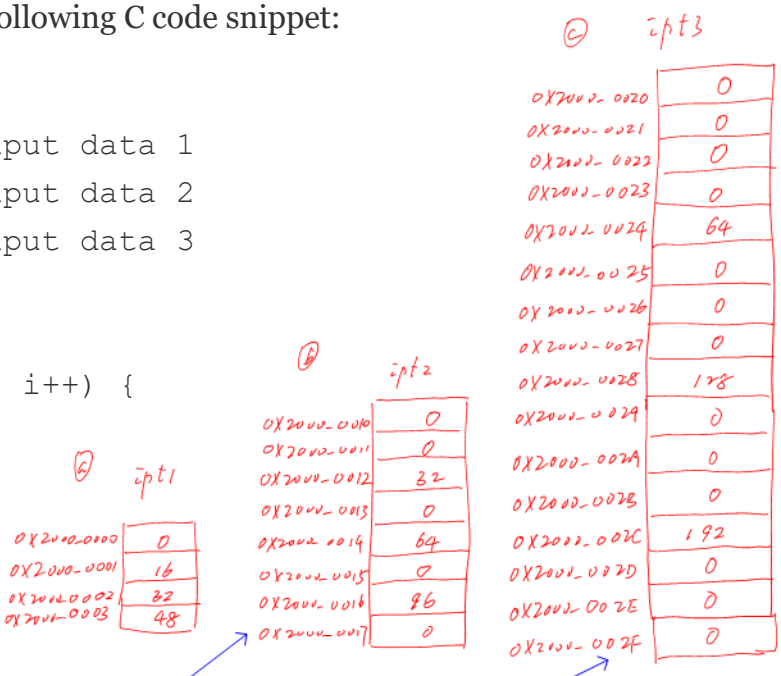
#HW, cec32x_HW_34_HW7_load_n_store_instructions, 140 points total.

HW 7. Load and Store Instructions

Prob 1 (45 points total) Consider the following C code snippet:

```
// Global variable
uint8_t ipt1[4];      // Input data 1
uint16_t ipt2[4];     // Input data 2
uint32_t ipt3[4];     // Input data 3

int main(void) {
    for (int i = 0; i < 4; i++) {
        ipt1[i] = i<<4;
        ipt2[i] = i<<5;
        ipt3[i] = i<<6;
    }
}
```



Assume the addresses of arrays `ipt1`, `ipt2`, and `ipt3`, are `0x2000_0000`, `0x2000_0010`, and `0x2000_0020`, respectively. Draw the memory map (**addresses in hexadecimal and contents in decimal**) in a similar way to those shown on page 1 of class notes of Module 7. Note that to be consistent with the display of Keil, please draw it with the lowest address at the top with each address containing ONE byte. (Note that we use little endian here; this will not be reminded later.)

- (10 points) Draw the memory map of the first 4 addresses starting from "ipt1".
- (15 points) Draw the memory map of the first 8 addresses starting from "ipt2".
- (20 points) Draw the memory map of the first 16 addresses starting from "ipt3".

Prob 2. (10 points) Suppose a word is read as "0x12345678" using little endian. What will be the value of this number if it were read using big endian? 0x7856_3412

Prob 3. (25 points) Assume the 16 1-BYTE values starting from address `0x2000_0000` are increasing numbers from 0, 1, 2, to 15. Consider the running of the following asm code:

```
LDR r4, =0x20000000;   (1)
LDR r0, [r4, #4];      (2)
LDR r1, [r4, #2]!;     (3)
LDR r2, [r4], #4;      (4)
LDR r3, [r4];          (5)
```

(1) r4 = 0x2000_0000

(2) r0 = 0x0706_0504, r4 = 0x2000_0000

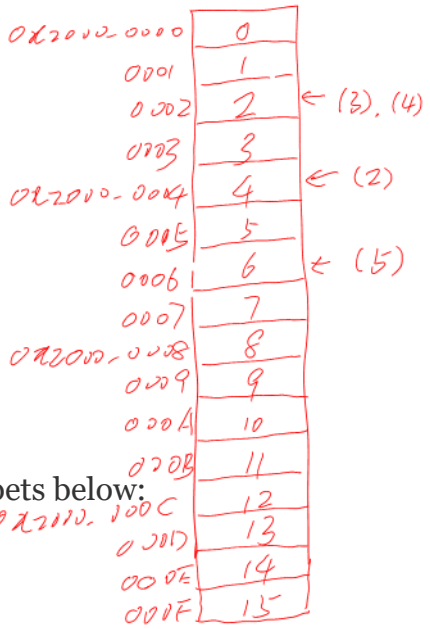
(3) r1 = 0x0504_0302, r4 = 0x2000_0002

(4) r2 = 0x0504_0302, r4 = 0x2000_0006

(5) r3 = 0x0908_0706

- (4 points) What is the value of `r4` after running (1)?
- (6 points) What are the value of `r0` and `r4` after running (2)?
- (6 points) What are the value of `r1` and `r4` after running (3)?

- (6 points) What are the value of `r2` and `r4` after running (4)?
- (3 points) What is the value of `r3` after running (5)?



Prob 4. (70 points) We have learned the following in the class:

1. `LDR Rt, [Rn, #offset]`: Load with immediate offset
2. `LDR Rt, [Rn, #offset]!`: Load with pre-indexed offset
3. `LDR Rt, [Rn], #offset`: Load with post-indexed offset
4. `LDR Rt, [Rn, Rm, shift]`: Load with register offset

Now, we practice these instructions here with the given program snippets below:

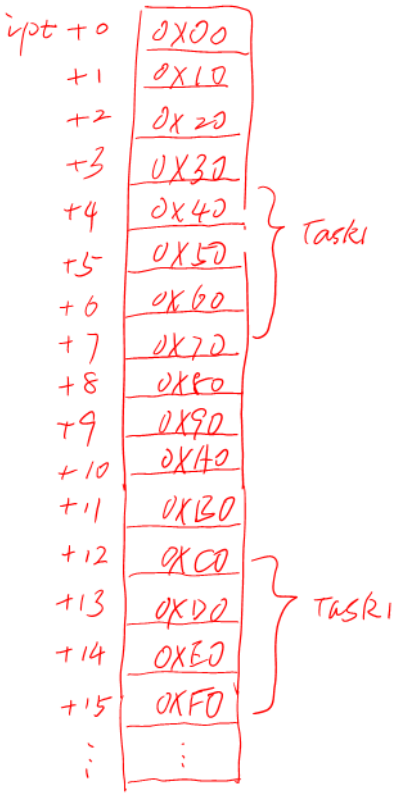
Part A: C code:

```
// Functions defined in a .s file
extern void task1(uint8_t *pIpt, uint32_t *pOup);
extern void task2(uint8_t *pIpt, uint32_t *pOup);
extern void task3(uint8_t *pIpt, uint32_t *pOup);
extern void task4(uint8_t *pIpt, uint32_t *pOup, int i);

// Global variable
uint8_t ipt[16];     // Input data
uint32_t opt[4];     // Output data

int main(void) {
    for (int i = 0; i < 16; i++) {
        ipt[i] = i<<4;
    }

    // Task 1.
    task1(ipt, opt);
    printf("Out1 = 0x%X, Out2 = 0x%X\n", opt[0], opt[1]);

    // Task 2.
    task2(ipt, opt);
    printf("Out1 = 0x%X, Out2 = 0x%X\n", opt[0], opt[1]);

    // Task 3.
    task3(ipt, opt);
    printf("Out1 = 0x%X, Out2 = 0x%X\n", opt[0], opt[1]);

    // Task 4.
    task4(ipt, opt, 1);
    printf("Out1 = 0x%X, Out2 = 0x%X\n", opt[0], opt[1]);
```

```
    while (1);
}
```

Part B: asm code:

```
        EXPORT task1
        EXPORT task2
        EXPORT task3
        EXPORT task4

        ALIGN       ; Align the data in the boundary of 4 bytes.

task1   PROC
        LDR  r2, [r0, #4]
        STR  r2, [r1, #0]
        LDR  r2, [r0, #0xC]
        STR  r2, [r1, #4]
        BX   lr
        ENDP

task2   PROC
        LDR  r2, [r0, #4]!
        STR  r2, [r1, #0]
        LDR  r2, [r0, #4]!
        STR  r2, [r1, #4]
        BX   lr
        ENDP

task3   PROC
        LDR  r2, [r0], #4
        STR  r2, [r1, #0]
        LDR  r2, [r0], #4
        STR  r2, [r1, #4]
        BX   lr
        ENDP

task4   PROC
        LDR  r3, [r0, r2, LSL #2]
        STR  r3, [r1, #0]
        ADD  r2, #1
        LDR  r3, [r0, r2, LSL #2]
        STR  r3, [r1, #4]
        ADD  r2, #1
```



```
        BX    lr
        ENDP
```

Answer the following questions (without running the program first):

(a) • (10 points) What will be the printout after Task 1? (a) 0x7060_5040, 0xF0E0_D0C0

(b) • (15 points) What will be the printout after Task 2 and what will be the values in `r0` and `r1` in hexadecimal after running Task 2 assuming their values are 0x20001010 and 0x20001040, respectively when the function is called?

(c) • (15 points) What will be the printout after Task 3 and what will be the values in `r0` and `r1` in hexadecimal after running Task 3 assuming their values are 0x20001010 and 0x20001040, respectively when the function is called?

(d) • (15 points) What will be the printout after Task 4 and what will be the values in `r2` and `r3` in hexadecimal after running Task 4?

(e) • (15 points) What will be the corresponding C code for each task?

(b) 0x7060_5040, 0xB0A0_9080. r0 = 0x2000_1018, r1 = 0x2000_1040

(c) 0x3020_1000, 0x7060_5040. r0 = 0x2000_1018, r1 = 0x2000_1040

(d) 0x7060_5040, 0xB0A0_9080, r2 = 3, r3 = 0xB0A0_9080.

(e) The solution is not unique:

Task 1

```
*opt = *((uint32_t *)(ipt + 4));
*(opt+1) = *((uint32_t *)(ipt + 12));
```

Task 2

```
uint32_t ipt32 = (uint32_t *) ipt;
*opt = *++ipt32;
*(opt+1) = *++ipt32;
```

Task 3

```
uint32_t ipt32 = (uint32_t *) ipt;
*opt = *ipt32++;
*(opt+1) = *ipt32++;
```

Task 4

```
uint32_t ipt32 = (uint32_t *) ipt;
*opt = *(ipt32 + i++);
*(opt+1) = *(ipt32 + i++);
```