Dalhousie University
Department of Electrical and Computer Engineering

ECED 3403 – Computer Architecture

Assignment 1: Design, implementation, and testing of a pre-assembler for the X-Makina assembler

1 Objectives

When a new machine such as the *Eagle* in *The Soul of a New Machine* is developed, software is required to allow software designers to design and write software for the machine. Computer engineers are often responsible for designing and writing such software, including tools known as *assemblers*, which translate *assembly-language* programs into *machine code*. The machine code program can then be loaded into the machine's memory for subsequent execution.

In these situations where the new machine is unable to run the assembler (for example, it is still under development), a *cross-assembler* is required.[1] The cross-assembler runs on a second machine, producing a program that can be loaded and executed on the target machine or an emulator of the target machine.

Since the assembler for the X-Makina machine has already been written, this assignment requires you to design, implement, and test a *pre-assembler*, which takes X-Makina assembly language programs with emulated instructions (see the X-Makina ISA document for details) and translates them into the equivalent X-Makina assembly code for assembly.

All software must be designed and the designs submitted before the programs are written. The designs are to be implemented in a high-level language (either C or C++) and run on the machine of your choice. All examples in class will be in C.

2 The pre-assembler

The pre-assembler is a stand-alone program which takes X-Makina assembly language programs with emulated instructions and translates them into the equivalent X-Makina assembly code for assembly. Pre-assemblers (and pre-compilers) are seldom part of an assembler (or compiler) as adding a tool such as this makes the target software more complex and potentially limits its application.[2]

The pre-assembler is to read a record from an X-Makina file, scan it for an emulated instruction, and, if found, translate it into the equivalent X-Makina instruction. X-Makina's emulated instructions are shown in Table 1.

---

[1] This is often the situation when code is required for an entirely new machine, one without an operating system, editor, or tools such as assemblers or compilers.

[2] It also violates the rule of thumb which says that a subroutine/function/program should do one thing and do it well.

Table 1: X-Makina's emulated instructions

Instruction	Emulation	Description
ADC.*x Rn*	ADDC.*x* #0,*Rn*	Add carry to Rn
CALL subr	BL subr	Call subr; Return address in LR
CLR.*x Rn*	MOV.*x* #0,*Rn*	Clear Rn
CLRC	BIC #1,R6	Clear carry bit
CLRN	BIC #4,R6	Clear negative bit
CLRZ	BIC #2,R6	Clear zero bit
DADC.*x Rn*	DADD.*x* #0,*Rn*	Decimal add carry to Rn
DEC.*x Rn*	SUB.*x* #1,*Rn*	Decrement Rn
DECD.*x Rn*	SUB.*x* #2,*Rn*	Double decrement Rn
INC.*x Rn*	ADD.*x* #1,*Rn*	Increment Rn
INCD.*x Rn*	ADD.*x* #2,*Rn*	Double increment Rn
INV.*x Rn*	XOR.*x* #−1,*Rn*	Invert Rn
JUMP *Rn*	MOV *Rn*,R7	Branch to destination (in Rn)
NOP	MOV R6,R6	No operation
PULL *Rn*	LD R5+,Rn	Stack Pull (POP) Rn
PUSH *Rn*	ST Rn,-R5	Stack Push Rn
RET	MOV R4,R7	Return from subroutine or ISR
RLC.*x Rn*	ADDC.*x Rn*,*Rn*	Rotate left Rn through carry
SBC.*x Rn*	SUBC.*x* #0,*Rn*	Subtract borrow (1−carry) from Rn
SETC	BIS #1,R6	Set carry bit
SETN	BIS #4,R6	Set negative bit
SETZ	BIS #2,R6	Set zero bit
SLA.*x Rn*	ADD.*x Rn*,*Rn*	Shift left arithmetic (shift left 1 bit) Rn; Multiply by 2
SPL0	MOVLZ #$0,R6	Set CPU priority to 0
SPL1	MOVLZ #$20,R6	Set CPU priority to 1
SPL2	MOVLZ #$40,R6	Set CPU priority to 2
SPL3	MOVLZ #$60,R6	Set CPU priority to 3
SPL4	MOVLZ #$80,R6	Set CPU priority to 4
SPL5	MOVLZ #$A0,R6	Set CPU priority to 5
SPL6	MOVLZ #$C0,R6	Set CPU priority to 6
SPL7	MOVLZ #$E0,R6	Set CPU priority to 7
TST.*x Rn*	CMP.*x* #0,*Rn*	Test Rn

For example, if a program writes an X-Makina assembly program with the instruction SETZ, the pre-assembler would replace the instruction with BIS #2,PSW:

```
Input record:     SETZ        ; Set the Z-bit
Output record:    BIS #2,PSW  ; Set the Z-bit
```

Records without an emulated instruction are left unchanged:

```
Input record:     MOV R0,R1   ; alpha = beta
Output record:    MOV R0,R1   ; alpha = beta
```

2.1 Arguments

Some emulated instructions include arguments (operands) that must be included in the output record. For example, the instruction DEC.x Rn must use the operand size, ".x", and the register, "Rn", to produce the instruction SUB.x #1,Rn. The combinations are shown in Table 2.

Table 2: Specifying the operand-size using .x and handling the register Rn

.x	Resulting instruction
Omitted	SUB #1,Rn
.W	SUB.W #1,Rn
.B	SUB.B #1,Rn

The register, Rn, is to be used as the argument in the output record; for example, in this case, "x" is "b" and "Rn" is "r2":

Input record:	dec.b r2	; r2--
Output record:	SUB.B #1,r2	; r2--

CALL also has an argument, this must be used as the operand for the BL (Branch with Link) instruction:

Input record:	CALL AddFunc	; AddFunc()
Output record:	BL AddFunc	; AddFunc()

`AddFunc` should be a Label. It is not the responsibility of the pre-assembler to determine if it exists – this is the responsibility of the assembler.

2.2 Requirements

The pre-assembler is to take an X-Makina assembler file and replace emulated instructions with their X-Makina counterparts. The pre-assembler should recognize all emulated instructions (regardless of case) and their X-Makina counterparts. If errors are detected, they should be flagged; however, pre-assembly should continue for the remaining records. The output of the pre-assembler should be free of emulator instructions and ready for the assembler.

A well-designed, implemented, and tested program written in either C or C++, meeting the above description, is required in this assignment.

2.3 Points to consider:

Here are some points to consider when implementing the pre-assembler (you are not required to submit answers to these question; however, your software should be able to handle the points raised):

- How are labels with same name as an emulated instruction handled?
- How are comments containing an emulated instruction as a name?
- Should the pre-assembler write to the supplied X-Makina assembler file or an entirely new file?

- More importantly, how are incorrect instructions or arguments, or both, handled?

3 Marking

The assignment will be graded out of 20 using the following marking scheme:

Design Document
The design document is to include an introduction as to the purpose of the software, your understanding of the problem, and a description of the algorithms and data structures required.

Total points: 6.

Software
A fully commented, indented, magic-numberless, tidy piece of software that meets the requirements described above and follows the design description.

Total points: 10.

System Testing
A set of system tests demonstrating that the software operates according to the design description. The submission must include the name of the test, its purpose or objective, the test configuration, and the test results. The designer of the software is responsible for defining and supplying the tests.[3]

Total points: 4.

Each part of the assignment must be submitted on paper.

4 Important Dates

Available: 14 May 2018

Design document submission (3pm, at start of class): 24 May 2018

Demonstrations (during lab): 5 June 2018

Software and testing submission (3pm, at start of lab): 5 June 2018

If the software deviates from the original design, the design document must be updated and resubmitted.

Late submissions for this assignment will not be accepted.

5 Miscellaneous

This assignment is to be completed individually.

Do *not* discard this work when completed, as it can be used with the remaining assignments or potentially in future courses.

---

[3] For more information on testing in general, see *Welcome to Software Testing Fundamentals* (http://softwaretestingfundamentals.com) and on system testing in particular, see System Testing (http://softwaretestingfundamentals.com/system-testing/).

This assignment is worth 10% of your overall course grade.

If you are having *any* difficulty with this assignment or the course, *please* contact Dr. Hughes or one of the TAs, Gary Hubley or Justin Lynch, as soon as possible.