NITROCOMPUTE ASSEMBLER DOCUMENTATION

TABLE OF CONTENT

LIST OF FIGURES

DESCRIPTION

The NitroCompute Assembler was built using Python programming language and converts Assembly Language into hexadecimal code fed into The NitroCompute microprocessor. When executed, an application GUI which has a menu bar, text area and title bar appears.

The menu bar contains the following drop-down menus.

- File
    - Open: Allow user to provide input from a text file
    - Exit: Close application

- Save
    - Save: Store the content in the text area into a specified file.

- Assemble
    - Assemble: Allow user to convert the assembly code into a hexadecimal code and store the output into a specified file. Its output is as well displayed below the text area.

The text area is a rectangular are with a white background where the user can type the assembly code.

INSTALLATION

This application is an executable file that requires no prior installation.

INSTRUCTION SET

There are two main categories of instructions, namely.

- ALU operations
- Miscellaneous operations

ALU OPERATION SYNTAX

Under the ALU, there are eight operations.

ADD	-	ADDITION
SUB	-	SUBTRACTION
MULT	-	MULTIPLICATION
DIV	-	DIVISION
SHL	-	LOGICAL SHIFT LEFT
SHR	-	LOGICAL SHIFT RIGHT
AND	-	LOGICAL MULTIPLICATION
OR	-	LOGICAL ADDITION

The general syntax for an ALU operation is,

[**ALU**] [**RA**],[**RB**],[**RC**]

Where:

ALU is the ALU operation

RA is the first register

RB is the second register

RC is the destination register

ADD

The desired operation to be performed is:

2 + 3 = 5

In assembly representation,

[ADD] [R1],[R2],[R3]

Where:

ADD is the operation being performed on the numbers
R1 contains the number **2**
R2 contains the number **3**
R3 is the register that will contains the result of the operation

SUB

The desired operation to be performed is:

6 - 2 = 5

In assembly representation,

[SUB] [R1],[R2],[R3]

Where:

SUB is the operation being performed on the numbers
R1 contains the number **6**
R2 contains the number **2**
R3 is the register that will contains the result of the operation

MUL

The desired operation to be performed is:

2 x 3 = 6

In assembly representation,

[MULT] [R1],[R2],[R3]

Where:

MULT is the operation being performed on the numbers
R1 contains the number **2**
R2 contains the number **3**
R3 is the register that will contains the result of the operation

DIV

The desired operation to be performed is:

4 ÷ 2 = 2

In assembly representation,

[DIV] [R1],[R2],[R3]

Where:

DIV is the operation being performed on the numbers
R1 contains the number 4
R2 contains the number 2
R3 is the register that will contains the result of the operation

SHL

The desired operation to be performed is:

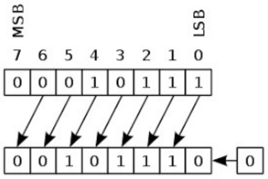

Figure 1.0

In assembly representation,

[SHL] [R1],[R2],[R3]

Where:

SHL is the operation being performed on the numbers
R1 contains the number 4 which is the value we want to do the operation on
R2 contains a number that tell the processor how many times to shift left. In this case the value will contain one to perform our required operation
R3 is the register that will contains the result of the operation

SHR

The desired operation to be performed is:

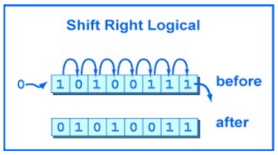

Figure 2.0

In assembly representation,

[SHR] [R1],[R2],[R3]

Where:

SHR is the operation being performed on the numbers
R1 contains the number 4 which is the value we want to do the operation on
R2 contains a number that tell the processor how many times to shift right. In this case the value will contain one to perform our required operation
R3 is the register that will contains the result of the operation

AND

The desired operation to be performed is:

2 & 3 = 2

In assembly representation,

[AND] [R1],[R2],[R3]

Where:

AND is the operation being performed on the numbers
R1 contains the number 2
R2 contains the number 3
R3 is the register that will contains the result of the operation

OR

The desired operation to be performed is:

2 OR 3 = 3

In assembly representation,

[ADD] [R1],[R2],[R3]

Where:

OR is the operation being performed on the numbers
R1 contains the number 2
R2 contains the number 3
R3 is the register that will contains the result of the operation

CISC OPERATION SYNTAX

Under the MISC, there are eight different operations.

MOVR	-	REGISTER TO REGISTER MOVE
MOVI	-	IMMEDIATE VALUE TO REGISTER MOVE
LOAD	-	PUSH CONTENT IN MEMORY TO REGISTER
STORE	-	SAVE DATA FROM REGISTER TO MEMORY
JMP	-	EXECUTE FROM SPECIFIED LOCATION IN MEMORY
JMPZ	-	EXECUTE FROM SPECIFIED LOCATION IF ZERO FLAG IS TRIGGERED
JMPN	-	EXECUTE FROM LOCATION IF NEGATIVE FLAG IS TRIGGERED
HALT/NOOP	-	STOP MICROPROCESSOR

MOVR

This operation allows us to move the contents from one register to another register.
The syntax for a MOVR operation is,

[MOVR] [RA],[RB]

Where:

MOVR is the operation being performed
RA is the **SOURCE** register
RB is the **DESTINATION** register

MOVI

This operation allows us to move an immediate value into a register
The syntax for a MOVI operation is,

[MOVI] [RA],[IMME]

Where:

MOVI is the operation being performed
RA is the **DESTINATION** register
IMME is the immediate value

LOAD

This operation allows us to load contents in memory into a register
The syntax for a MOVI operation is,

[LOAD] [RA],[ADDR]

Where:

LOAD is the operation being performed
RA is the **DESTINATION** register
ADDR is the address location of the data in memory.

STORE

This operation allows us to store the data from a register into memory.
The syntax for a STORE operation is,

[STORE] [RA],[ADDR]

Where:

STORE is the operation being performed
RA field is the **SOURCE** register
ADDR is the address location to store the value

JMP

This operation allows us to jump to a specified instruction location in memory.
The syntax for a JMP operation is,

[JMP] [ADDR]

Where:

JMP is the operation being performed
ADDR is the address location of the instruction to jump to

JMPZ

This operation allows us to jump to a specified instruction location in memory if the zero flag is enabled,
The syntax for a JMPZ operation is,

[JMPZ] [ADDR]

Where:

JMPZ is the operation being performed
ADDR is the address location of the instruction to jump to

JMPN

This operation allows us to jump to a specified instruction location in memory if the negative flag is enabled.
The syntax for a JMPN operation is,

[JMPN] [ADDR]

Where:

JMPN is the operation being performed
ADDR is the address location of the instruction to jump to

NOOP

This opcode means no operation. That is no operation is to be done.
The syntax for a NOP operation is,

[NOP]

Where:

NOP is the operation being performed

HALT

This opcode halts the microprocessor
The syntax for a HALT operation is,

[HALT]

Where:

HALT is the operation being performed

DEVELOPERS

maxotuteye (Max Otuteye) (github.com)
hanskod (Hansen Koduah) (github.com)
dzeble (github.com)
Adaks (github.com)
Dillys3567 (github.com)

APPENDIX

https://github.com/maxotuteye/nitro-compute.git

ALU	- ARITHMETIC AND LOGIC UNIT
ADD	- ADDITION
SUB	- SUBTRACTION
MULT	- MULTIPLICATION
DIV	- DIVISION
SHL	- SHIFT LEFT
SHR	- SHIFT RIGHT
AND	- LOGICAL MULTIPLICATION
OR	- LOGICAL ADDITION
RA	- REGISTER A
RB	- REGISTER B
RC	- REGISTER C
R1	- REGISTER 1
R2	- REGISTER 2
R3	- REGISTER 3
MOVR	- REGISTER TO REGISTER MOVE
MOVI	- IMMEDIATE VALUE TO REGISTER MOVE
LOAD	- PUSH CONTENT IN MEMORY TO REGISTER
STORE	- SAVE DATA FROM REGISTER TO MEMORY
JMP	- EXECUTE FROM SPECIFIED LOCATION IN MEMORY
JMPZ	- EXECUTE FROM SPECIFIED LOCATION IF ZERO FLAG IS TRIGGERED
JMPN	- EXECUTE FROM LOCATION IF NEGATIVE FLAG IS TRIGGERED
HALT	- STOP MICROPROCESSOR
NOP	- STOP MICROPROCESSOR (NO OPERATION)
IMME	- IMMEDIATE VALUE
ADDR	- ADDRESS LOCATION OF THE DATA IN MEMORY.