Exercise sheet 6 – Processor architecture

Goals:

- Registers
- Addressing modes

Exercise 6.1: Addressing modes (theoretical)

(a) Which addressing modes can be used for direct realisation of stack-operations on a CISC architecture (Freescale ColdFire, 32 bit architecture)? Are there any alternatives if those addressing modes are not available? Explain this by pushing the content of the D0 register to the stack; after that, pop the stack content to the D1 register. *Hint: You may use some pseudo-code (assembler) to express your idea.*

Proposal for solution: *Hint: MOVE.L for long word (or double word) (4 bytes, 32 bits)*

```
Push D0:   MOVE.L D0, -(SP)
Pop D1:    MOVE.L (SP)+, D1

Push D0:   SUB #4, SP
           MOVE.L D0, (SP)
Pop D1:    MOVE.L (SP), D1
           ADD #4, SP
```

(b) How can a CISC architecture (Freescale ColdFire) support array-accesses? Consider a 32 bit architecture. Use given values to describe your idea. *Hint: You may write some pseudo-code (assembler) and draw a sketch.*

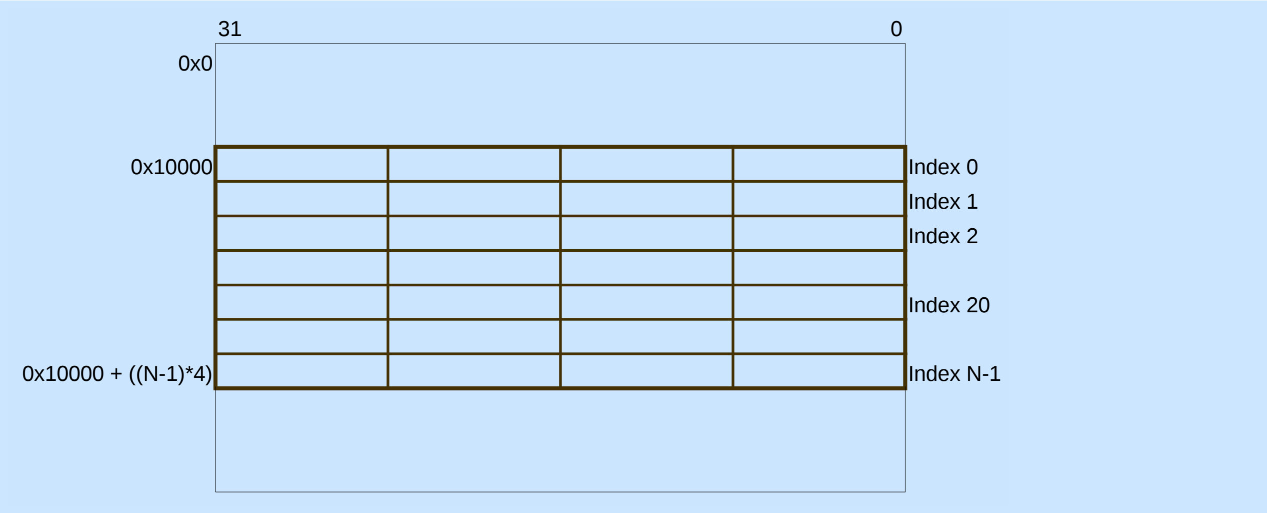
- access to x[i] (element i of array x, x contains integer data)
- x starts at memory address 0x10000
- i = 20
- use the registers A2 and D3

Proposal for solution:

```
int x[N]; // Array with N elements

// Prepare register content
MOVE.L 0x10000, A2
MOVE.L 20,      D3

C: x[i]
Assembly:     (0, A2, D3*4)
results in -> (0, 0x10000, 20 * 4)
```



Exercise 6.2: Understanding a concrete Intel x86/64 instruction (theoretical)

(a) Try to understand the `XCHG` (Exchange Register/Memory with Register) assembler instruction. Here are useful links:

- `https://www.felixcloutier.com/x86/xchg`
- `https://www.amd.com/system/files/TechDocs/24594.pdf` (page 360)

(b) Which addressing modes does the `XCHG` instruction support?

Proposal for solution: It supports register to register and register to memory (and the other memory addressing alternatives).

Exercise 6.3: Use a concrete Intel x86/64 instruction (coding)

Given is the same endianness example („Endianness with integer (coding)“) as from the last exercise: A *big-endian* system program—the Java runtime environment—that transfers data via a file to a little-endian system C program.
Now, we want to use a single assembler instruction to perform the swap operation.

(a) Update the `RA_exercises` repository with `git pull`.

(b) Change into the directory `RA_exercises/sheet_06/Endianness/C_LE_asm_swap`

Proposal for solution: `cd RA_exercises/sheet_06/Endianness/C_LE_asm_swap`

(c) Inspect, build, and run the given C program.

Proposal for solution:

```
make                    #build
./c_le_example          #execute
```

(d) Analyse the output of the C program. What has happened? What could be the cause of this?

Proposal for solution: The C program is reading the content of *output.txt*, which was generated by the java-program. Because of the different endianness of Java (big endian) and C (little endian), the output of the C program is switched.

(e) Fix the problem in the C program, following the *TODOs*. *Hint: use the* `XCHG reg/mem8, reg8` *variant of the* `XCHG` *instruction to perform the swap.*

Proposal for solution:

```c
#include <stdio.h>  //fopen, ...
#include <stdlib.h> //EXIT_SUCCESS
#include <stdint.h> //uint8_t, uint16_t

int main(void) {

    uint16_t value = 0;

    FILE* file = NULL;
    file = fopen("../output.txt", "rb");

    if (file == NULL) {
        printf("Error opening output.txt\n");
        return EXIT_FAILURE;
    }

    fread(&value, sizeof(uint16_t), 1, file);
    fclose(file);

    printf("Read from output.txt -> : %2x\n", value);

    //Hint:
    // Syntax (Intel): Op-code dst, src
    // Example (Intel): MOV EAX, 1; //moves a 1 into the EAX register
    //- with XCHG instruction you can exchange bytes within a register:
    //- with AL you can use the AL register part (byte 0) of the EAX register
    //- with AH you can use the AH register part (byte 1) of the EAX register
    //  - https://www.felixcloutier.com/x86/xchg
    //  - infos: AMD64 Architecture Programmer's Manual Volume 3:
    //           General Purpose and System Instructions: Page 396 (356)
    //           http://support.amd.com/TechDocs/24594.pdf
    //  - https://c9x.me/x86/html/file_module_x86_id_328.html
    //  - https://www.utd.hs-rm.de/infobuch2/Buch_Webseite/kap03/Assemblerbefehle.pdf
    //  - https://www.ibiblio.org/gferg/ldp/GCC-Inline-Assembly-HOWTO.html
    // volatile: let the compiler don't move the the ASM instructions around
    __asm__ volatile (
        "XCHG AL, AH;"    // swaps (exchange) the AL byte with the AH byte
        : "=a" (value)    // output: saves the AX register into the value
                          // can be considered as: MOV value, AX
        : "a"  (value)    // input: loads the value into the AX register
                          // can be considered as: MOV AX, value
    );

    //print the fixed value
    printf("Converted to LE -> : %2x\n", value);

```

(f) Build and run the C program again. Is the problem now solved?

Proposal for solution:

```
make                    #build
./c_le_example          #execute
```