NOVA IMS Information Management School

Project 10

HSTI Fall 2021 - Exam Programming Submission

Student: Nikolaj Skov Wacher
Student no.: e20211470
Course name (PT): Hardware e Software das Tecnologias de Informação
Course name (EN): Informations Technologies Hardware and Software
Curricular Unit Code: 100071
Semester: Fall 2021

Table of Contents

Introduction / Objective

In the exam submission handout, the problem statement for project 10 is titled: ”Write the assembler code that corresponds to the following C code”, with a maximum amount of points to be given, being 13 out 20. The C code to be converted to assembler code is:

```
A=0;
B=0;
for(i=1;i>0;i--);
 {
  A=A+i:
  B=B+2*i:
 }
```

Fig 1. code from hand-out

```
1  A = 0;
2  B = 0;
3  for (i = 1; i > 0; i--) {
4      A = A + i;
5      B = B + 2 * i;
6  }
```

Fig 2. code rewritten with corrected syntax in IDE

The code firstly assigns a variable titled *A* to an integer value of 0, then variable *B* to the integer value of 0. On line 3, the code begins a for loop, with the variable *i* being equal to 1, which will run for as long as *i* is greater than 0, decrementing by 1 each time it has completed a loop. The code within the loop firstly sets the variable *A* equal to itself plus the current value of *i*, secondly, it sets the variable *B* to itself plus 2 times the current value of *i*. Following execution the variable will have changed values, namely:

Variable values at the end of line 2
A = 0;
B = 0;

Variable values at the end of line 6
A = 1;
B = 2;

The objective of the project is to convert this C code into assembler code and create a detailed fluxogram / flowchart, as well as a contributing memory map. The assembler code should have a “main” program that calls the routine, and a sub-routine that implements the desired functionality. After presenting the diagrams, the code will be presented, with detailed comments. Lastly, I will conclude on the project in the conclusions section.

Fluxograms / Flowcharts

Main routine “_start”

The flow begins with the program entry point, which is defined in the .text section (ie. the code section of an assembly program), as the _start routine. The _start routine firstly moves, using the *mov* instruction, storing the db values *A* and *B* in the registers *eax* and *ebx*. It then assigns the register *ecx* to integer 1, which will be used as a counter for our for loop routine. We then use the *cmp* instruction to compare the two operands *ecx* (counter) and 0. If *ecx* is equal to 0, the code moves to the forLoop subroutine, if not, to the subroutine _exit.

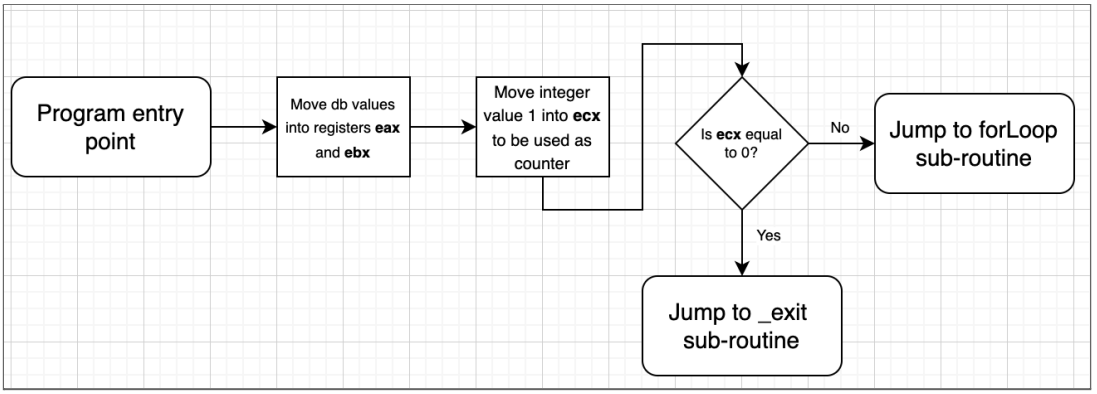

Fig 3. Flowchart for the main routine _start

Sub-routine “forLoop”

If the value stored in *ecx* is not equal to 0, the runtime will continue here, and execute the functionality of the for loop. We add the stored value of *ecx* to the stored value of *eax*, to mimic the functionality as in the C code of A = A + i. We then add the value of *ecx* twice to *ebx*, essentially achieving the functionality of the C code B = B + 2 * i. Finally, we decrease the counter value by 1, so that when we return to the _start main routine, the data register *ecx* will have an updated value during conditional checking to see whether it should run again.

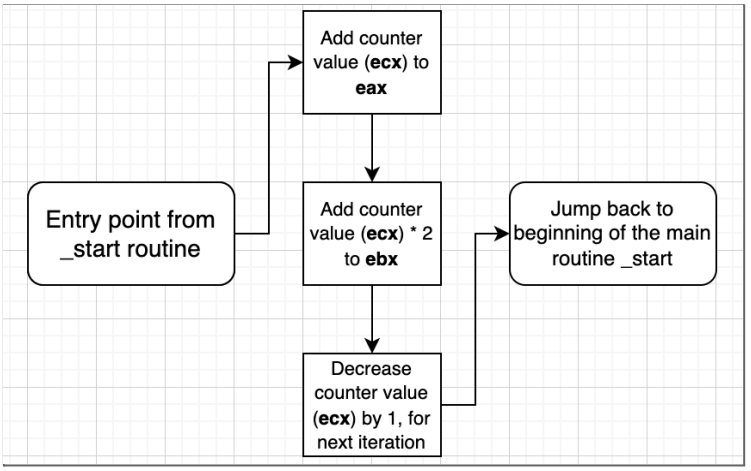

Fig 4. Flowchart for the sub-routine forLoop

Sub-routine “_exit”

The sub-routine _exit sets the *eax* data register to 1, the *ebx* data register to 0, and then using an interrupt instruction, terminates the program. The data registers are set, such that following the program running, the registers can be checked to see if the program ran successfully.

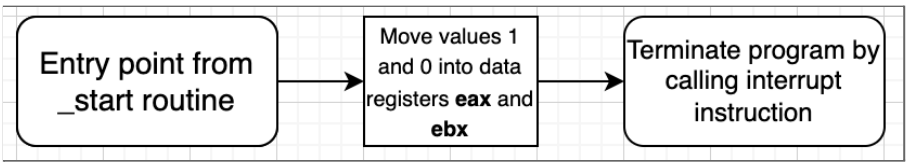

Fig 5. Flowchart for the sub-routine _exit

Memory Mapping

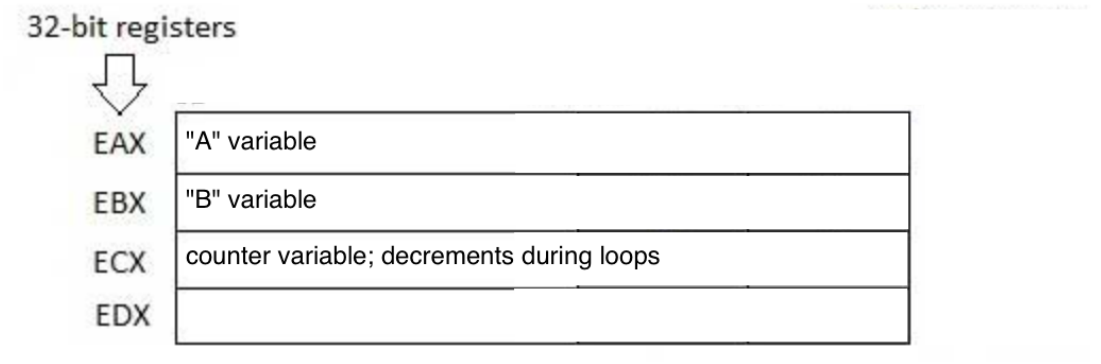

Fig 6. Flowchart for the sub-routine _exit

Code (w/ comments)

The following code is the C code converted into assembly code. A detailed explanation of each area of the code can be found below

```
section .data           ;for declaring initialised data
    A db 0              ;assigns value 0 to variable type DB (8-bits), named A
    B db 0              ;assigns value 0 to variable type DB (8-bits), named B

section .text
    global _start       ;labels _start as the starting point at runtime

_start:                 ;main routine, the entry point, known by linker due to above declaration
    mov eax, A          ;move A to eax register
    mov ebx, B          ;move B to ebx register
    mov ecx, 1          ;use ecx as i, the counter, increase this value to increase loop times

    cmp ecx, 0          ;compare if counter is equal to 0
    jg forLoop          ;call loop sub-routine if counter is not 0

    jmp _exit           ;exit program if previous condition is not met

forLoop:                ;to complete program functionality
    add eax, ecx        ;increase stored value in eax (A), by i

    add edx, ecx        ;add counter value twice to edx
    add edx, ecx
    add ebx, edx        ;add multiplied value to B

    dec ecx             ;decrement counter by 1, before returning

_exit:                  ;exit routine, called at the end of _start
    mov eax, 1
    mov ebx, 0
    int 0x80

```

Fig 7. C code converted into assembly code

Initialization

At the beginning of the assembly code, the .data section is defined wherein initialized data is declared. The two integer values of 0 are assigned to the “variables” *A* and *B*, using the data byte datatype. The .text section is also defined, wherein the _start routine is declared as the main routine of the program.

```
1  section .data           ;for declaring initialised data
2      A db 0              ;assigns value 0 to variable type DB (8-bits), named A
3      B db 0              ;assigns value 0 to variable type DB (8-bits), named B
4
5  section .text
6      global _start       ;labels _start as the starting point at runtime
7
```

Fig 8. Initialization of program, the runtime starting point

Main routine "_start"

The main routine _start is the starting point of the program. The previously declared "variables" in our .data section are moved to the registers *eax* and *ebx* here. The integer value 1 is moved to *ecx* to be used as a counter for the loop. A conditional jump *jg* (meaning Jump if Greater), checks whether the value of *ecx* (counter) is greater than 0, if it is, move to the forLoop subroutine, if it is not, jump to the subroutine _exit to exit the program.

```
8  _start:                 ;main routine, the entry point, known by linker due to above declaration
9      mov eax, A          ;move A to eax register
10     mov ebx, B          ;move B to ebx register
11     mov ecx, 1          ;use ecx as i, the counter, increase this value to increase loop times
12
13     cmp ecx, 0          ;compare if counter is equal to 0
14     jg forLoop          ;call loop sub-routine if counter is not 0
15
16     jmp _exit           ;exit program if previous condition is not met
17
```

Fig 9. "_start" the "main" routine, where the program starts and calls sub-routines

Sub-routine "forLoop"

The forLoop subroutine begins by increasing the value of *eax* by *ecx* (counter), as this achieves the same functionality as in the C code (A = A + i). It then adds the value of *ecx* twice to *ebx*, essentially achieving the functionality of the C code (B = B + 2 * i). Finally, it decreases the counter value by 1.

```
18  forLoop:                ;to complete program functionality
19      add eax, ecx        ;increase stored value in eax (A), by i
20
21      add edx, ecx        ;add counter value twice to edx
22      add edx, ecx
23      add ebx, edx        ;add multiplied value to B
24
25      dec ecx             ;decrement counter by 1, before returning
26
```

Fig 10. "forLoop" sub-routine, the converted for loop of the C code

Sub-routine “_exit”

The sub-routine _exit sets the data register *eax* to 1, which is the system exit code, and *ebx* to 0 which signifies a normal end to the program (as opposed to an error during runtime), and then calls the interrupt instruction with 0x80 as the interrupt number (the kernel). Essentially this function is equal to the C function of exit(0), it terminates the program.

```
27  _exit:                  ;exit routine, called at the end of _start
28      mov eax, 1
29      mov ebx, 0
30      int 0x80
31
```

Fig 11. “_end” is a sub-routine that is called at the end to terminate the program

Conclusions

In conclusion, by using machine-language instructions such as *mov*, *int*, *add*, *dec*, *cmp*, *jg*, *jmp*, it is possible to convert high-level C code, into the lower level programming language, assembly, which can be translated into machine level language code. Using data registers as memory, I converted variable declarations, for loop, conditional statements, from C into assembly.

Annex

Annex 1: EAX; EBX; ECX