香港公開大學 科技學院
THE OPEN UNIVERSITY OF HONG KONG
SCHOOL OF SCIENCE AND TECHNOLOGY

ELEC S411F (2020/21)

Electronic and Computer Engineering Project

Final Report

Implementation of a new executable and linkable format for RISC-V

Project number:	A04
Student name	Ma Chi Wai
Supervisor:	Angus Wong
Submission Date:	8-3-2021

Declaration of Originality

I, Ma Chi Wai, declare that this report and the work reported herein was composed by and originated entirely from me. This report has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the reference section.

6-3-2021

Abstract

This Report is to cover the improvement for the Executable and Linkable Format(ELF). The new format for the RISC-V will be built and create a library to contain the new ELF file format. RISC-V is an open standard instruction set architecture based on established reduced instruction set computer principles. It has been used in small embedded systems, personal computer, supercomputers with vector processors and warehouse-scale 19-inch rack-mounted parallel computers. Unfortunately, RISC-V is a kind of elf file format and the data stored inside the file are byte data which mean the file are full of number and human cannot really read it.

The report contains the decoding for the ELF file format and describe the instruction of the ELF file format. Then, the implementation of creating new format for the ELF and the command set for the ELF file format will be shown.

TABLE OF CONTENTS

1. INTRODUCTION

The Executable and Linkable Format(ELF) is a common standard file format for executables, object, shared libraries and core dumps. In 1999, it was chosen as the standard binary file format for Unix and Unix-like systems. Although ELF file format does not have big issue and still working nowadays, the ELF file format is not be updated for a long time. The design of it is old and efficiency because some of the technology problem such as RAM and not suitable for any software.

The motivation for this research is to enhance the ELF file format to make it more intelligent and efficiency because the ELF file format is ignored by IT developer. The outcomes of making improvement of the ELF file format are huge, for example, the compiling time of the program will become shorter if the developer is doing a large project.

1.1 PROJECT OBJECTIVES

I. Decode the ELF format

To Read the ELF specification document and find out the structure of the file format and the value of the byte data. The decoding program will be created in java language on the Netbeans coding platform. It shows the readable content to make sure all the data can be decoded successfully.

II. Design a new Format for the ELF

Rewrite the structure of the ELF file format to reduce the issue found in the observing stage.

III. Create a library to contain the new ELF file format

Use the tools to transfers the new ELF file format into a program library to let user can call it to generate new format for the ELF file.

1.2 ORGANIZATION OF THE REPORT

This report is organized as follows:

Chapter 2 introduces the background of the Executable and Linking Format.

Chapter 3 presents the new format design approach for the Executable and Linking Format.

Chapter 4 shows the implementation details.

Chapter 5 concludes the overall of the project and identifies further works.

2. BACKGROUND

In this project, most of the document are specification document because it is a format structure. Although there is no developer to update the version of the ELF format which mean there are not many literatures could be found, most of the fundamental information are contained in a specification document such as ELF and Dwarf format. Therefore, the background information is the main core in this section.

The Executable and Linking Format(ELF) was developed and published by UNIX System Laboratories(USL) as part of the Application Binary Interface(ABI). The evolving ELF standard has been selected by the Tool Interface Standards committee(TIS) as a portable object file format. It works on 32-bit Intel Architecture environment for numbers of operation systems. By providing developers with a set of binary interface definitions which extend across servals operation system.

Relocatable, Executable and shared object are three main types of object files. A relocatable file contains data and code suitable for linking with other object files to create a shared object or executable file. A shared object file also contains code and data suitable but for linking two contexts. The link editor will process to create another object file with other shared object and relocatable file. Then, the dynamic linker combines this with and other shared objects and executable file to create a process graphic. An executable file contains a program suitable for execution. Those object files are created by assembler and link editor, they are binary representations of programs which intended to execute directly on a processor. If the programs require other abstract machines, they will be excluded.

For the file format, object files take part in program linking and execution. The object file format support parallel views of a file's contents, reflecting the differing needs of these processes.[2]

2.1 Review of Related Work

Rewrite Format

According to [13], it pointed out that C++ and C# are the extension base on the C language. C language was invented in 1972 and published in 1978 by Dennis Ritchie. C language is a low-level programming language. C++ is a language which was developed as an extension of the C language. It was created by a Ph.D. student in Denmark who called Bjarne Stroustrup. He wanted to enhance C and add object-oriented programming capabilities without sacrificing speed or efficiency. The other language C# is a high-level, object-oriented programming language(OOP). It is also an extension of C language. This language was developed by a Microsoft team lead by Anders Hejlsberg in 2002. It is based in the .NET framework but the backbone of it still clearly the C language.

Format design

There are different formats on internet for different use such as YAML, XML and JSON. Their formats are human readable which can be referenced.

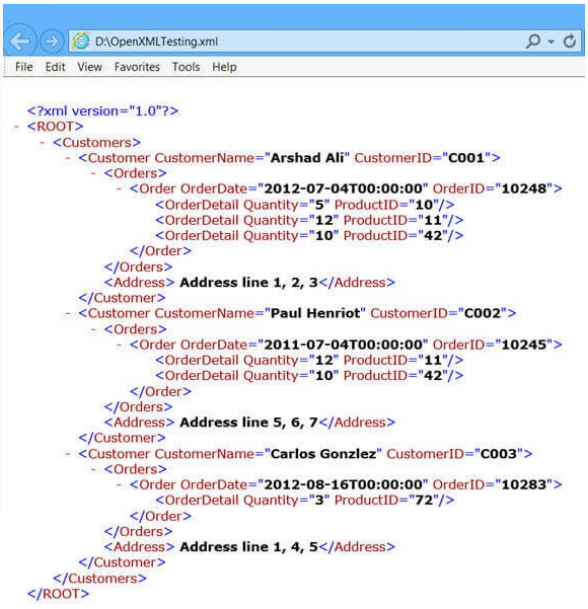


Figure2.1.1 Figure2.1.2 Figure2.1.3

Figure2.1.1 is a data format of JSON. Although there are some curly brackets in the data, you can easily understand what it means. The text of quotation mark on left side is the entity and another side is the value of the entity. The value can be an object or array and using the brackets to represent the data.

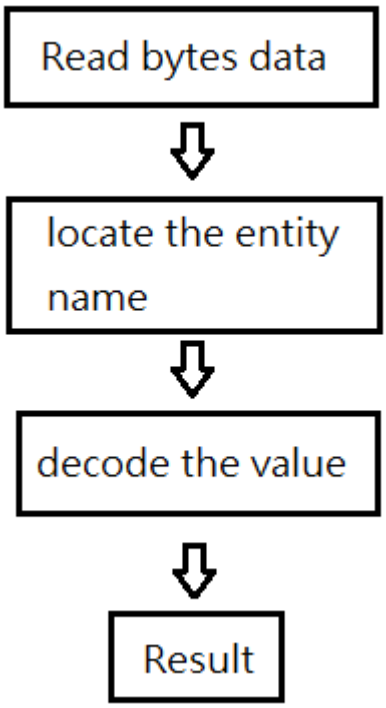
Figure2.1.2 is a data format of XML. This format uses brackets to contains the entity and use an equal symbol to represent value. Although the data of XML is using English, there are so many duplicate wordings to make it not easy to read by a human.

Figure2.1.3 is a data format of YAML. It uses a colon to separate the entity on left side and value on right side. If the value is "|" or ">", it means Newlines preserved and Newlines folded. If the right side do not have value and the next line entity have a space before the entity, which means the data is a child of it. This format is easy to watch and understand.

3. DESIGN AND METHODOLOGY

In this project, the new format for ELF is designed to make the format human readable and create command line for it. There are three curial tools, and one design are needed. They are the parser, lexer, org.apache.commons.cli and the new format design.

3.1 DECODING METHODOLOGY



(Figure3.1.1)

Figure3.1.1 is the flow of decoding. First, the bytes data of ELF file is read by program. NetBeans is used as a program platform to decode elf file structure. The hexadecimal index will add in front of each row of the bytes data to create a clean image for human reading. After that, we have to following the specification document[2], [3] and some other document [1],[4],[5]. For the coding to handle the condition for different value, I used switch function rather than if function. Compare with if function, switch function creates a clear view of case operation.

3.2 NEW FORMAT DESIGN

```
-HK
 -magic:HK
 -name:kernel
 -developer:Peter Cheung <peter@quantr.hk>
 -section
  -code
   -file_size:0x10000
   -virt_addr:0x10000
   -virt_size:0x10000
   -flag:rwx
  -code2
   -file_size:0x0
  -data
   -file_size:0x10000
   -virt_addr:0x10000
   -virt_size:@(C:\Users\Sammy\Downloads\Elearning Result.xlsx)
   -flag:@(https://www.google.com)
```

Figure 3.3.1

Figure 3.3.1 is the design of the new format. The right side of "-" symbol is label. If there is no colon next to the label, it means the label has child. Otherwise, the right side of colon is identifier. It can support array, string and object. The data type can be a URL or link to another path in the same file.

3.3 NEW FORMAT METHODOLOGY

In this section, ANTLR is a main tool to create a new format. The full name of ANTLR is Another Tool for language recognition. It is a parse generator for reading, processing, executing, or translating structured text or binary files. It can generate a parser that can build and walk parse trees.

Lexer

```
NEWLINE : [\r\n]+ ;
INT     : [0-9]+ ;
```

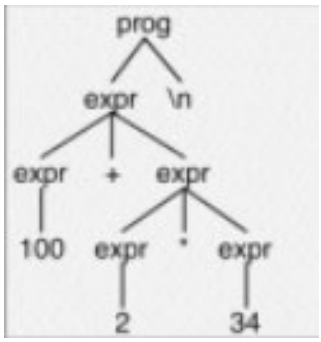
(Figure3.3.1)

The function of a lexer is taking an input character stream and convert it into tokens. Figure3.3.1 is an example of lexer. It defines the symbol '\r' and '\n' to NEWLINE and define number 0 to 9 are the INT.

Parser

```
prog:   (expr NEWLINE)*
;
expr:   expr ('*'|'/')
expr
    |   expr ('+'|'-')
expr
    |   INT
    |   '(' expr ')'
    ;
```

(Figure3.3.2)



(Figure3.3.3)

The function of parser is to determine the flow of the program. If we input 100+2*34 to the system. It will have 2 new lines because it has 2 expr which are '+' and '*' . Figure3.3.3 is the result of output.

3.4 COMMAND SET DESIGN

```
java -jar target\hk-data-format.jar 1.hk -c print /MYDATA/header/section1/offset

java -jar target\hk-data-format.jar 1.hk -c print section1/offset

java -jar target\hk-data-format.jar 1.hk -c copy /MYDATA/header/section1 /MYDATA/header/section3
```

(Figure4.3.1)

There are 4 commands for this new file format. They are print, copy, move and delete. Figure4.3.1 are the structure of the command. The detail shown as below.

The first part 'java -jar target\hk-data-format.jar'. It is to call the new elf file format library.
The second part '1.hk' is the file name which you want to config.
The third part '-c' is to call the command line.
The fourth part 'print' / 'copy' is to select the function of command line you want to use.
The fifth part is the path of the file. If the command requires two path, you have to new a space and type another file path.

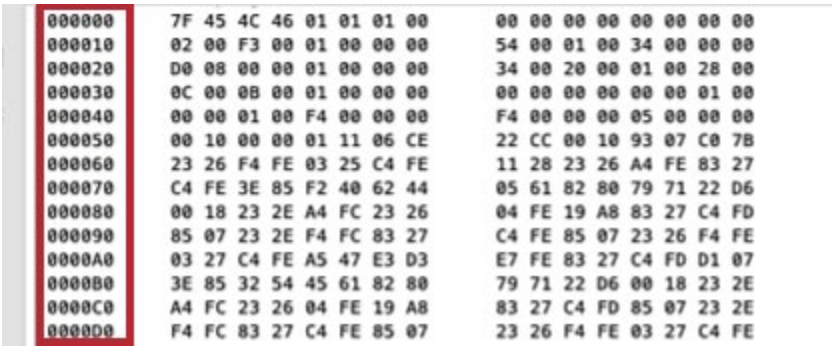
4. IMPLEMENTATION

There are 3 sections of implementation. The first section is to understand the ELF format and the decode method will be shown. The second section show how to use the tool to build up a new format and let the device understand. The final section is to show the step of creating a command line for the new format which allow user to config the file.

4.1 DECODE THE ELF AND DWARF

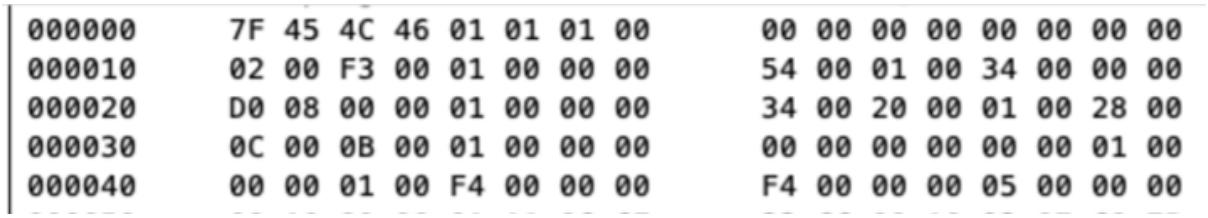
ELF Header

At the beginning, we need to use the FileInputStream function in java to read the byte data of the ELF file first. Then, the hexadecimal need to add in front of each row to create a clear address information for decoding. The result shows as below.



(figure4.1.1)

Then, we need to follow the document[2] to start decoding.



(figure4.1.2)

Figure4.1.2 is the byte data of the ELF Header.

The first 4 byte are fixed. 0x7F followed by ELF (45 4c 46) in ASCII and they constitute the magic number.

The fourth byte represent the bit format. 1 is 32-bit format and 2 is 64-bit format.

The fifth byte is set to 1 or 2 to signify big endianness or little respectively and it will affect interpretation of multi-byte fields starting with offset 0x10.
The sixth byte set to 1 for the original and current version of ELF.
The seventh byte identifies the operating system ABI of target. The representing value show as below.

ABI	System V	HP-UX	NetBSD	Linux	GNU Hurd	Solaris	AIX	IRIX	FreeBSD
Value	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08

ABI	Tru64	Novell Modesto	OpenBSD	OpenVMS	NonStop Kernel	AROS	FenixOS
Value	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F	0x10
ABI	CloudABI	Stratus Technologies OpenVOS					
Value	0x11	0x12					

The eighth byte specifies the ABI version and the interpretation of it depends on the target ABI.
The ninth and following 6 bytes are not currently use and they should be filled with zeros.
The sixteenth and seventeenth byte identifies object file type, and the representing value are shown as below.

Type	ET_NONE	ET_REL	ET_EXEC	ET_DYN	ET_CORE	ET_LOOS
Value	0x00	0x01	0x02	0x03	0x04	0xFE00
Type	ET_HIOS	ET_LOPROC	ET_HIPROC			
Value	0xFEFF	0xFF00	0xFFFF			

The eighteenth and nineteenth byte specifies target instruction set architecture. The value of RISC-V is 0xF3.

The twentieth and following 3 bytes set 1 represent the original version of ELF.
The twenty-fourth is the memory address of the entry point from where the process starts executing.

The twenty-eighth and following 3 bytes are the combining address of the program header table.

The thirty-second and following 3 bytes are the address of the section header table.

The thirty-sixth and following 3 bytes are the flags and it represents interpretation of this field depends on the target architecture.

The fortieth and fortieth-first bytes are the size of the ELF header. Usually 52 bytes for 32-bit format and 64 bytes for 64-bit format.

The fortieth-second and fortieth-third bytes are the size of a program header table entry.

The fortieth-fourth and fortieth-fifth bytes are the number of entries in the program header table.

The fortieth-sixth and fortieth-seventh bytes are the size of a section header table.

The fortieth-eighth and fortieth-ninth bytes are the number if entries in the section header table.

The fifty and fifty-first are the index of the section header table entry where stores the section names.

The fifty-second byte is the end of the ELF Header.

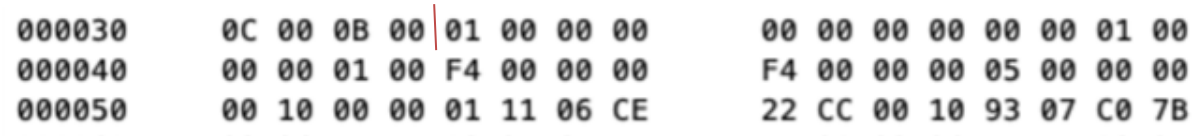
```
Program Header:
 Magic:   7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00
 Class:                             ELF32
 Data:                              2's complement, Little endian
 Version:                           1 (current)
 OS/ABI:                            UNIX - System V
 ABI Version:                       0
 Type:                              EXEC (Executable file)
 Machine::                          RISC-V
 Version:                           0x1
 Entry point address:               0x0010054
 Start of program headers:          52 (bytes into file)
 Start of section headers:          2256 (bytes into file)
 Flags:                             0x1, RVC, soft-float ABI
 Size of this headers:              52 (bytes)
 Size of program headers:           32 (bytes)
 Number of program headers:         1
 Size of section headers:           40 (bytes)
 Number of section headers:         12
 Section header string table index: 11
```

(figure4.1.3)

Figure4.1.3 are the ELF header in output of the decoding program and the decoded information of figure4.1.1.

Program Header

To decode the program header, the program header information in figure4.1.3 is needed.



(figure4.1.4)

Figure4.1.4 is the range of program header. Based on the result before. The program starts on 52 bytes which is 34 in hexadecimal index. The size of program headers are 32 bytes and there is 1 program header only.

The first 4 bytes is to identify the type of segment. The representing value shown as below.

Name	PT_NULL	PT_LOAD	PT_DYNAMIC	PT_INTERP
Value	0x00000000	0x00000001	0x00000002	0x00000003
Name	PT_NOTE	PT_SHLIB	PT_PHDR	PT_TLS
Value	0x00000004	0x00000005	0x00000006	0x00000007
Name	PT_LOOS	PT_HIOS	PT_LOPROC	PT_HIPROC
Value	0x60000000	0x6FFFFFFF	0x70000000	0x7FFFFFFF

PT_NULL means no using of program header table entry.

PT_LOAD means loadable segment.

PT_DYNAMIC means dynamic linking information.

PT_INTERP means interpreter information.

PT_NOTE means Auxiliary information.

PT_SHLIB means reserved.

PT_PHDR means segment having program header itself.

PT_TLS means thread-Local Storage template

PT_LOOS, PT_HIOS, PT_LOPROC and PT_HIPROC are an inclusive reserved ranges for operation systems specific semantics.

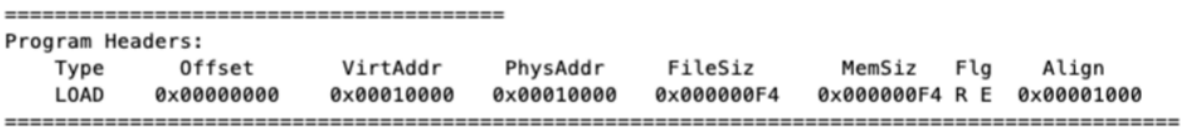
The fourth and following 3 bytes are the offset of the segment in the file image.

The eighth and following 3 bytes are the virtual address of the segment in memory.

The twelfth and following 3 bytes are where physical address is relevant, reserves for segment's physical address on systems.

The sixteenth and following 3 bytes are the size in bytes of the segment in the file image.

The twentieth and following 3 bytes are the size in bytes of the segment in memory.
The twenty-fourth and following 3 bytes are segment-dependent flags.
The twenty-eighth and following 3 bytes are representing alignment. 0 and 1 means no alignment. Otherwise, the value should be a positive, integral power of 2.
If the following bytes reach the end of the program header, it ends. Otherwise, repeat the decoding.



```
========================================
Program Headers:
    Type      Offset       VirtAddr     PhysAddr     FileSiz       MemSiz   Flg    Align
    LOAD    0x00000000    0x00010000   0x00010000   0x000000F4   0x000000F4 R E  0x00001000
========================================================================================================
```

(figure4.1.5)

Figure4.1.5 are the program header in output of the decoding program.

Section Header

To decode the program header, the program header information in figure4.1.3 is needed.

```
0008C0   72 61 6D 65 00 2E 64 65   62 75 67 5F 73 74 72 00
0008D0   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0008E0   00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
0008F0   00 00 00 00 00 00 00 00   1B 00 00 00 01 00 00 00
000900   06 00 00 00 54 00 01 00   54 00 00 00 A0 00 00 00
000910   00 00 00 00 00 00 00 00   02 00 00 00 00 00 00 00
000920   21 00 00 00 01 00 00 00   30 00 00 00 00 00 00 00
000930   F4 00 00 00 33 00 00 00   00 00 00 00 00 00 00 00
000940   01 00 00 00 01 00 00 00   2A 00 00 00 01 00 00 00
000950   00 00 00 00 00 00 00 00   27 01 00 00 60 00 00 00
000960   00 00 00 00 00 00 00 00   01 00 00 00 00 00 00 00
000970   39 00 00 00 01 00 00 00   00 00 00 00 00 00 00 00
000980   87 01 00 00 44 01 00 00   00 00 00 00 00 00 00 00
000990   01 00 00 00 00 00 00 00   45 00 00 00 01 00 00 00
0009A0   00 00 00 00 00 00 00 00   CB 02 00 00 3A 01 00 00
0009B0   00 00 00 00 00 00 00 00   01 00 00 00 00 00 00 00
0009C0   53 00 00 00 01 00 00 00   00 00 00 00 00 00 00 00
0009D0   05 04 00 00 47 01 00 00   00 00 00 00 00 00 00 00
0009E0   01 00 00 00 00 00 00 00   5F 00 00 00 01 00 00 00
0009F0   00 00 00 00 00 00 00 00   4C 05 00 00 A0 00 00 00
000A00   00 00 00 00 00 00 00 00   04 00 00 00 00 00 00 00
000A10   6C 00 00 00 01 00 00 00   30 00 00 00 00 00 00 00
000A20   EC 05 00 00 85 00 00 00   00 00 00 00 00 00 00 00
000A30   01 00 00 00 01 00 00 00   01 00 00 00 02 00 00 00
000A40   00 00 00 00 00 00 00 00   74 06 00 00 70 01 00 00
000A50   0A 00 00 00 0C 00 00 00   04 00 00 00 10 00 00 00
000A60   09 00 00 00 03 00 00 00   00 00 00 00 00 00 00 00
000A70   E4 07 00 00 75 00 00 00   00 00 00 00 00 00 00 00
```

(Figure 4.1.6)

Figure4.1.6 is the range of program header. Based on the result before. The section starts on 2256 bytes which is 8D0 in hexadecimal index. The size of section headers are 40 bytes, and the number of section headers are 12. The section header string table index is 11.

The first 4 bytes are the section name offset to a string in the .shstrtab section.

The fourth and following 3 bytes are the type of this header. Some values shown below.

Name	SHT_NULL	SHT_PROGBITS	SHT_SYMTAB	SHT_STRTAB
Meaning	Unused	Program data	Symbol table	String table
Value	0x0	0x1	0x2	0x3
Name	SHT_RELA	SHT_HASH	SHT_DYNAMIC	SHT_NOTE
Meaning	Relocation entries	Symbol hash table	Dynamic linking information	Notes

Value	0x4	0x5	0x6	0x7
Name	SHT_NOBITS	SHT_REL	SHT_SHLIB	SHT_DYNSYM
Meaning	Program space with no data	Relocation entries, no addends	Reserved	Dynamic linker symbol
Value	0x8	0x9	0x0A	0x0B
Name	SHT_INIT_ARRAY	SHT_FINI_ARRAY	SHT_GROUP	…
Meaning	Array of constructors	Array of destructors	Array of pre-constructors	…
Value	0x0C	0x0D	0x0E	…

The eighth and following 3 bytes are the attributes of the section. Some values shown below.

Name	SHF_WRITE	SHF_ALLOC	SHF_EXECINSTR	SHF_MERGE
Meaning	Writable	Occupies memory during execution	Executable	Might be merged
Value	0x1	0x2	0x4	0x10
Name	SHF_STRINGS	SHF_INFO_LINK	SHF_LINK_ORDER	SHF_OS_NONCONFORMING
Meaning	Contains null-terminated strings	‘sh_info’ contains SHT index	Preserve order after combing	Non-standard OS specific handling required
Value	0x20	0x40	0x80	0x100

Name	SHF_GROUP	SHF_TLS	SHF_MASKOS	…
Meaning	Section is member of a group	Section holds thread-local data	OS-specific	…
Value	0x200	0x400	0x0FF00000	…

The twelfth and following 3 bytes are the virtual address of the section in memory and it is for loaded sections.

The sixteenth and following 3 bytes are the offset of the section in the file image.

The twentieth and following 3 bytes are the size in bytes of the section in file image.

The twenty-fourth and following 3 bytes have the section index of related section. The purpose of it is depended on the type of section.

The twenty-eighth and following 3 bytes have the additional information of the section. The purpose of it is depended on the type of section.

The thirty-second and following 3 bytes are the required alignment of the section. The value has to be a power of two.

The thirty-sixth and following 3 bytes are the size in bytes of each entry for different sections that contain fixed-size entries. The value can be zero if nothing.

If the following bytes reach the end of the section header, it ends. Otherwise, repeat the decoding.

```
================================================================================================
Section Headers:
    [Nr] Name                      Type            Addr     Off      Size       ES  Flg  LK  Inf Al
    [ 0]                           Null            00000000 00000000 00000000   00         0    0  0
    [ 1] .text                     PROGBITS        00010054 00000054 000000A0   00   AX    0    0  2
    [ 2] .comment                  PROGBITS        00000000 000000F4 00000033   01   MS    0    0  1
    [ 3] .debug_aranges            PROGBITS        00000000 00000127 00000060   00         0    0  1
    [ 4] .debug_info               PROGBITS        00000000 00000187 00000144   00         0    0  1
    [ 5] .debug_abbrev             PROGBITS        00000000 000002CB 0000013A   00         0    0  1
    [ 6] .debug_line               PROGBITS        00000000 00000405 00000147   00         0    0  1
    [ 7] .debug_frame              PROGBITS        00000000 0000054C 000000A0   00         0    0  4
    [ 8] .debug_str                PROGBITS        00000000 000005EC 00000085   01   MS    0    0  1
    [ 9] .symtab                   SYMTAB          00000000 00000674 00000170   10        10   12  4
    [10] .strtab                   STRTAB          00000000 000007E4 00000075   00         0    0  1
    [11] .shstrtab                 STRTAB          00000000 00000859 00000077   00         0    0  1
================================================================
```

(Figure4.1.7)

Figure4.1.7 are the section header in output of the decoding program.

.Debug_line

To decode the program header, the program header information in figure4.1.7 is needed.

```
000400  03 08 00 00 00 53 00    00 00 03 00 1D 00 00 00 01
000410  01 FB 0E 0D 00 01 01    01 01 00 00 00 01 00 00 01
000420  00 6D 61 69 6E 2E 63    00 00 00 00 00 05 0B 00 05
000430  02 54 00 01 00 15 05    06 03 01 09 08 00 01 05 04
000440  03 01 09 08 00 01 05    09 03 01 09 0A 00 01 05 01
000450  03 01 09 04 00 01 09    0A 00 00 01 01 74 00 00 00
000460  03 00 1A 00 00 00 01    01 FB 0E 0D 00 01 01 01 01
000470  00 00 00 01 00 00 01    00 61 2E 63 00 00 00 00 00
000480  05 10 00 05 02 7C 00    01 00 01 05 0B 03 02 09 0A
000490  00 01 05 02 03 00 09    04 00 01 05 04 00 02 04 03
0004A0  03 01 09 02 00 01 05    15 00 02 04 03 03 7F 09 0A
0004B0  00 01 05 02 00 02 04    01 03 00 09 0A 00 01 05 0A
0004C0  03 03 09 0A 00 01 05    01 03 01 09 06 00 01 09 08
0004D0  00 00 01 01 74 00 00    00 03 00 1A 00 00 00 01 01
0004E0  FB 0E 0D 00 01 01 01    01 00 00 00 01 00 00 01 00
0004F0  62 2E 63 00 00 00 00    00 05 12 00 05 02 B8 00 01
000500  00 01 05 0B 03 02 09    0A 00 01 05 02 03 00 09 04
000510  00 01 05 05 00 02 04    03 03 01 09 02 00 01 05 18
000520  00 02 04 03 03 7F 09    0A 00 01 05 02 00 02 04 01
000530  03 00 09 0A 00 01 05    0B 03 03 09 0A 00 01 05 01
000540  03 01 09 06 00 01 09    08 00 00 01 01 0C 00 00 00
```

(Figure4.1.8)

Figure4.1.8 is the range of .debug_line. Based on the result before. The offset of .debug_line is 405 and the size of it is 147 in hexadecimal.

The first 4 bytes are the length.

The fourth and fifth bytes are the version of dwarf.

The sixth and following bytes are the prologue length.

The eighth byte is the minimum instruction length.

The ninth byte is the initial value of ‘is_stmt’.

The tenth byte is the line base.

The eleventh byte is the line range.

The twelfth byte if the opcode base.

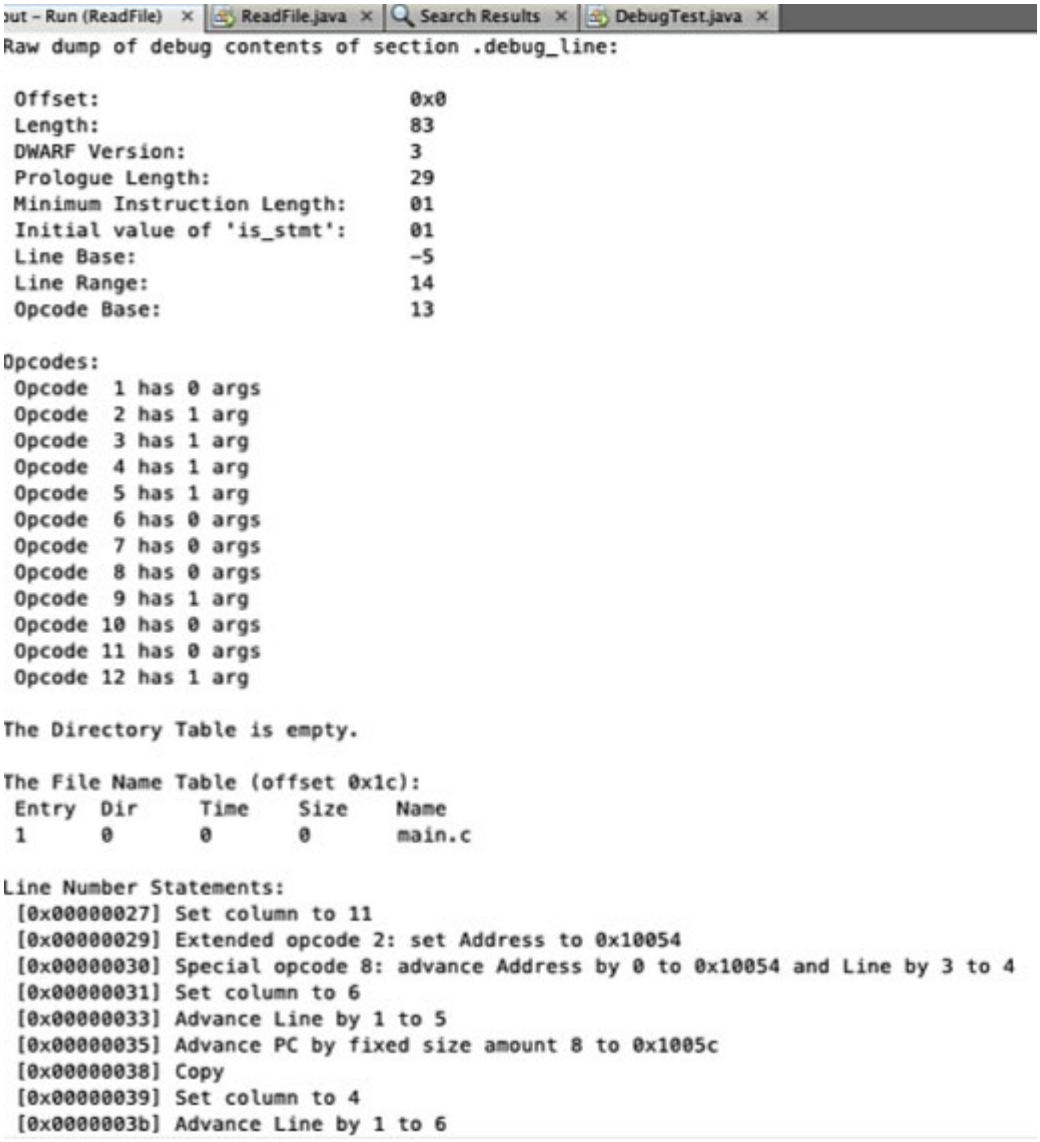
The Address and Line calculation:

Address += ((Opcode - Opcode base) / Line range) * Min instruction length

Line += Line base + (Opcode - Opcode base) % Line range

In the following byte, the bytes after that depend on the opcode. The detail shown as below.

Opcode Name	**Value**
DW_LNS_copy	0x01
DW_LNS_advance_pc	0x02
DW_LNS_advance_line	0x03
DW_LNS_set_file	0x04
DW_LNS_set_column	0x05
DW_LNS_negate_stmt	0x06
DW_LNS_set_basic_block	0x07
DW_LNS_const_add_pc	0x08
DW_LNS_fixed_advance_pc	0x09
DW_LNS_set_prologue_end	0x0A
DW_LNS_set_epilogue_begin	0x0B
DW_LNS_set_isa	0x0C



(Figure4.1.9)

Figure4.1.9 are the .debug_line in output of the decoding program.

4.2 NEW FORMAT

Source Settings History

```
1  lexer grammar HKDataFormatLexer;
2
3  WS          :   (' '|'\t')+ → channel(1);
4  NL          :   '\r'? '\n' → skip;
5
6  DASH        :   '-';
7  COLON       :   ':';
8  IDENTIFIER  :   [a-zA-Z0-9 _<>@\\.:@()/]+;
9
10
11
```

(Figure4.2.1)

Figure4.2.1 is the design of lexer. The definitions are below.

WS means the space. ‘ ’ and ‘\t’ are the representing input.
NL means next line. ‘\r’ and ‘\n’ are the representing input.
DASH means the ‘-’ symbol.
COLON means the ‘:’ symbol.
IDENTIFIER mean string. It can be character a to z, A to Z, number 0 to 9 and symbol.

```
1  parser grammar HKDataFormatParser;
2  options { tokenVocab=HKDataFormatLexer; }
3
4  hk          :   lines EOF
5              ;
6
7  lines       :   line*
8              ;
9
10 line        :   WS? DASH name=IDENTIFIER (COLON attribute=IDENTIFIER)? #handleLine
11             ;
12
13
```

(Figure4.2.2)

Figure4.2.2 is the design of parser. The definitions are below.

First, the program will read lines and EOF means read until the end of the file. Then it will read the line per row. Finally, it will do a checking of token. The symbol of ‘?’ meaning the token is not a compulsory.

5. COMMAND LINE RESULTS

The command line for print, copy, move and delete are develop as below.

5.1 PRINT FUNCTION

```
C:\Users\Sammy\Desktop\hk-data-format>java -jar target\HKDataFormat-1.0.jar 1.hk -c print /HK/section/code/
================================
Function : Print
================================
The value of (/HK/section/code/):
-code
 -file_size:0x10000
 -virt_addr:0x10000
 -virt_size:0x10000
 -flag:rwx
================================
```

(Figure5.1.1)

Typing java -jar target\HKDataFormat-1.0.jar 1.hk -c print /HK/section/code/ will return the value of the path, /HK/section/code/, to you as Figure5.1.1.

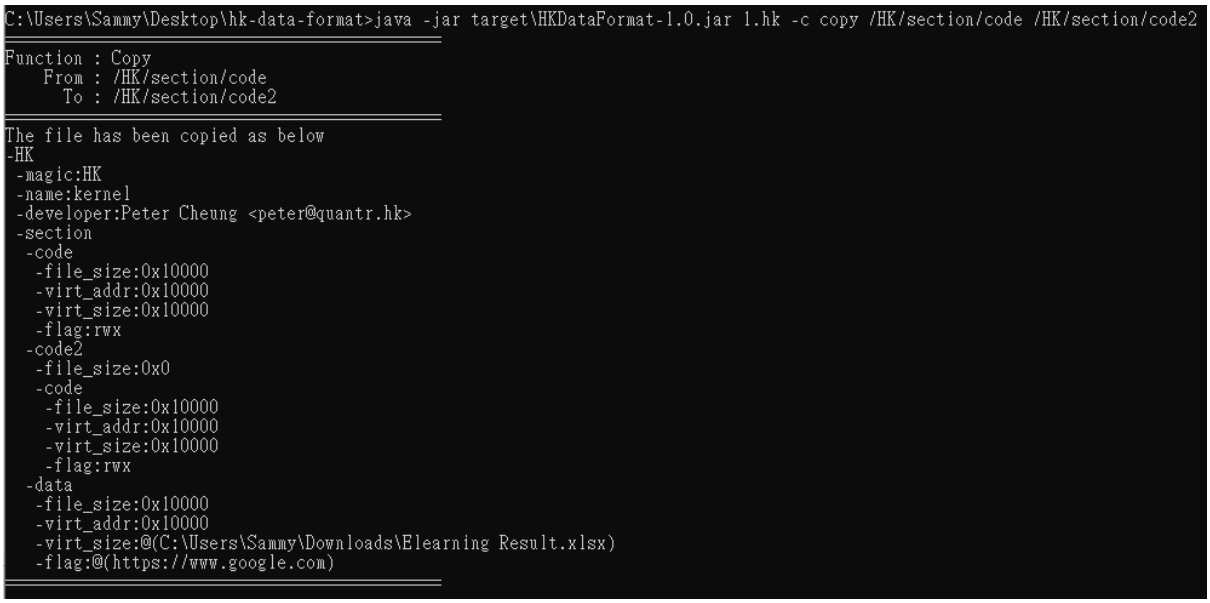
5.2 DELETE FUNCTION

```
C:\Users\Sammy\Desktop\hk-data-format>java -jar target\HKDataFormat-1.0.jar 1.hk -c delete /HK/section/code
========================================
Function : Delete
========================================
/HK/section/code
The file has been deleted as below
-HK
 -magic:HK
 -name:kernel
 -developer:Peter Cheung <peter@quantr.hk>
 -section
  -code2
   -file_size:0x0
  -data
   -file_size:0x10000
   -virt_addr:0x10000
   -virt_size:@(C:\Users\Sammy\Downloads\Elearning Result.xlsx)
   -flag:@(https://www.google.com)
========================================
```

(Figure5.4.1)

Typing java -jar target\HKDataFormat-1.0.jar 1.hk -c delete /HK/section/code/ will delete the value of the path, /HK/section/code/, to you as Figure5.4.1.

5.3 COPY FUNCTION



(Figure5.2.1)

Typing java -jar target\HKDataFormat-1.0.jar 1.hk -c copy /HK/section/code/ /HK/section/code2/ will copy the value of the path '/HK/section/code/' to '/HK/section/code2/' as Figure5.2.1.

5.3 MOVE FUNCTION

(Figure5.3.1)

Typing java -jar target\HKDataFormat-1.0.jar 1.hk -c move /HK/section/code/ /HK/section/code2/ will copy the value of the path '/HK/section/code/' to '/HK/section/code2/' as Figure5.3.1.

6. CONCLUSION AND FURTHER WORK

In this project, I created a program to decode the main tasks of elf format which are elf header, program header and section. The program turns the bytes data of the elf file into a human readable character.

The new format for RISC-V of elf has been design, built and the command set for this format also has been created. The user can now config the elf file like addressing in the new format.

For the further work, there are serval items can be applying. For example:

- creating a more user-friendly for command line interface such as create an alert to ensure user really want to use the function.
- Allowing to encrypt the data.
- Supporting data type of array, string or object.

https://gitlab.com/quantr/toolchain/hk-data-format is the URL of the project. You can get more information of the project with it.

REFERENCES

[1] "Executable and Linkable Format," *Wikipedia*, 03-Mar-2021. [Online]. Available: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format. [Accessed: 07-Mar-2021].

[2] "Tool Interface Standard (TIS) Executable and Linking ..." [Online]. Available: https://refspecs.linuxbase.org/elf/elf.pdf. [Accessed: 07-Mar-2021].

[3] "DWARF Debugging Information Format V4." [Online]. Available: http://dwarfstd.org/doc/DWARF4.pdf. [Accessed: 12-Jun-2021].

[4] "DWARF," *DWARF - OSDev Wiki*. [Online]. Available: https://wiki.osdev.org/DWARF. [Accessed: 07-Mar-2021].

[5] "DWARF," *Wikipedia*, 28-Aug-2020. [Online]. Available: https://en.wikipedia.org/wiki/DWARF. [Accessed: 07-Mar-2021].

[6] T. Parr, *Language Implementation Patterns Create Your Own Domain-specific and General Programming Languages*. Erscheinungsort nicht ermittelbar: Oreilly & Associates Inc, 2018.

[7] T. Parr, *The definitive ANTLR 4 reference*. Dallas, TX: The Pragmatic Bookshelf, 2013.

[8] "NetBeans," *Wikipedia*, 02-Mar-2021. [Online]. Available: https://en.wikipedia.org/wiki/NetBeans. [Accessed: 07-Mar-2021].

[9] S. W. Soltero, *Dual language education: program design and implementation*. Portsmouth, NH: Heinemann, 2016.

[10] *What is a Programming Language?*, 09-Jun-2020. [Online]. Available: https://www.computerhope.com/jargon/p/programming-language.htm. [Accessed: 07-Mar-2021].

[11] G. Berry and G. Gonthier, "The Esterel synchronous programming language: design, semantics, implementation," *Science of Computer Programming*, 26-Mar-2002.

[Online]. Available: https://www.sciencedirect.com/science/article/pii/016764239290005V. [Accessed: 07-Mar-2021].

[12] “YAML Syntax¶,” *YAML Syntax - Ansible Documentation*, 18-Feb-2021. [Online]. Available: https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html. [Accessed: 07-Mar-2021].

[13] J. Friedman, “Understanding the Differences Between C#, C++, and C,” *C# Station*, 28-Jun-2018. [Online]. Available: https://csharp-station.com/understanding-the-differences-between-c-c-and-c/. [Accessed: 08-Mar-2021].

APPENDIX A – CODE FOR DECODING

```
package com.mycompany.readfile;


import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;


/**
 *
 * @author sammyma
 */
public class ReadFile {

    public static void main(String[] args){

        //GD32VF103C_START.elf
        ///Users/sammyma/Desktop/main

        ///Users/sammyma/NetBeansProjects/quantr-executable-library/out.bin


      //    File file = new
File("/Users/sammyma/NetBeansProjects/quantr-executable-library/out.bin");

        //File file = new File("/Users/sammyma/Desktop/main");
          File file = new File("/Users/sammyma/Desktop/GD32VF103C_START.elf");

        FileInputStream fin = null;
        try {
            // create FileInputStream object
            fin = new FileInputStream(file);

            byte fileContent[] = new byte[(int)file.length()];

            // Reads up to certain bytes of data from this input stream into an array of
bytes.
            fin.read(fileContent);

            int i = 0;

                //Elf header Magic//

                    System.out.println("Program Header:");
                    System.out.printf(" Magic:   %02X %02X %02X %02X %02X
%02X %02X %02X %02X %02X %02X %02X %02X %02X %02X %02X "
```

```
                                , fileContent[i], fileContent[i+1], fileContent[i+2],
fileContent[i+3], fileContent[i+4], fileContent[i+5]
                                , fileContent[i+6], fileContent[i+7], fileContent[i+8],
fileContent[i+9], fileContent[i+10], fileContent[i+11]
                                , fileContent[i+12], fileContent[i+13], fileContent[i+14],
fileContent[i+15]);
                        System.out.println();
                        //Elf header Class//
                            if (fileContent[i+4] == 0x1){
                                System.out.println(" Class:
ELF32");
                            }else {
                                System.out.println(" Class:
ELF64");
                            }
                        //Elf header Data//
                            switch (fileContent[i+5]){
                                case 0x00:
                                    System.out.println(" Data:
Invalid data encoding");
                                    break;
                                case 0x01:
                                    System.out.println(" Data:
2's complement, Little endian");
                                    break;
                                case 0x02:
                                    System.out.println(" Data:
2's complement, Little endian");
                                    break;
                            }
                        //Elf header Version//
                            if (fileContent[i+6] == 0x01){
                                System.out.println(" Version:
1 (current)");
                            }else {
                                System.out.println(" Version:
0 (invalid)");
                            }

                        //Elf header OS/ABI//
                            switch (fileContent[i+7]){
                                case 0x00:
                                    System.out.println(" OS/ABI:
UNIX - System V");
                                    break;
                                case 0x01:
                                    System.out.println(" OS/ABI:
UNIX - HP-UX");
                                    break;
                                case 0x02:
```

```
                    System.out.println(" OS/ABI:
UNIX - NetBSD");
                    break;
                case 0x03:
                    System.out.println(" OS/ABI:
UNIX - Linux");
                    break;
                case 0x04:
                    System.out.println(" OS/ABI:
UNIX - GNU Hurd");
                    break;
                case 0x06:
                    System.out.println(" OS/ABI:
UNIX - Solaris");
                    break;
                case 0x07:
                    System.out.println(" OS/ABI:
UNIX - AIX");
                    break;
                case 0x08:
                    System.out.println(" OS/ABI:
UNIX - IRIX");
                    break;
                case 0x09:
                    System.out.println(" OS/ABI:
UNIX - FreeBSD");
                    break;
                case 0x0A:
                    System.out.println(" OS/ABI:
UNIX - Tru64");
                    break;
                case 0x0B:
                    System.out.println(" OS/ABI:
UNIX - Novell Modesto");
                    break;
                case 0x0C:
                    System.out.println(" OS/ABI:
UNIX - OpenBSD");
                    break;
                case 0x0D:
                    System.out.println(" OS/ABI:
UNIX - OpenVMS");
                    break;
                case 0x0E:
                    System.out.println(" OS/ABI:
UNIX - NonStop Kernel");
                    break;
                case 0x0F:
                    System.out.println(" OS/ABI:
UNIX - AROS");
```

```
                                        break;
                                    case 0x10:
                                        System.out.println(" OS/ABI:
UNIX - Fenix OS");
                                        break;
                                    case 0x11:
                                        System.out.println(" OS/ABI:
UNIX - CloudABI");
                                        break;
                                    case 0x12:
                                        System.out.println(" OS/ABI:
UNIX - Stratus Technologies OpenVOS");
                                        break;



                            }
                        //Elf header ABI Version//
                            if (fileContent[i+8] == 00){
                                System.out.println(" ABI Version:
0");
                            }
                        //Elf header Type//


                        int i2 = i+16;
                                String T1 =
String.format("%8s",Integer.toBinaryString(fileContent[i2] & 0xFF)).replace(' ','0');
                                String T2 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+1] & 0xFF)).replace(' ','0');

                                String T= T2+T1;
                                int Result_T = Integer.parseInt(T, 2);
                                switch (Result_T) {
                                    case 0x00:
                                        System.out.println(" Type:
NONE (No file type)");
                                        break;
                                    case 0x01:
                                        System.out.println(" Type:
REL (Rekocatable file)");
                                        break;
                                    case 0x02:
                                        System.out.println(" Type:
EXEC (Executable file)");
                                        break;
                                    case 0x03:
                                        System.out.println(" Type:
DYN (Shared object file)");
                                        break;
```

```
                                        case 0x04:
                                                System.out.println(" Type:
CORE (Core file)");
                                                break;
                                        case 0xFE00:
                                                System.out.println(" Type:
CORE (Core file)");
                                                break;
                                        case 0xFEFF:
                                                System.out.println(" Type:
CORE (Core file)");
                                                break;
                                        case 0xFF00:
                                                System.out.println(" Type:
CORE (Core file)");
                                                break;
                                        case 0xFFFF:
                                                System.out.println(" Type:
CORE (Core file)");
                                                break;

                                }

                        //Elf header Machine//
                        String M1 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+2] & 0xFF)).replace(' ','0');
                        String M2 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+3] & 0xFF)).replace(' ','0');

                        String M= M2+M1;
                        int Result_M = Integer.parseInt(M, 2);
                        switch (Result_M){
                                case 0x00:
                                        System.out.println(" Machine::
No specific instruction set");
                                        break;
                                case 0x01:
                                        System.out.println(" Machine::
AT & T WE 32100");
                                        break;
                                case 0x02:
                                        System.out.println(" Machine::
SPARC");
                                        break;
                                case 0x03:
                                        System.out.println(" Machine::
x86");
                                        break;
                                case 0x04:
```

```
                    System.out.println(" Machine::
Motorola 68000 (M68k)");
                    break;
                case 0x05:
                    System.out.println(" Machine::
Motorola 88000 (M68k)");
                    break;
                case 0x06:
                    System.out.println(" Machine::
Intel MCU");
                    break;
                case 0x07:
                    System.out.println(" Machine::
Intel 80860");
                    break;
                case 0x08:
                    System.out.println(" Machine::
MIPS");
                    break;
                case 0x09:
                    System.out.println(" Machine::
IBM_System/370");
                    break;
                case 0x0A:
                    System.out.println(" Machine::
MIPS RS3000 Little-endian");
                    break;
                case 0x0B:
                    System.out.println(" Machine::
Reserved for future use");
                    break;
                case 0x0C:
                    System.out.println(" Machine::
Reserved for future use");
                    break;
                case 0x0D:
                    System.out.println(" Machine::
Reserved for future use");
                    break;
                case 0x0E:
                    System.out.println(" Machine::
Hewlett-Packard PA-RISC");
                    break;
                case 0x0F:
                    System.out.println(" Machine::
Reserved for future use");
                    break;
                case 0x13:
                    System.out.println(" Machine::
Intel 80960");
```

```
                                    break;
                                case 0x14:
                                    System.out.println(" Machine::
PowerPC");
                                    break;
                                case 0x15:
                                    System.out.println(" Machine::
PowerPC (64-bit)");
                                    break;
                                case 0x16:
                                    System.out.println(" Machine::
S390, including S390x");
                                    break;
                                case 0x28:
                                    System.out.println(" Machine::
ARM (up to ARMv7/Aarch32");
                                    break;
                                case 0x2A:
                                    System.out.println(" Machine::
SuperH");
                                    break;
                                case 0x32:
                                    System.out.println(" Machine::
IA-64");
                                    break;
                                case 0x3E:
                                    System.out.println(" Machine::
amd64");
                                    break;
                                case 0x8C:
                                    System.out.println(" Machine::
TMS320C6000 Family");
                                    break;
                                case 0xB7:
                                    System.out.println(" Machine::
ARM 64-bits (ARMv8/Aarch64)");
                                    break;
                                case 0xF3:
                                    System.out.println(" Machine::
RISC-V");
                                    break;


                                }

                            //Elf header Version//

                            String V1 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+4] & 0xFF)).replace(' ','0');
```

```
                String V2 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+5] & 0xFF)).replace(' ','0');
                String V3 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+6] & 0xFF)).replace(' ','0');
                String V4 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+7] & 0xFF)).replace(' ','0');

                if (fileContent[0+5]==0x01){
                    String Binary_Result_V = V4+V3+V2+V1;
                    int INT_Binary_Result_V = Integer.parseInt(Binary_Result_V,2);

                        System.out.printf(" Version:
0x%d",INT_Binary_Result_V);
                        System.out.println();
                }else {
                    String Binary_Result_V = V1+V2+V3+V4;
                    int INT_Binary_Result_V = Integer.parseInt(Binary_Result_V,2);
                        System.out.printf(" Version:
0x%d",INT_Binary_Result_V);
                        System.out.println();
                }




                //Elf header Entry point address//
                if (fileContent[0+4]== 0x01){
                    System.out.printf(" Entry point address:
0x%X%02X%02X%02X"

,fileContent[i2+11],fileContent[i2+10],fileContent[i2+9],fileContent[i2+8]);
                    System.out.println();
                }else {
                    System.out.printf(" Entry point address:
0x%X%02X%02X%02X"

,fileContent[i2+8],fileContent[i2+9],fileContent[i2+10],fileContent[i2+11]);
                    System.out.println();
                }

                //Elf header Start of program headers//
                int Start_programheader;
                String S_PH1 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+12] & 0xFF)).replace(' ','0');
                String S_PH2 =
String.format("%8s",Integer.toBinaryString(fileContent[i2+13] & 0xFF)).replace(' ','0');
```

```
                    if (fileContent[0+5]== 0x01){
                    String Binary_Result_S_PH = S_PH2+S_PH1;

                    int INT_Binary_Result_S_PH =
Integer.parseInt(Binary_Result_S_PH,2);


                        System.out.printf(" Start of program headers:          %d
(bytes into file)",INT_Binary_Result_S_PH);
                        System.out.println();
                        Start_programheader = INT_Binary_Result_S_PH;
                    }else {
                        String Binary_Result_S_PH = S_PH1+S_PH2;

                    int INT_Binary_Result_S_PH =
Integer.parseInt(Binary_Result_S_PH,2);


                        System.out.printf(" Start of program headers:          %d
(bytes into file)",INT_Binary_Result_S_PH);
                        System.out.println();
                        Start_programheader = INT_Binary_Result_S_PH;
                    }


                    int i3 = i + 32;
                    //Elf header Start of section headers//
                    int Start_sectionheader;
                    String SCH1 =
String.format("%8s",Integer.toBinaryString(fileContent[i3] & 0xFF)).replace(' ','0');
                    String SCH2 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+1] & 0xFF)).replace(' ','0');
                    String SCH3 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+2] & 0xFF)).replace(' ','0');
                    String SCH4 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+3] & 0xFF)).replace(' ','0');

                    if (fileContent[0+5]==0x01){
                    String Binary_Result_SCH = SCH4+SCH3+SCH2+SCH1;

                    int INT_Binary_Result_SCH = Integer.parseInt(Binary_Result_SCH,2);

                        System.out.printf(" Start of section headers:          %d (bytes
into file)",INT_Binary_Result_SCH);
                        System.out.println();
                        Start_sectionheader = INT_Binary_Result_SCH;
                    } else {
                    String Binary_Result_SCH = SCH1+SCH2+SCH3+SCH4;

                    int INT_Binary_Result_SCH = Integer.parseInt(Binary_Result_SCH,2);
```

```
                    System.out.printf(" Start of section headers:          %d (bytes
into file)",INT_Binary_Result_SCH);
                    System.out.println();
                    Start_sectionheader = INT_Binary_Result_SCH;
                }


                //Elf header Flags//
                if (fileContent[i3+4]==0x01){
                    System.out.println(" Flags:                                0x1,
RVC, soft-float ABI");
                }

                //Elf header Size of this headers//
                if (fileContent[0+4]==0x01){
                    System.out.println(" Size of this headers:          52
(bytes)");
                }else {
                    System.out.println(" Size of this headers:          64
(bytes)");
                }

                //Elf header Size of program headers//

                String HS_PH1 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+10] & 0xFF)).replace(' ','0');
                String HS_PH2 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+11] & 0xFF)).replace(' ','0');

                if (fileContent[0+5]== 0x01){
                String Binary_Result_HS_PH = HS_PH2+HS_PH1;

                int INT_Binary_Result_HS_PH =
Integer.parseInt(Binary_Result_HS_PH,2);


                    System.out.printf(" Size of program headers:          %d
(bytes)",INT_Binary_Result_HS_PH);
                    System.out.println();
                }else {
                    String Binary_Result_HS_PH = HS_PH1+HS_PH2;

                int INT_Binary_Result_HS_PH =
Integer.parseInt(Binary_Result_HS_PH,2);


                    System.out.printf(" Size of program headers:          %d
(bytes)",INT_Binary_Result_HS_PH);
                    System.out.println();
```

```
                    }


                    //Elf header Number of program headers//
                    int Number_programheaders;
                    String HN_PH1 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+12] & 0xFF)).replace(' ','0');
                    String HN_PH2 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+13] & 0xFF)).replace(' ','0');

                    if (fileContent[0+5]== 0x01){
                    String Binary_Result_HN_PH = HN_PH2+HN_PH1;

                    int INT_Binary_Result_HN_PH =
Integer.parseInt(Binary_Result_HN_PH,2);


                        System.out.printf(" Number of program headers:
%d",INT_Binary_Result_HN_PH);
                        System.out.println();
                        Number_programheaders = INT_Binary_Result_HN_PH;

                    }else {
                    String Binary_Result_HN_PH = HN_PH1+HN_PH2;

                    int INT_Binary_Result_HN_PH =
Integer.parseInt(Binary_Result_HN_PH,2);


                        System.out.printf(" Number of program headers:
%d",INT_Binary_Result_HN_PH);
                        System.out.println();
                        Number_programheaders = INT_Binary_Result_HN_PH;
                    }
                    //Elf header Size of section headers//

                    String S_SH1 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+14] & 0xFF)).replace(' ','0');
                    String S_SH2 =
String.format("%8s",Integer.toBinaryString(fileContent[i3+15] & 0xFF)).replace(' ','0');

                    if(fileContent[0+5]== 0x01){
                    String Binary_Result_S_SH = S_SH2+S_SH1;

                    int INT_Binary_Result_S_SH =
Integer.parseInt(Binary_Result_S_SH,2);


                        System.out.printf(" Size of section headers:          %d
(bytes)",INT_Binary_Result_S_SH);
```

```
                    System.out.println();
                }else{
                String Binary_Result_S_SH = S_SH1+S_SH2;

                int INT_Binary_Result_S_SH =
Integer.parseInt(Binary_Result_S_SH,2);


                    System.out.printf(" Size of section headers:                %d
(bytes)",INT_Binary_Result_S_SH);
                    System.out.println();
                }

                int i4 = i+48;
                //Elf header Number of section headers //
                int Number_sectionheader;
                String N_SH1 =
String.format("%8s",Integer.toBinaryString(fileContent[i4] & 0xFF)).replace(' ','0');
                String N_SH2 =
String.format("%8s",Integer.toBinaryString(fileContent[i4+1] & 0xFF)).replace(' ','0');

                if (fileContent[0+5]== 0x01){
                String Binary_Result_N_SH = N_SH2+N_SH1;

                int INT_Binary_Result_N_SH =
Integer.parseInt(Binary_Result_N_SH,2);


                    System.out.printf(" Number of section headers:
%d",INT_Binary_Result_N_SH);
                    System.out.println();
                    Number_sectionheader = INT_Binary_Result_N_SH;
                }else{
                String Binary_Result_N_SH = N_SH1+N_SH2;

                int INT_Binary_Result_N_SH =
Integer.parseInt(Binary_Result_N_SH,2);


                    System.out.printf(" Number of section headers:
%d",INT_Binary_Result_N_SH);
                    System.out.println();
                    Number_sectionheader = INT_Binary_Result_N_SH;
                }
                //Elf header Section header string table index//
                String SH_Index1 =
String.format("%8s",Integer.toBinaryString(fileContent[i4+2] & 0xFF)).replace(' ','0');
                String SH_Index2 =
String.format("%8s",Integer.toBinaryString(fileContent[i4+3] & 0xFF)).replace(' ','0');
```

```
                    if (fileContent[0+5]==0x01){
                    String Binary_Result_SH_Index = SH_Index2+SH_Index1;

                    int INT_Binary_Result_SH_Index =
Integer.parseInt(Binary_Result_SH_Index,2);


                        System.out.printf(" Section header string table index:
%d",INT_Binary_Result_SH_Index);
                        System.out.println();
                    }else{
                    String Binary_Result_SH_Index = SH_Index1+SH_Index2;

                    int INT_Binary_Result_SH_Index =
Integer.parseInt(Binary_Result_SH_Index,2);


                        System.out.printf(" Section header string table index:
%d",INT_Binary_Result_SH_Index);
                        System.out.println();
                    }




System.out.println("=========================================");
                    // Program header//


                    System.out.println("Program Headers:");
                    System.out.println("  Type           Offset             VirtAddr
PhysAddr         FileSiz            MemSiz     Flg    Align");



//    ShiftReader
                    Start_programheader -= 4;

                    for(i = 0; i < Number_programheaders;i++){

                    int ShiftReader = 4;

                    Start_programheader += ShiftReader;
//TYPE
                    String PH_Type1 =
String.format("%02X",(fileContent[Start_programheader]));
                    String PH_Type2 =
String.format("%02X",(fileContent[Start_programheader+1]));
```

```
                String PH_Type3 =
String.format("%02X",(fileContent[Start_programheader+2]));
                String PH_Type4 =
String.format("%02X",(fileContent[Start_programheader+3]));

                String Result_PH_Type =
PH_Type4+PH_Type3+PH_Type2+PH_Type1;
                int INT_Result_PH_Type = Integer.parseInt(Result_PH_Type);

                switch (INT_Result_PH_Type){
                    case 0x00000000:
                            System.out.printf("     Null");
                            break;
                    case 0x00000001:
                            System.out.printf("     LOAD");
                            break;
                    case 0x00000002:
                            System.out.printf("     DYNAMIC");
                            break;
                    case 0x00000003:
                            System.out.printf("     INTERP");
                            break;
                    case 0x00000004:
                            System.out.printf("     NOTE");
                            break;
                    case 0x00000005:
                            System.out.printf("     SHLIB");
                            break;
                    case 0x00000006:
                            System.out.printf("     PHDR");
                            break;
                    case 0x00000007:
                            System.out.printf("     TLS");
                            break;
                    case 0x60000000:
                            System.out.printf("     LOOS");
                            break;
                    case 0x6FFFFFFF:
                            System.out.printf("     HIOS");
                            break;
                    case 0x70000000:
                            System.out.printf("     LOPROC");
                            break;
                    case 0x7FFFFFFF:
                            System.out.printf("     HIPROC");
                            break;

                }
//OFFSET
                Start_programheader += ShiftReader;
```

```
                String PH_Offset1 =
String.format("%02X",(fileContent[Start_programheader]));
                String PH_Offset2 =
String.format("%02X",(fileContent[Start_programheader+1]));
                String PH_Offset3 =
String.format("%02X",(fileContent[Start_programheader+2]));
                String PH_Offset4 =
String.format("%02X",(fileContent[Start_programheader+3]));

                String Result_PH_Offset =
PH_Offset4+PH_Offset3+PH_Offset2+PH_Offset1;
                System.out.printf("       0x"+Result_PH_Offset);

//VIRTADDR
                Start_programheader += ShiftReader;
                String PH_VirtAddr1 =
String.format("%02X",(fileContent[Start_programheader]));
                String PH_VirtAddr2 =
String.format("%02X",(fileContent[Start_programheader+1]));
                String PH_VirtAddr3 =
String.format("%02X",(fileContent[Start_programheader+2]));
                String PH_VirtAddr4 =
String.format("%02X",(fileContent[Start_programheader+3]));

                String Result_PH_VirtAddr =
PH_VirtAddr4+PH_VirtAddr3+PH_VirtAddr2+PH_VirtAddr1;
                System.out.printf("       0x"+Result_PH_VirtAddr);
//PHYSADDR
                Start_programheader += ShiftReader;
                String PH_PhysAddr1 =
String.format("%02X",(fileContent[Start_programheader]));
                String PH_PhysAddr2 =
String.format("%02X",(fileContent[Start_programheader+1]));
                String PH_PhysAddr3 =
String.format("%02X",(fileContent[Start_programheader+2]));
                String PH_PhysAddr4 =
String.format("%02X",(fileContent[Start_programheader+3]));

                String Result_PH_PhysAddr =
PH_PhysAddr4+PH_PhysAddr3+PH_PhysAddr2+PH_PhysAddr1;
                System.out.printf("     0x"+Result_PH_PhysAddr);
//FileSiz
                Start_programheader += ShiftReader;
                String FileSiz1 =
String.format("%02X",(fileContent[Start_programheader]));
                String FileSiz2 =
String.format("%02X",(fileContent[Start_programheader+1]));
                String FileSiz3 =
String.format("%02X",(fileContent[Start_programheader+2]));
```

```
                String FileSiz4 =
String.format("%02X",(fileContent[Start_programheader+3]));

                String Result_FileSiz = FileSiz4+FileSiz3+FileSiz2+FileSiz1;
                System.out.printf("      0x"+Result_FileSiz);

//MemSiz
                Start_programheader += ShiftReader;
                String MemSiz1 =
String.format("%02X",(fileContent[Start_programheader]));
                String MemSiz2 =
String.format("%02X",(fileContent[Start_programheader+1]));
                String MemSiz3 =
String.format("%02X",(fileContent[Start_programheader+2]));
                String MemSiz4 =
String.format("%02X",(fileContent[Start_programheader+3]));

                String Result_MemSiz = MemSiz4+MemSiz3+MemSiz2+MemSiz1;
                System.out.printf("      0x"+Result_MemSiz);

//Flg
                Start_programheader += ShiftReader;
                String Flg1 =
String.format("%02X",(fileContent[Start_programheader]));
                String Flg2 =
String.format("%02X",(fileContent[Start_programheader+1]));
                String Flg3 =
String.format("%02X",(fileContent[Start_programheader+2]));
                String Flg4 =
String.format("%02X",(fileContent[Start_programheader+3]));

                String Result_Flg = Flg4+Flg3+Flg2+Flg1;
                int INT_Result_Flg = Integer.parseInt(Result_Flg);
                switch (INT_Result_Flg){
                    case 0x00000007:
                        System.out.printf(" RWE");
                        break;
                    case 0x00000006:
                        System.out.printf(" RW ");
                        break;
                    default:
                        System.out.printf(" R E");
                }

//Align
                Start_programheader += ShiftReader;
                String Align1 =
String.format("%02X",(fileContent[Start_programheader]));
                String Align2 =
String.format("%02X",(fileContent[Start_programheader+1]));
```

```
                    String Align3 =
String.format("%02X",(fileContent[Start_programheader+2]));
                    String Align4 =
String.format("%02X",(fileContent[Start_programheader+3]));

                    String Result_Align = Align4+Align3+Align2+Align1;
                    System.out.printf("    0x"+Result_Align);
                    System.out.println();
                    }

System.out.println("==================================================
=========================================");

// Section header
                    int Shiftreader = 4;
                    Start_sectionheader -=Shiftreader;
                    System.out.println("Section Headers:");
                    System.out.println("       [Nr] Name                                    Type
Addr       "
                                + " Off          Size            ES    Flg    LK    Inf Al");

//Get the name from last section header

                    String SH_LAST_Offset1 =
String.format("%8s",Integer.toBinaryString(fileContent[(Start_sectionheader+Shiftreader)+4
0*(Number_sectionheader-1)+4*Shiftreader] & 0xFF)).replace(' ','0');
                    String SH_LAST_Offset2 =
String.format("%8s",Integer.toBinaryString(fileContent[(Start_sectionheader+Shiftreader)+4
0*(Number_sectionheader-1)+4*Shiftreader+1] & 0xFF)).replace(' ','0');
                    String SH_LAST_Offset3 =
String.format("%8s",Integer.toBinaryString(fileContent[(Start_sectionheader+Shiftreader)+4
0*(Number_sectionheader-1)+4*Shiftreader+2] & 0xFF)).replace(' ','0');
                    String SH_LAST_Offset4 =
String.format("%8s",Integer.toBinaryString(fileContent[(Start_sectionheader+Shiftreader)+4
0*(Number_sectionheader-1)+4*Shiftreader+3] & 0xFF)).replace(' ','0');


                    String Result_SH_LAST_Offset =
SH_LAST_Offset4+SH_LAST_Offset3+SH_LAST_Offset2+SH_LAST_Offset1;
                    int INT_Result_SH_LAST_Offset =
Integer.parseInt(Result_SH_LAST_Offset, 2);
                    int Start_String_Table = INT_Result_SH_LAST_Offset;

                    for (i=0;i<Number_sectionheader;i++){
//Number

                    System.out.printf("       [%2d] ", i);

//NAME
                    Start_sectionheader += Shiftreader;
```

```
                String SH_Name1 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader] &
0xFF)).replace(' ','0');
                String SH_Name2 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+1] &
0xFF)).replace(' ','0');
                String SH_Name3 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+2] &
0xFF)).replace(' ','0');
                String SH_Name4 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+3] &
0xFF)).replace(' ','0');


                String Result_SH_Name =
SH_Name4+SH_Name3+SH_Name2+SH_Name1;
                int INT_Result_SH_Name = Integer.parseInt(Result_SH_Name,2);
                int Start_Name = Start_String_Table + INT_Result_SH_Name;

                int INT_DisplayName = 1;
                char Char_DisplayName;

                for (int k=0;INT_DisplayName!=0;k++){
                String DisplayName =
String.format("%8s",Integer.toBinaryString(fileContent[Start_Name+k] & 0xFF)).replace('
','0');
                INT_DisplayName = Integer.parseInt(DisplayName, 2);
                Char_DisplayName = (char) INT_DisplayName;

                if(INT_DisplayName!=0){
                System.out.printf("%c",Char_DisplayName);
                }else{
                      int remain = 26 - k;
                      for (int j=0;j<remain;j++){
                      System.out.printf(" ");}
                }

                }
//TYPE
                Start_sectionheader += Shiftreader;
                String SH_Type1 =
String.format("%02X",(fileContent[Start_sectionheader]));
                String SH_Type2 =
String.format("%02X",(fileContent[Start_sectionheader+1]));
                String SH_Type3 =
String.format("%02X",(fileContent[Start_sectionheader+2]));
                String SH_Type4 =
String.format("%02X",(fileContent[Start_sectionheader+3]));
```

```
                String Result_SH_Type =
SH_Type4+SH_Type3+SH_Type2+SH_Type1;
                int INT_Result_PH_Type = Integer.parseInt(Result_SH_Type);

                switch (INT_Result_PH_Type){
                    case 0:
                            System.out.printf("     Null      ");
                            break;
                    case 1:
                            System.out.printf("     PROGBITS");
                            break;
                    case 2:
                            System.out.printf("     SYMTAB    ");
                            break;
                    case 3:
                            System.out.printf("     STRTAB    ");
                            break;
                    case 4:
                            System.out.printf("     RELA      ");
                            break;
                    case 5:
                            System.out.printf("     HASH      ");
                            break;
                    case 6:
                            System.out.printf("     DYNAMIC ");
                            break;
                    case 7:
                            System.out.printf("     NOTE      ");
                            break;
                    case 8:
                            System.out.printf("     NOBITS    ");
                            break;
                    case 9:
                            System.out.printf("     REL       ");
                            break;
                    case 10:
                            System.out.printf("     SHLIB     ");
                            break;
                    case 11:
                            System.out.printf("     DYNSYM    ");
                            break;
                    case 0x70000000:
                            System.out.printf("     LOPROC    ");
                            break;
                    case 0x7FFFFFFF:
                            System.out.printf("     HIPROC    ");
                            break;
                    case 0x80000000:
                            System.out.printf("     LOUSER    ");
                            break;
```

```
                    case 0x8FFFFFFF:
                            System.out.printf("       HIUSER    ");
                            break;
                }

//Flg
                Start_sectionheader += Shiftreader;
                String Flg1 = String.format("%02X",(fileContent[Start_sectionheader]));
                String Flg2 =
String.format("%02X",(fileContent[Start_sectionheader+1]));
                String Flg3 =
String.format("%02X",(fileContent[Start_sectionheader+2]));
                String Flg4 =
String.format("%02X",(fileContent[Start_sectionheader+3]));

                String SH_Result_Flg = Flg4+Flg3+Flg2+Flg1;
                int INT_SH_Result_Flg = Integer.parseInt(SH_Result_Flg);
//Address
                Start_sectionheader += Shiftreader;
                String SH_Addr1 =
String.format("%02X",(fileContent[Start_sectionheader]));
                String SH_Addr2 =
String.format("%02X",(fileContent[Start_sectionheader+1]));
                String SH_Addr3 =
String.format("%02X",(fileContent[Start_sectionheader+2]));
                String SH_Addr4 =
String.format("%02X",(fileContent[Start_sectionheader+3]));

                String Result_SH_Addr =
SH_Addr4+SH_Addr3+SH_Addr2+SH_Addr1;
                System.out.printf("              "+Result_SH_Addr);

//OFFSET
                Start_sectionheader += Shiftreader;
                String SH_Offset1 =
String.format("%02X",(fileContent[Start_sectionheader]));
                String SH_Offset2 =
String.format("%02X",(fileContent[Start_sectionheader+1]));
                String SH_Offset3 =
String.format("%02X",(fileContent[Start_sectionheader+2]));
                String SH_Offset4 =
String.format("%02X",(fileContent[Start_sectionheader+3]));

                String Result_SH_Offset =
SH_Offset4+SH_Offset3+SH_Offset2+SH_Offset1;
                System.out.printf(" "+Result_SH_Offset);


//Siz
                Start_sectionheader += Shiftreader;
```

```
                    String Siz1 = String.format("%02X",(fileContent[Start_sectionheader]));
                    String Siz2 =
String.format("%02X",(fileContent[Start_sectionheader+1]));
                    String Siz3 =
String.format("%02X",(fileContent[Start_sectionheader+2]));
                    String Siz4 =
String.format("%02X",(fileContent[Start_sectionheader+3]));

                    String Result_Siz = Siz4+Siz3+Siz2+Siz1;
                    System.out.printf(" "+Result_Siz);

//LINK
                    Start_sectionheader += Shiftreader;
                    String Link1 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader]&
0xFF)).replace(' ','0');
                    String Link2 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+1]&
0xFF)).replace(' ','0');
                    String Link3 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+2]&
0xFF)).replace(' ','0');
                    String Link4 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+3]&
0xFF)).replace(' ','0');


                    String Result_Link = Link4+Link3+Link2+Link1;
                    int INT_Result_Link = Integer.parseInt(Result_Link,2);
//INFO
                    Start_sectionheader += Shiftreader;
                    String Info1 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader]&
0xFF)).replace(' ','0');
                    String Info2 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+1]&
0xFF)).replace(' ','0');
                    String Info3 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+2]&
0xFF)).replace(' ','0');
                    String Info4 =
String.format("%8s",Integer.toBinaryString(fileContent[Start_sectionheader+3]&
0xFF)).replace(' ','0');

                    String Result_Info = Info4+Info3+Info2+Info1;
                    int INT_Result_Info = Integer.parseInt(Result_Info,2);

//ALIGN
                    Start_sectionheader += Shiftreader;
```

```
                String SH_Align1 =
String.format("%02X",(fileContent[Start_sectionheader]));
                String SH_Align2 =
String.format("%02X",(fileContent[Start_sectionheader+1]));
                String SH_Align3 =
String.format("%02X",(fileContent[Start_sectionheader+2]));
                String SH_Align4 =
String.format("%02X",(fileContent[Start_sectionheader+3]));

                String Result_Align = SH_Align4+SH_Align3+SH_Align2+SH_Align1;
                int INT_Result_Align = Integer.parseInt(Result_Align);


//ENTSiz

                Start_sectionheader += Shiftreader;
                String ENTSiz1 =
String.format("%02X",(fileContent[Start_sectionheader]));
                String ENTSiz2 =
String.format("%02X",(fileContent[Start_sectionheader+1]));
                String ENTSiz3 =
String.format("%02X",(fileContent[Start_sectionheader+2]));
                String ENTSiz4 =
String.format("%02X",(fileContent[Start_sectionheader+3]));

                String Result_ENTSiz = ENTSiz4+ENTSiz3+ENTSiz2+ENTSiz1;
                int INT_Result_ENTSiz = Integer.parseInt(Result_ENTSiz);

//prinf ES-AL
                System.out.printf("     %02d",INT_Result_ENTSiz);

                switch (INT_SH_Result_Flg){
                    case 0x00000000:
                        System.out.printf("         ");
                        break;
                    case 0x00000001:
                        System.out.printf("       W");
                        break;
                    case 0x00000002:
                        System.out.printf("       A");
                        break;
                    case 0x00000004:
                        System.out.printf("       X");
                        break;
                    case 0x00000010:
                        System.out.printf("       M");
                        break;
                    case 0x00000020:
                        System.out.printf("       S");
                        break;
```

```
                case 0x00000040:
                    System.out.printf("       I");
                    break;
                case 0x00000080:
                    System.out.printf("       L");
                    break;
                case 0x00000100:
                    System.out.printf("       O");
                    break;
                case 0x00000200:
                    System.out.printf("       G");
                    break;
                case 0x00000400:
                    System.out.printf("       T");
                    break;
                case 0x0FF00000:
                    System.out.printf("       o");
                    break;
                case 0xF0000000:
                    System.out.printf("       p");
                    break;
                case 0x40000000:
                    System.out.printf("         ");
                    break;
                case 0x80000000:
                    System.out.printf("       E");
                    break;
                case 0x00000003:
                    System.out.printf("      WA");
                    break;
                case 0x00000006:
                    System.out.printf("      AX");
                    break;
                case 30:
                    System.out.printf("      MS");
                    break;
                default:
                    System.out.printf("      NO");
            }
            System.out.printf("    %2d",INT_Result_Link);
            System.out.printf("     %2d",INT_Result_Info);
            if(INT_Result_Align == 0||INT_Result_Align == 1||INT_Result_Align
== 2){
                System.out.printf(" %2d",INT_Result_Align);
            }else if(INT_Result_Align >2 && INT_Result_Align <=4){
                System.out.printf("    4",INT_Result_Align);
            }else if(INT_Result_Align >4 && INT_Result_Align <=8){
                System.out.printf("    8",INT_Result_Align);
            }else if(INT_Result_Align >8 && INT_Result_Align <=64){
                System.out.printf(" 64",INT_Result_Align);
```

```
                    }else if(INT_Result_Align >64 && INT_Result_Align <=128){
                          System.out.printf(" 128",INT_Result_Align);
                    }
                    System.out.println();

                    }


System.out.println("==========================================================
===");









//Dwarf debug_line
                    System.out.println("Raw dump of debug contents of section
.debug_line:");
                    System.out.println();

                int DBL_Offset = 1029;
                int DBL_Size = 327;
                int Offset_CU = 0;

                while(Offset_CU!=DBL_Size){


//Offset
                    System.out.printf(" Offset:
0x%x",Offset_CU); System.out.println();
//Length
                    String Length1 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset] & 0xFF)).replace(' ','0');
                    String Length2 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset+1] & 0xFF)).replace('
','0');
                    String Length3 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset+2] & 0xFF)).replace('
','0');
                    String Length4 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset+3] & 0xFF)).replace('
','0');
```

```
                    Offset_CU += 4;
                    DBL_Offset+=4;

                    String Length = Length4+Length3+Length2+Length1;
                    int Result_Length = Integer.parseInt(Length, 16);

                    System.out.printf(" Length:
"+Result_Length); System.out.println();
//Dwarf Version

                    String Version1 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset] & 0xFF)).replace(' ','0');
                    String Version2 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset+1] & 0xFF)).replace('
','0');

                    Offset_CU += 2;
                    DBL_Offset+=2;

                    String Version = Version2+Version1;
                    int Result_Version = Integer.parseInt(Version, 16);

                    System.out.printf(" DWARF Version:
"+Result_Version); System.out.println();
//Prologue Length

                    String Prologue_Length1 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset] & 0xFF)).replace(' ','0');
                    String Prologue_Length2 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset+1] & 0xFF)).replace('
','0');
                    String Prologue_Length3 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset+2] & 0xFF)).replace('
','0');
                    String Prologue_Length4 =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset+3] & 0xFF)).replace('
','0');

                    Offset_CU+=4;
                    DBL_Offset+=4;

                    String Prologue_Length =
Prologue_Length4+Prologue_Length3+Prologue_Length2+Prologue_Length1;
                    int Result_Prologue_Length = Integer.parseInt(Prologue_Length, 16);
                    System.out.printf(" Prologue Length:
"+Result_Prologue_Length); System.out.println();

//Minimum Instruction Length
                    int MIL = fileContent[DBL_Offset];
```

```
                String Minimum_Instrustion_Length =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset] & 0xFF)).replace(' ','0');
                System.out.printf(" Minimum Instruction Length:          "+
Minimum_Instrustion_Length);System.out.println();

                Offset_CU +=1;
                DBL_Offset+=1;

//Initial value of 'is_stmt'

                String IS_STMT =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset] & 0xFF)).replace(' ','0');
                System.out.printf(" Initial value of 'is_stmt':          "+
IS_STMT);System.out.println();

                Offset_CU +=1;
                DBL_Offset+=1;

//Line Base
                int Line_Base = fileContent[DBL_Offset];
                System.out.printf(" Line Base:
"+Line_Base);System.out.println();
                Offset_CU +=1;
                DBL_Offset+=1;

//Line Range
                int Line_Range = fileContent[DBL_Offset];
                System.out.printf(" Line Range:
"+Line_Range);System.out.println();
                Offset_CU +=1;
                DBL_Offset+=1;

//Opcode Base
                int Opcode_Base = fileContent[DBL_Offset];
                System.out.printf(" Opcode Base:
"+Opcode_Base);System.out.println();System.out.println();
                Offset_CU +=1;
                DBL_Offset+=1;

//Opcode 12 Standard opcode
                System.out.printf("Opcodes:");System.out.println();
                for(int o=1;o<Opcode_Base;o++){
                    switch (fileContent[DBL_Offset]){
                        case 0x01:
                            System.out.printf(" Opcode %2d has %d
arg",o,fileContent[DBL_Offset]);System.out.println();
                            Offset_CU +=1;
                            DBL_Offset+=1;
                            break;
                        default:
```

```
                                    System.out.printf(" Opcode %2d has %d
args",o,fileContent[DBL_Offset]);System.out.println();
                                    Offset_CU +=1;
                                    DBL_Offset+=1;
                                    break;
                                }
                            }
//Directory table
                    System.out.println();
                    if (fileContent[DBL_Offset]==0x00){
                        System.out.printf("The Directory Table is
empty.");System.out.println();System.out.println();
                        Offset_CU +=1;
                        DBL_Offset+=1;
                    }
//File name offset

                    System.out.printf("The File Name Table (offset
0x%2x):",Offset_CU);System.out.println();
                    System.out.printf(" Entry       Dir   Time       Size
      Name");System.out.println();

//save offset for name & get the Dir Time and Size
                    int Save_Offset = DBL_Offset;
                    int One = 1;

                    while (One!=0){
                        if (fileContent[DBL_Offset]!=0x00){
                            DBL_Offset+=1;
                            Offset_CU+=1;
                        }else{
                            DBL_Offset+=1;
                            Offset_CU+=1;
                            String Dir =
String.format("%1s",Integer.toHexString(fileContent[DBL_Offset] & 0xFF)).replace(' ','0');
                            String Time =
String.format("%1s",Integer.toHexString(fileContent[DBL_Offset+1] & 0xFF)).replace('
','0');
                            String Size =
String.format("%1s",Integer.toHexString(fileContent[DBL_Offset+2] & 0xFF)).replace('
','0');

                            Offset_CU +=3;
                            DBL_Offset +=3;

                            System.out.printf(" 1           "+Dir+"           "+Time+"
"+Size+"           ");
                            One = 0;
                        }
                    }
```

```
                int DebugLine_DisplayFileName = 1;
                char Char_DebugLine_DisplayFileName;

                for (int k=0;DebugLine_DisplayFileName!=0;k++){
                    String DisplayName =
String.format("%2s",Integer.toHexString(fileContent[Save_Offset+k] & 0xFF)).replace('
','0');
                    DebugLine_DisplayFileName = Integer.parseInt(DisplayName, 16);
                    Char_DebugLine_DisplayFileName = (char)
DebugLine_DisplayFileName;

                    if(DebugLine_DisplayFileName!=0){
                    System.out.printf("%c", Char_DebugLine_DisplayFileName);
                    }
                }
                System.out.println();System.out.println();

                if(fileContent[DBL_Offset]==0){
                    DBL_Offset+=1;
                    Offset_CU+=1;
                }

// main code

                System.out.println("Line Number Statements:");
                int CU_addr=0;
                String Result_Ex_addr = "";
                int Line =1;
                int Copy=0;
                int CU_addr_add;
                int Line_add;
                int end=0;

                while (end!=1){
                System.out.printf(" [0x%08x] ",Offset_CU);


                switch (fileContent[DBL_Offset]){

                    case 0x01:
                        if(Copy==0){System.out.printf("Copy");System.out.println();
                        }else{
                            System.out.printf("Copy (view%d)",
Copy);System.out.println();}
                        Copy+=1;
                        DBL_Offset+=1;
                        Offset_CU+=1;
                        break;
                    case 0x02:
```

```
                        DBL_Offset+=1;
                        Offset_CU+=1;
                            System.out.printf("Set column to %d",
fileContent[DBL_Offset]);System.out.println();
                        DBL_Offset+=1;
                        Offset_CU+=1;
                        break;
                    case 0x03:
                        DBL_Offset+=1;
                        Offset_CU+=1;
                        int LEB128 = fileContent[DBL_Offset];
                            if(LEB128>=64){
                                LEB128=-1*(128-LEB128);
                            }
                            Line +=LEB128;
                            System.out.printf("Advance Line by %d to %d",
LEB128,Line);System.out.println();
                        DBL_Offset+=1;
                        Offset_CU+=1;
                        break;
                    case 0x04:
                    case 0x05:
                        DBL_Offset+=1;
                        Offset_CU+=1;
                            System.out.printf("Set column to %d",
fileContent[DBL_Offset]);System.out.println();
                        DBL_Offset+=1;
                        Offset_CU+=1;
                        break;
                    case 0x06:
                    case 0x07:
                    case 0x08:
                    case 0x09:
                        DBL_Offset+=1;
                        Offset_CU+=1;
                        CU_addr+= fileContent[DBL_Offset];
                            System.out.printf("Advance PC by fixed size amount %d to
0x%x",fileContent[DBL_Offset],CU_addr);
                            System.out.println();
                        DBL_Offset+=2;
                        Offset_CU+=2;
                        break;
                    case 0x00:
                        DBL_Offset+=1;
                        Offset_CU+=1;
                        int Byte_num = fileContent[DBL_Offset];
                        DBL_Offset+=1;
                        Offset_CU+=1;
                        switch (fileContent[DBL_Offset]){
                            case 0x01:
```

```
                                    DBL_Offset+=1;
                                    Offset_CU+=1;
                                    System.out.printf("Extended opcode 1: End of
Sequence");System.out.println();System.out.println();
                                    end += 1;
                                    break;
                                case 0x02:
                                        DBL_Offset+=1;
                                        Offset_CU+=1;
                                    for(int k=1;k<Byte_num;k++){
                                        String Ex_addr =
String.format("%2s",Integer.toHexString(fileContent[DBL_Offset] & 0xFF)).replace(' ','0');
                                        DBL_Offset+=1;
                                        Offset_CU+=1;
                                        Result_Ex_addr = Ex_addr+Result_Ex_addr;
                                        CU_addr = Integer.parseInt(Result_Ex_addr, 16);
                                    }
                                    System.out.printf("Extended opcode 2: set Address to
0x%x",CU_addr);System.out.println();
                                    break;
                                case 0x04:
                                        DBL_Offset+=1;
                                        Offset_CU+=1;
                                        System.out.printf("Extented opcode 4: set
Discriminator to %d", fileContent[DBL_Offset]);System.out.println();
                                        DBL_Offset+=1;
                                        Offset_CU+=1;
                                    break;
                            }
                            break;
                        default:
                            int Sp_Opcode = fileContent[DBL_Offset]-Opcode_Base;
                            CU_addr_add = (Sp_Opcode/Line_Range)*MIL;
                            Line_add = Line_Base+Sp_Opcode%Line_Range;
                            Line +=Line_add;
                            CU_addr += CU_addr_add;

                            System.out.printf("Special opcode %d: advance Address by %d to
0x%x and Line by %d to %d"
                                    , Sp_Opcode,CU_addr_add, CU_addr, Line_add,
Line);System.out.println();



                            DBL_Offset+=1;
                            Offset_CU+=1;
                            break;
```

```
                }

                }

                }



}


        catch (FileNotFoundException e) {
                System.out.println("File not found" + e);
        }
        catch (IOException ioe) {
                System.out.println("Exception while reading file " + ioe);
        }
        finally {
                // close the streams using close method
                try {
                        if (fin != null) {
                                fin.close();
                        }
                }
                catch (IOException ioe) {
                        System.out.println("Error while closing stream: " + ioe);
                }
        }
    }
}
```

APPENDIX B – CODE FOR COMMAND LINE

```
package hk.quantr.hkdataformat;

import static hk.quantr.assembler.riscv.TestUtil.logger;
import hk.quantr.hkdataformat.antlr.HKDataFormatLexer;
import hk.quantr.hkdataformat.antlr.HKDataFormatParser;
import hk.quantr.hkdataformat.datastructure.Node;
import hk.quantr.peterswing.CommonLib;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URL;
import java.util.Arrays;
import java.util.logging.Level;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.DefaultParser;
import org.apache.commons.cli.Option;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.apache.commons.lang.exception.ExceptionUtils;

/**
 *
 * @author Peter <peter@quantr.hk>
 */
public class HKDataFormat {

    public static void write(OutputStream os, Node node) throws IOException {
        write(os, node, 0);
    }

    private static void write(OutputStream os, Node node, int indent) throws IOException {
        writeString(os, " ".repeat(indent) + "-" + node.name);
        if (node.value != null) {
            if (node.value instanceof byte[]) {
                byte[] bytes = (byte[]) node.value;
                writeString(os, ":" + CommonLib.arrayToHexString(bytes));
            } else if (node.value instanceof Integer) {
                int i = (int) node.value;
                writeString(os, ":0x" + Integer.toHexString(i));
            } else if (node.value instanceof File) {
                File file = (File) node.value;
                writeString(os, ":@(" + file.getAbsolutePath() + ")");
```

```
            } else if (node.value instanceof URL) {
                URL url = (URL) node.value;
                writeString(os, ":@(" + url + ")");
            } else {
                writeString(os, ":" + node.value.toString());
            }
        }
        writeString(os, "\n");
        for (Node child : node.children) {
            write(os, child, indent + 1);
        }
    }

    private static void writeString(OutputStream os, String str) throws IOException {
        os.write(str.getBytes());
    }

    public static void main(String args[]) throws FileNotFoundException, IOException,
IOException {

        CommandLineParser parser1 = new DefaultParser();
        Options options = new Options();
        try {
            options.addOption(Option.builder("c")
                    .required(false)
                    .argName("command")
                    .hasArg()
                    //                  .desc("")
                    //                  .longOpt("")
                    .numberOfArgs(1)
                    //                  .valueSeparator('=')
                    .build());


            CommandLine cmd = parser1.parse(options, args);
            HKDataFormatLexer lexer = new
HKDataFormatLexer(CharStreams.fromStream(new FileInputStream(cmd.getArgs()[0])));
            CommonTokenStream tokenStream = new CommonTokenStream(lexer);
            HKDataFormatParser parser = new HKDataFormatParser(tokenStream);
            MyListener listener = new MyListener(lexer, tokenStream);
            parser.addParseListener(listener);
            parser.hk();

            Node root = listener.root;
            if (!cmd.hasOption("c")) {
                System.out.println("no Command");
                System.exit(1);
            } else {
                if (cmd.getOptionValue("c").equals("print")) {
                    String queryP = cmd.getArgs()[1];
```

```
//                      String queryP = cmd.getOptionValues("c")[2];
                        String[] strsP = queryP.split("/");
                        strsP = Arrays.copyOfRange(strsP, 1, strsP.length);

                        Node nodeP = getNode(root, strsP);

                        if (nodeP.children.size() > 0) {

    System.out.println("================================================");
                            System.out.println("Function : Print");

    System.out.println("================================================");

                            System.out.println("The value of (" + queryP + "):");
                            nodeP.print(0);

    System.out.println("================================================");
                        } else {

    System.out.println("================================================");
                            System.out.println("Function : Print");

    System.out.println("================================================");

                            System.out.println("The value of (" + queryP + "):");
                            System.out.println(nodeP.name);

    System.out.println("================================================");
                        }
                    } else if (cmd.getOptionValue("c").equals("copy")) {
                        String from = cmd.getArgs()[1];
                        String to = cmd.getArgs()[2];
                        Node nodeFrom = getNode(root, from);


    System.out.println("================================================");
                        System.out.println("Function : Copy");
                        System.out.println("       From : " + from + "");
                        System.out.println("         To : " + to + "");

    System.out.println("================================================");
                        Node nodeTo = getNode(root, to);
                        nodeTo.add(nodeFrom);
                        System.out.println("The file has been copied as below");
                        root.print(0);

    System.out.println("================================================");
                    } else if (cmd.getOptionValue("c").equals("move")) {

                        String from = cmd.getArgs()[1];
```

```
                        String to = cmd.getArgs()[2];
                        Node nodeFrom = getNode(root, from);

                        Node nodeTo = getNode(root, to);
                        nodeTo.add(nodeFrom);

    System.out.println("================================================");
                        System.out.println("Function : Move");
                        System.out.println("       From : " + from + "");
                        System.out.println("         To : " + to + "");

    System.out.println("================================================");

                        String[] nodesParentArray = to.split("/");
                        nodesParentArray = Arrays.copyOfRange(nodesParentArray, 1,
nodesParentArray.length - 1);
                        Node deleteParent = getNode(root, nodesParentArray);
                        Node deleteNode = getNode(root, from);
                        deleteParent.children.remove(deleteNode);
                        System.out.println("The file has been moved as below");
                        root.print(0);

    System.out.println("================================================");

                    } else if (cmd.getOptionValue("c").equals("delete")) {

    System.out.println("================================================");
                        System.out.println("Function : Delete");

    System.out.println("================================================");

                        String delete = cmd.getArgs()[1];
                        System.out.println(delete);

                        String[] nodesParentArray = delete.split("/");
                        nodesParentArray = Arrays.copyOfRange(nodesParentArray, 1,
nodesParentArray.length - 1);
                        Node deleteParent = getNode(root, nodesParentArray);
                        Node deleteNode = getNode(root, delete);
                        deleteParent.children.remove(deleteNode);
                        System.out.println("The file has been deleted as below");
                        root.print(0);

    System.out.println("================================================");
                    }

                }

            } catch (ParseException ex) {
                logger.log(Level.SEVERE, ExceptionUtils.getFullStackTrace(ex));
```

```
                System.exit(1);
            }
        }

        public static Node getNode(Node node, String path) {
            String[] temp = path.split("/");
            temp = Arrays.copyOfRange(temp, 1, temp.length);
            return getNode(node, temp);
        }

        public static Node getNode(Node node, String arr[]) {
            if (arr.length == 0) {
                return node;
            } else {
                if (node.name.equals(arr[0])) {
                    arr = Arrays.copyOfRange(arr, 1, arr.length);
                    Node tempNode = getNode(node, arr);
                    return tempNode;
                } else {
                    for (Node child : node.children) {
                        if (child.name.split(":")[0].equals(arr[0])) {
                            arr = Arrays.copyOfRange(arr, 1, arr.length);
                            Node tempNode = getNode(child, arr);
                            return tempNode;
                        }

                    }
                }
            }
            return null;
        }
    }
```