Tags: #cec32xNotes, #ws, cec32x_ws_34_WS07_load_n_store_in_assembly

WS 7. Assembly Load and Store Instructions and C Pointers

Instruction

In project 140_load_store2, we addressed the load and store operations of 32-bit numbers. In this workshop, we extend these operations to handle 8- and 16-bit numbers. Note that there are two versions of the load instruction for both the 8- and 16-bit numbers—the signed version and the unsigned version. Note also that, as we have learned earlier, the MCU does not care if the numbers are signed or unsigned, but we care—hence it's our programmers' responsibility to use the right version of assembly instructions to have the correct sign extension for loading: filling in zeros or ones for the most significant bits for signed numbers.

Workshop tasks

You are given the following C file, `ws--load_n_store_main.c`, with 6 tasks implemented in C using pointers. It also calls 6 assembly functions defined in an assembly file, `ws--load_n_store_asm_functions_prob.s`.

The `ws--load_n_store_main.c` file:

```c
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>

#define MY_ARRAY_BASE (0x20010000)

// Functions defined in a .s file
extern void task10(void);
extern void task11(void);
extern void task12(void);
extern void task13(void);
extern void task14(void);
extern void task15(void);

// Global variable
// Pointers for the C version.
int8_t      *gPtrArray10c = (int8_t *)   (MY_ARRAY_BASE + 0x00);
uint8_t     *gPtrArray11c = (uint8_t *)  (MY_ARRAY_BASE + 0x40);
int16_t     *gPtrArray12c = (int16_t *)  (MY_ARRAY_BASE + 0x80);
uint16_t    *gPtrArray13c = (uint16_t *) (MY_ARRAY_BASE + 0xC0);
int32_t     *gPtrArray14c = (int32_t *)  (MY_ARRAY_BASE + 0x100);
uint32_t    *gPtrArray15c = (uint32_t *) (MY_ARRAY_BASE + 0x140);
// Pointers for the asm version.
```

```
int8_t      *gPtrArray10a = (int8_t *)   (MY_ARRAY_BASE + 0x20);
uint8_t     *gPtrArray11a = (uint8_t *)  (MY_ARRAY_BASE + 0x60);
int16_t     *gPtrArray12a = (int16_t *)  (MY_ARRAY_BASE + 0xA0);
uint16_t    *gPtrArray13a = (uint16_t *) (MY_ARRAY_BASE + 0xE0);
int32_t     *gPtrArray14a = (int32_t *)  (MY_ARRAY_BASE + 0x120);
uint32_t    *gPtrArray15a = (uint32_t *) (MY_ARRAY_BASE + 0x160);

int8_t gVar1 = 8;

int main(void) {
    // Task 10. Register offset-based addressing---a simple example.
    for (int i = 0; i < gVar1; i++) {
        *(gPtrArray10c + i) = (int8_t)(i*4 - 15); // i is saved in a register
    }
    task10();       // task10: perform the above task using assembly

    // Task 11. Register offset-based addressing---another simple example.
    for (int i = 0; i < gVar1; i++) {
        *(gPtrArray11c + i) = (uint8_t)(i*32 + 2);
    }
    task11();       // task11: perform the above task using assembly

    // Task 12. Register offset-based addressing---more complicated case.
    for (int i = 0; i < gVar1; i++) {
        *(gPtrArray12c + i) = 10 - (*(gPtrArray10c + i))++;
    }               // Note that the input data is changed as well in the above line
    task12();       // task12: perform the above task using assembly

    // Task 13. Immediate offset addressing.
    for (int i = 0; i < gVar1-1; i++) {
        // We use immediate offset addressing twice below (righthand side)
        *(gPtrArray13c + i) = *(gPtrArray11c) + *(gPtrArray11c+1);
        gPtrArray11c++;
    }
    task13();    // task13: perform the above task using assembly

    // Task 14. Immediate offset addressing and pre-index addressing.
    for (int i = 0; i < gVar1-1; i++) {
        uint32_t temp = *(gPtrArray12c);
        *(gPtrArray14c + i) = temp + 8 * (*(++gPtrArray12c));
    }
    task14();    // task14: perform the above task using assembly

    // Task 15. Post-index addressing and immediate offset addressing.
    for (int i = 0; i < gVar1-1; i++) {
        uint32_t temp = *(gPtrArray13c++);
```

```
        *(gPtrArray15c + i) = temp + 16 * (*gPtrArray13c);
    }
    task15();    // task15: perform the above task using assembly

    printf("All tasks are done! \n");

    while (1);
}
```

The ws--load_n_store_asm_functions_prob.s file:

```
        AREA my_fancy_asm_code, CODE, READONLY   ; Define the program area

        ; Export functions defined in this file. These functions need to be declared
        ; in the file calling them.
        EXPORT task10
        EXPORT task11
        EXPORT task12
        EXPORT task13
        EXPORT task14
        EXPORT task15

        IMPORT gPtrArray10a
        IMPORT gPtrArray11a
        IMPORT gPtrArray12a
        IMPORT gPtrArray13a
        IMPORT gPtrArray14a
        IMPORT gPtrArray15a
        IMPORT gVar1

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

task10  PROC
        LDR  r0, =gPtrArray10a  ; Loading the address of the global variable gPtrArray10a
        LDR  r0, [r0]           ; Loading the content of the global variable gPtrArray10a
        LDR  r1, =gVar1         ; Loading the address of the global variable gVar1
        LDR  r1, [r1]           ; Loading the content of the global variable gVar1
        MOV  r2, #0             ; variable (int) i
task10_loop
        CMP  r2, r1                 ; test = r2 - r1
        BGE  task10_end             ; if test >= 0, then branch to task10_end
        MOV  r3, r2, LSL #2         ; r3 <- r2 * 4
        SUB  r3, #15                ; r3 <- r3 - 15
        STRB r3, [r0, r2]           ; r3 -> mem[r0 + r2] or r3 -> mem[r0 + i]
        ADD  r2, #1                 ; r2 <- r2 + 1
        B    task10_loop            ; branch to task10_loop
task10_end
```

```
        BX   lr                     ; return
        ENDP

; If you need to use registers starting from r4, you need to PUSH them first to save the
; run-time environment for the caller. You need to POP them up at the exit of the code.

task11  PROC
        BX   lr
        ENDP

task12  PROC
        PUSH {r4-r5, lr}
        LDR  r0, =gPtrArray10a
        LDR  r0, [r0]
        LDR  r4, =gPtrArray12a
        LDR  r4, [r4]
        LDR  r1, =gVar1
        LDR  r1, [r1]
        MOV  r2, #0
task12_loop
        CMP  r2, r1
        BGE  task12_end
        LDRSB  r3, [r0, r2]
        LDR  r5, =10
        SUB  r5, r3
        STRH r5, [r4, r2, LSL #1]
        ADD  r3, #1
        STRB r3, [r0, r2]
        ADD  r2, #1
        B    task12_loop
task12_end
        POP  {r4-r5, pc}       ; Pop lr to pc, which is the same as BX lr.
        ENDP

task13  PROC
        PUSH {r4-r5, lr}
        POP  {r4-r5, pc}
        ENDP

task14  PROC
        PUSH {r4-r5, lr}
        LDR  r0, =gPtrArray12a
        LDR  r0, [r0]
        LDR  r4, =gPtrArray14a
        LDR  r4, [r4]
        LDR  r1, =gVar1
```

```
        LDR  r1, [r1]
        SUB  r1, #1
        MOV  r2, #0
task14_loop
        CMP  r2, r1
        BGE  task14_end
        LDRSH  r3, [r0]
        LDRSH  r5, [r0, #2]!
        ADD  r3, r5, LSL #3
        STR  r3, [r4, r2, LSL #2]
        ADD  r2, #1
        B    task14_loop
task14_end
        POP  {r4-r5, pc}
        ENDP

task15  PROC
        PUSH {r4-r5, lr}
        POP  {r4-r5, pc}
        ENDP

        END
```

Among the 6 assembly functions, you are given the implementation of three, and you are supposed to finish the remaining three, Tasks 11, 13, and 15. You can refer to the example codes in `140_load_store2` as the functionalities of this workshop and those of that project are very similar.

Note that you need to rename the assembly source file to `ws--load_n_store_asm_functions_soln.s` so that you can save a clean copy of the given code.

Each of the assembly function has 30 points.

When the workshop is done, you should be able to see the results for Tasks 10 to 15 shown in Figures 1 to 7 obtained from the Keil debugger in the enclosed file.

Submission of work

10 points are given for the writing of the report in pdf following the assignment/submission requirements:

- Code snippets of all your own assembly instructions in the `ws--load_n_store_asm_functions_soln.s` file of the project.
- The screenshot of the running results like those given in Figures 1 to 7.