Clock technologies

	Crystal	RC
Startup speed	Slow startup speed	Fast start up speed (< 1ms)
Accuracy	Very accurate ~ 3 μsecond / month	Not very accurate
Stability	Stable but high power comption	Unstable ex. 12khz hang variation between 4khz and 20khz / low power consumption
Price	expensive	Cheap

6. Part A) embeedded ccgs extensions in c?, why
Unique cequirment that C-Standard does not have. Extension ccc like additions to the language that ccc used by the MCU.

Part B) Because the ~~code~~ extensions ccc not C-standered, tk extensions cause low ~~for~~ portability between pccgcams.

Part C) Intrinsic functions ccc functions that cannot be expressed in C.

Part D). We use the intrinsic functions _enable/_disable _interrupts() because C cannot access the cegisters. Only assembly code can.

Question 7

Part A) bits in int in C

Using sizeof(int) in my own PC, int is 4 bytes or 32 bits, but it can be 2 bytes or 16 bits as well (short int)

Part B) Because of the example I gave, Embedded uses its own property definition so it is completely understood how many bits the data type for the purpose of correctly marking and assigning bits to functions.

Part C) Using an unsigned int 8 or 16 bit would not work to increase the delay to 90000 because 16 bit only has a limitation of up to 65536. This is calculated by taking the number of bits and using it as a power for 2 (2^x, where $x = 16$)

To create the delay, you need to ~~for the in~~ loop the initial for loop twice (for(i=0; i<2; i++){

for(j=0; j<45000; j++){}

}

Part D) The compilers optimization option ignores the code because the loop is empty. To fix this, we have to set the compilers optimization to 0.

Question 8

Part A)

// Set bit 6

```
#define BIT5 BIT5
int main() {
    uint_8t data = 0x00;
    data |= BIT5;   // OR to set Data bits
// clear bit 6
    data &= BIT5;
// invet bit 6
    data ^= BIT5;
```

5

PART B)

```
// set bits 4 and 5
#define BIT4 BIT4   // 5th bit
#define BIT3 BIT3   // 4th bit
int main() {
    uint_8t data = 0x00;
    data |= (BIT3 | BIT4);   // 0000 0000 Data
                             // 0001 1000 BIT 3|4
                             // 0001 1000 Data |=
// Clear bits 4 and 5
    data &= ~(BIT3 | BIT4);   // Demorgans ~BIT4 & ~BIT3
// invert BITS 4 and 5
    data ^= (BIT3 | BIT4);
// Set bit 4 clear bit 5
    data |= BIT3;
    data &= BIT4;
```

Set |
clr &
invert ^

Part c)

```
// Check 2 is 1
#define BIT0 BIT0
#define BIT1 BIT1
#define BIT2 BIT2
#define BIT3 BIT3
    int main(){
      uint_8t data = 0x00;
    if((Data & BIT1) == BIT0);
// check 2 is 0
    if((Data & BIT1) == 0);
// Check if 3,4 are 1,1
    if((data & (BIT2|BIT3)) == (BIT2|BIT3));
// Check 3 is 0, 4 is 1
    if((Data & (BIT2|BIT3)) == BIT3);
// Check if bits 3,4 are 00
    if(!Data & (BIT2|BIT3)) == 0);
// Check if clk = 0 2 4 6
    if(!
```

Question 9 SLP_2 CLK_6

PART A: CTL = ~~SLP_4~~ | ~~CLK_5~~ | CAP_1 | IE;

PART B: CTL val and mask

CTL ~~Data~~ = 0000 0000

SLP_2 | 10 00 00 00

OR CLK_6 | 00 11 00 00

CAP_1 | 00 00 01 0

IE; 0000 000 1

1011 0011 = CTL

PART C: SLP to 1, SLP Unknown

CTL = ??? 0011

AND 00? 1 1111

00? 1 0011

SET 0?X0 0000

OR 0?11 0011

CODE:

~~CTL &= SLP_3;~~

CTL &= ~SLP_3;

CTL |= SLP_1;

PART D; CLK to 5, CLK unknown

CTL &= ~CLK_7 Check/reset

CTL |= CLK_5 set

Part E; if checks is SLP = 1

~~if ((CTL =~~

if((CTL & SLP_3) == SLP_1)

PART F; if CLK is 5

if((CTL & CLK_7) == CLK_5)

Part G; if CLK is 2 4 6 0

~~if((CTL & (CLK_6)) != CLK_1)~~

111 001 110

& 110 110 100

110 001

if((CTL & (CLK_6)) != clk_1)

if((CTL ^ (CLK_6)) != clk-1)