Design and implement a two pass assembler for the machine described below using a language suitable for writing system software. I must be able to compile and run your program on bullwinkle.

The project must be properly documented:

**Specification** – general and technical (supplied). Include test procedures and data.

**Analysis** – detailed analysis of the specification. Complete with explanation of problems and solutions to those problems.

**Design** – functional and non-functional design documents. Include algorithms for major program segments.

**Implementation** –

Well documented code.

Test results – both successful and unsuccessful

**User Manual** – describe in detail how to use the program.

**Technical Manual** – describe any technical details that may need maintenance.

ARCHITECTURE:

1. Memory: 1024K bytes, byte addressed from 00000 to FFFFF16.

32 bit words.

2> CPU Registers:

PC -> 20 bit register holds the address of the next instruction to be executed

IR -> 32 bit register holds the current instruction

ACC -> 32 bit accumulator

X, Y -> 16 bit indexing registers

PSB -> 8 bit program status byte (s,v,b,,i,z,n,c)

S -> 20 bit stack pointer (Stack builds from high memory down).

EP -> environment pointer

INSTRUCTIONS:

1> 1 word (4 bytes) instruction

Format

8 bit opcode bits 31- 24

4 bit format/indexing code bits 23 - 20

20 bit displacement /address/data bits 19 - 0

addressing modes bits

23 22 21 20 Meaning

0 0 0 0 PC displacement mode. TA = PC + displacement *default*

0 0 0 1 X indexed TA = PC + displacement + X

0 0 1 0 Y indexed TA = (PC + displacement) + Y

0 0 1 1 direct TA = address

0 1 0 0 indirect TA = (address)

0 1 0 1 indirect/indirect TA = ((address))

0 1 1 0 immediate TA = address of the instruction + 1

( high order 4 bits are ignored)

0 1 1 1 stack relative TA = ((S) + displacement)

TA is the Effective Address – the location of the data.

1. 1 byte instructions – high order bit (7) is 1. All 1 word instructions will have a 0 in the high order bit.

2> Machine codes and mnemonics:

Flags affected

oc mnemonic effect s v b i n z c

F0 NOP no operation - - - - - - -

04 ADD ACC <- ACC + (TA) - √ - - √√√

08 SUB ACC <- ACC - (TA) - √ - - √√√

0C MUL ACC <- ACC \* (TA) - √ - - √√√

10 DIV ACC <- ACC div (TA) - √ - - √√√

14 LDA ACC <- (TA) - √ - - √√-

1C LDX X <- (TA) - - - - - - -

20 LDY Y <- (TA) - - - - - - -

24 STA (TA) <- ACC{1 word} - √ - - √√-

2C STX (TA) <- X {bit extended} - √ - - √√-

30 STY (TA) <- Y {bit extended} - √ - - √√-

50 JMP PC <- TA - - - - - - -

54 JMPSUB Stack <- PC - - - - - - -

PC <- TA √ - - - - - -

5C JMPZ If z set PC <- TA - - - - - √ -

60 JMPN If n set PC <- TA - - - - √ - -

64 JMPC If c set PC <- TA - - - - - - √

80 PUSH Stack <- ACC{1 word} √√ - - √√ -

84 TXA ACC <- X {bit extended} - √ - - √√-

88 TYA ACC <- Y {bit extended} - √ - - √√ -

8C INA ACC <- ACC + 1 - √ - - √√√

94 INY Y <- Y + 1 - √ - - √√√

98 RET PC <- Stack √ - - - - - -

A4 POP ACC <- Stack{1 word} √√ - - √√ -

A8 SHFLA ACC <- ACC \* 2 - √ - - √√√

AC SHFRA ACC <- ACC div 2 - √ - - √√√

B8 RTAD ACC <- input - √ - √ √√√

BC WRITE output <- ACC - - - √ - - -

F4 INX X <- X + 1 - √ - - √√√

F8 CLRA ACC <- 0 - - - - - √ -

FC STOP PC <- 000000 - - - - - - -

ASSEMBLY LANGUAGE INSTRUCTION FORMAT:

[label:] [opcode [operand] [comment]] | comment

label ::= letter {letter | digit}

opcode ::= mnemonic operation code | assembler directive

operand ::= label[(X | Y)] | expression | @operand | @(operand) | #operand |

%operand | $operand

comment ::= **;** character string

constant ::= digit {digit}

Note: constants are in base 10

expression ::= label | constant {operator expression}

operator ::= + | - | \*

A letter is an UPPER or lower case alphabetic character (case insensitive).

A digit is a numeric character 0..9.

**Addressing Modes**:

Default addressing mode is PC relative. (TA = PC + displacement)

@ preceding n operand means indirect addressing.

@(operand) indirect/indirect addressing.

# preceding an operand means immediate mode.

% preceding the operand means direct addressing.

(X) following the operand means X indexed addressing.

(Y) following the operand means Y indexed indirect addressing.

$ preceding the operand means stack relative addressing.

A comment may be an entire line or follow assembly code on a line.

Constants are interpreted as decimal integers.

Operators are evaluated using standard precedence.

Labels must begin in column 1. If there is no label on the line, column

1 must be either blank or contain a ;.

maximum line length = 132 characters

**Assembler directives**:

.PROG program name

label .EQU expression | **.**

[label] .WORD expression list

[label] .BYTE expression list

[label] .BLOCK expression

[label] .END [program name]

An expression list is at least one value with each succeeding value preceded by a

comma.

.PROG names the program and sets the entry point. (Generates no code.)

The entry point is stored as an address relative to 0.

.EQU equates the value of the label with expression.

'.' as the operand means use the value of the current location counter.

.WORD reserves words of memory – one word for each value listed . It initializes them

sequentially to the values in the list.

.BYTE reserves bytes of memory – one byte for each value listed. It initializes them

sequentially.

.BLOCK reserves expression bytes of memory and does not initialize them.

.END end of assembly. Branching to end label is not allowed.

NOTES:

Forward references are not supported in any assembler directives.

Integer values are stored in 2's complement notation (including displacements and symbol table values from .EQU's).

Addresses are unsigned 20 bit integers.

By convention, words are represented as 8 hexadecimal digits in printouts. Addresses are

5 hexadecimal digits. Hexadecimal values A-F are upper case.

ERRORS:

Check only for undefined or multiply defined symbols, invalid opcodes or assembler directives, and missing operands.

INPUT: Any CS4800 assembly language program. e.g.

LAB2: .WORD 1024

ARRAY: .BLOCK 40

SUM: .BYTE 0, 0, 0, 0

LAB1: .WORD 623

.PROG EXAMPLE

LDA LAB1

PUSH

START: ADD LAB2

STA SUM

LDA LAB1

STA ARRAY

MOVE: LDA LAB2

STA ARRAY + 4 ; don’t need #

LDA SUM

STA ARRAY + 8

WHEW: STOP ; be sure program exits

DONE: .END EXAMPLE

OUTPUT: LISTING FILE

EXAMPLE.ASM FEBRUARY 9, 2017 PAGE 1

LINE CODE LOC | LABEL OPCODE OPERAND

001 | ; This is an example program

002 00 00 04 00 0 00 00 | LAB2: .WORD 1024 ; “constant”

003 ?? ?? ?? ?? 0 00 04 | ARRAY: .BLOCK 40

004 00 00 00 00 0 00 36 | SUM: .BYTE 0, 0, 0, 0

005 00 00 02 6F 0 00 3A | LAB1: .WORD 623

006 0 00 3E | START: .PROG EXAMPLE

007 14 FF FF F8 0 00 3E | LDA LAB1

008 80 0 00 42 | PUSH

009 04 FF FF B7 0 00 45 | ADD LAB2

010 24 FF FF EB 0 00 49 | STA SUM

011 14 FF FF EB 0 00 4B | MOVE: LDA LAB1

012 24 FF FF B1 0 00 4F | STA ARRAY

013 14 FF FF A9 0 00 53 | LDA LAB2

014 24 FF FF AD 0 00 57 | STA ARRAY + 4 ; don’t need #

015 14 FF FF D7 0 00 5B | LDA SUM

016 24 FF FF A9 0 00 5F | STA ARRAY + 8

016 FC 0 00 63 | WHEW: STOP ; be sure program exits

017 0 00 64 | DONE: .END EXAMPLE

16 lines assembled with no errors

Symbol Table:

Label Value

ARRAY 0 00 04

DONE 0 00 64

LAB1 0 00 3A

LAB2 0 00 00

MOVE 0 00 4B

START 0 00 3E

SUM 0 00 36

WHEW 0 00 63

Notes on Listing:

Word values in the machine code listing are in high bit to low bit left to right form.

Error messages will appear under the line in which the error occurred.

A statement indicating the number of lines assembled and number of errors in assembly will follow the listing.

Page your output with headings and page numbers on each page.

Lastly, list the symbol table.

All machine code and the location counter values will be printed in hex. The line numbers will be in decimal.

If assembly is successful, generate an object code file as a text file in the following format.

OUTPUT: OBJECT CODE

**Header record:**

H <program name><program length> (hex word - low byte first)

**Text record:**

T starting address (hex - low byte first)) length in bytes (hex word - low byte first) text

(low to high) – maximum length 50 characters – words may not break across two lines

**End record:**

E program name Entry Point(hex – low to high)

Print out the object code.

Example: (spaces are for readability and will not be part of the object file)

H EXAMPLE 64000000

T 00000 04000000 00040000

T 36000 21000000 00000000 6F020000 F8FFFF14 80 B7FFFF04

T 4B000 2000000 EBFFFF14 B1FFFF24 A9FFFF14 ADFFFF24

T 5B000 0900000 D7FFFF14 A9FFFF24 FC

E EXAMPLE 3E000