The Implementation of Simplified Universal Assembler in Forth Assembly Language

Gang-Jeng Huang and Shiuh-Ku Weng
Department of Information Engineering, Chung Cheng Institute of Technology National Defense University, Taoyuan, Taiwan (R.O.C)
Email: qoo24571864@gmail.com, skw@ndu.edu.tw

Jyi-Jinn Chang
Instrument Technology Research Center, National Applied Research Laboratories, Taipei, Taiwan (R.O.C)

***Abstract*—In this paper, SDK (Software Development Kits) and IDE (Integrated Development Environment) of forth assembly language is designed. It can be assembled to different machine codes and would replace the traditional assembly languages, such as x86, ARM, MIPS and so on. The forth language is a kind of algebra language that is easier to code and to debug the programs. Therefore, it is a good tool language to be taught in the courses of computer organization and architecture for the students learning the low-level language easily and quickly.**

***Index Terms*—forth, assembly language, machine code, assembler**

I. Introduction

Forth language is a natural, mathematical, and algebra language. Unlike the other traditional assembly languages, it is easy to learn and develop applications for students and programmers. This assembler can translate assembly codes to the machine codes depended on different kinds of CPU. Especially, it will be assembled directly to the machine codes, because it is assembly language, it does not need to compile. [1], [2]

This language can be used in embedded system, PC desktop application and so on. The beginner who does not know about the knowledge of computer will feel easy to learn assembly language. The programmer can design and debug programs faster and easier than the other traditional languages.

This forth system assembly language is simple to use, easy to learn. From now on, there are too many assembly language in this world, and each assembly language cannot compile with each other, like x86 and ARM. If it could be defined a standard assembly language, programmers can just use only one assembly language to develop applications, device's drivers and so on. This will help programmer to program in more effect and do not spend too much time in debugging. It can be not only used in embedded system, but also in PC desktop to design a programs, applications, and drivers. Especially PC desktop, it supports to assemble the source codes in forth assembly language in Linux, Windows, MacOS, and so on.

The machine codes assembled from the forth assembly language are extremely tinny. Therefore, it can be assembled to many types of CPU's machine codes, such as x86, ARM, MIPS, and etc.

Although it is a not popular programming language, it can be used for the spirit of the forth language system to develop a simplest universal assembler. At that time, the learner can just use only one assembly language and shorten the schedule of developing programs and drivers. Also, it is much easier to design a compiler for higher-level because it does not need to compile with each of CPU's instructions. [3]

II. Motivation

The forth assembly language can be assembled to different machine codes. If there has only one assembly language, it would be helpful for designing compiler for high-level programming language, such as C, C++, Basic, and so forth, and shortening the time of developing and enhancing the performance for execution. That is why the forth assembly language may compile with most of the CPU instructions to the only one assembly language.

The forth language has been developed and researched by many experts. It has inherited the spiritual of high-level programming language so that it is simple to use, easy to learn. Because of its advantage, it can be a good course for senior high students, freshmen to learn about the foundation of computer science.

III. Design

A. About Assembler

This assembler is developed by C++ language. It can be compiled at Windows, Linux, and MacOS operation systems. The user can command the orders to do something, like assembling, debugging, and dis-assembling. As the source codes are assembled, it would generate two files, one is ".hex", and another is ".bin". The hex file recorded the machine codes in hex number. The bin file is generated in binary format. And, the binary file can be executed. However, if it has errors, then it would not generate any file and trigger warning messages.

Manuscript received December 30, 2013; revised April 28, 2014.

doi: 10.12720/ijsps.2.1.78-82

B. *Define Operator, Branch, Instruction*

And now, it can be designed the base of the forth language rule to build a table for compiling with other CPU instructions. From the Table III, like R1=23. R1=23 means load the value "23" (in forth language, all numbers are hex number) into R1 (Register 1). In x86 instruction, it is needed to use the "MOV" instruction to let the number move to the register [4]. In MIPS instruction [5], it is needed to use "ADDI" instruction to let number move to the register. And the syntax is not like x86, ARM, it should be designed as "ADDI R1, R1, 23". Another, if the programmer wants to let a register to subtract a value, like R1-23, in ARM's instruction it should be designed as "SUB R1, 23", and in MIPS it should be designed as "ADDI R1, R1, 23". [6], [7]

Then, these confusing and complex problems may let the beginners feel difficult to learn about each CPU's instructions and he/she has to pay a lot of time to understand each of CPU's instruction working. The forth language can solve the problem, all of users do not need to learn each of assembly language but just learn forth assembly language. It will be helpful for teacher because it is similar like mathematical language. For the programmer, it is much easy to understand the codes meaning in a short time. If one who did not know about ARM's instructions, but he/she has learned the forth assembly language, he/she can use the forth assembly source codes to assemble to different machine codes. [8]

TABLE I. FORTH LANGUAGE OPERATOR TABLE

Operator	Forth Assembly Language
Plus/Add	+
Minus/Subtract	-
Multiple	*
And	&
Or	|
Not	/
Exclusive - Or	^
Right Shift	>>
Left Shift	<<
Rotation Right Shift	>|>

TABLE II. BRANCH

Meaning	Forth Assembly Language
Equal	==
Not Equal	!= or <>
Signed Greater Than or Equal	>=
Signed Less Than	<
Signed Greater Than	>
Signed Less Than or Equal	<=

The Table I and Table II are the branch for the forth assembly language used. The Table III is an example for ARM cortex M0 instructions and forth assembly language table [9], [10]. On the first column is ARM cortex M0 instruction opcode, the second column is its assembly language, and the third column is forth system language. It can be found that the forth language is translated to machine codes directly without any intermediate codes and methods. We used the ARM cortex M0 instructions for example. Above of all the situations, it can be discovered that the forth assembly language is based on mathematical, algebra language and use some of spiritual of high-level programming language.

TABLE III. THE ARM CORTEX M0 INSTRUCTIONS VS FORTH ASSEMBLY LANGUAGE TABLE

Opcode (Hex)	ARM Assembly Language	Forth Assembly Language
2000	MOV_R,#	R=#
4600	MOV_R1,R2	R1=R2
5800	LDR_R1,[R2,R3]	R1=[R2+R3]
6800	LDR_R1,[R2,#]	R1=[R2+#]
4800	LDR_R,[pc,#]	R=[pc+#]
9800	LDR_R,[sp,#]	R=[sp+#]
5C00	LDRB_R1,[R2,R3]	R1=[R2+R3].b
7800	LDRB_R1,[R2,#]	R1=[R2+#].b
5A00	LDRH_R1,[R2,R3]	R1=[R2+R3].h
8800	LDRH_R1,[R2,#]	R1=[R2+#].h
5600	LDRSB_R1,[R2,R3]	R1=[R2+R3].sb
5E00	LDRSH_R1,[R2,R3]	R1=[R2+R3].sh
5000	STR_R1,[R2,R3]	[R2+R3]=R1
6000	STR_R1,[R2,#]	[R2+#]=R1
9000	STR_R,[sp,#]	[sp+#]=R
5400	STRB_R1,[R2,R3]	[R2+R3]=R1.b
7000	STRB_R1,[R2,#]	[R2+#]=R1.b
5200	STRH_R1,[R2,R3]	[R2+R3]=R1.h
8000	STRH_R1,[R2,#]	[R2+#]=R1.h
3000	ADD_R,#	R=+#
4400	ADD_R1,R2	R1=+R2
1C00	ADD_R1,R2,#	R1=R2+#
1800	ADD_R1,R2,R3	R1=R2+R3
A000	ADD_R,pc,#	R=+(pc+#)
A800	ADD_R,sp,#	R=+(sp+#)
4140	ADC_R1,R2	R1=+R2+C
3800	SUB_R,#	R=-#
1E00	SUB_R1,R2,#	R1=R2-#
B080	SUB_R1,R2,R3	R1=R2-R3
4180	SBC_R1,R2	R1=-R2-C
4340	MUL_R1,R2	R1=*R2
43C0	MVN_R1,R2	R1=/R2
4000	AND_R1,R2	R1=&R2
4240	NEG_R1,R2	R1=-R2
4300	ORR_R1,R2	R1=|R2
4040	EOR_R1,R2	R1=^R2
4380	BIC_R1,R2	R1=@R2
0000	LSL_R1,R2,#	R1=R2<<#
4040	LSL_R1,R2	R1=<<R2
0800	LSR_R1,R2,#	R1=R2>>#
40C0	LSR_R1,R2	R1=>>R2
1000	ASR_R1,R2,#	uR1=R2>>#
4100	ASR_R1,R2	uR1=>>R2
41C0	ROR_R1,R2	R=>|>R
2800	CMP_R,#	R:#
4280	CMP_R1,R2	R1:R2
4500	CMP_R1,R2(h)	R1:R2(h)
42C0	CMN_R1,R2	R1:-R2
F800	BL_#	JUMP_#
4700	BX_R	JUMP_R
F000	BL{X}_#	JUMP{X}_#
4780	BLX_R	CALL_R
E800	BLX_#	CALL_#
BE00	BKPT_#	BREAK_#
4200	TST_R1,R2	TEST.R1&R2
BC00	POP_{Rlist,pc}	POP{Rlist,pc}
B400	PUSH_{Rlist,lr}	PUSH{Rlist,lr}
C800	LDMIA_R,{Rlist}	POP_R,{Rlist}
C000	STMIA_R,{Rlist }	PUSH_R,{Rlist}
DF00	SWI_#	SWI_#

C. *Assembler Method and Syntax Defination*

The assembler algorithm has two states called Pass 1 and Pass 2, also this forth language supports "if", "else if", "else", and "while" syntax. But it is not like higher programming language, it will be assembled directly to the shortest and simplest machine codes. This will be much easy to establish applications, drivers, and also accelerate the execution. In addition, it will record the syntax with the data structure – queue. [11]

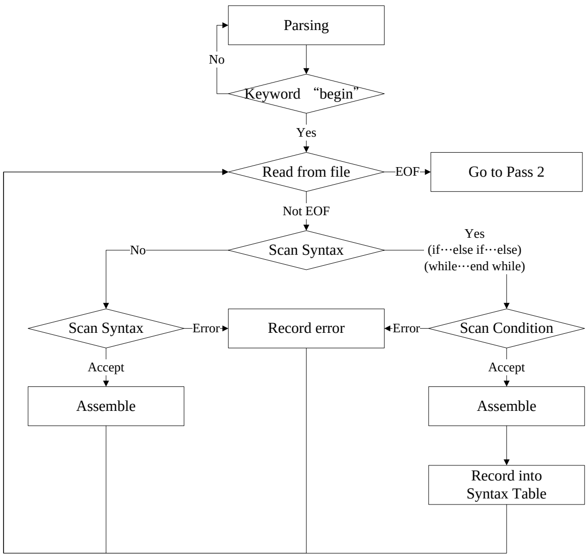

Figure 1. Pass 1 of the assembler

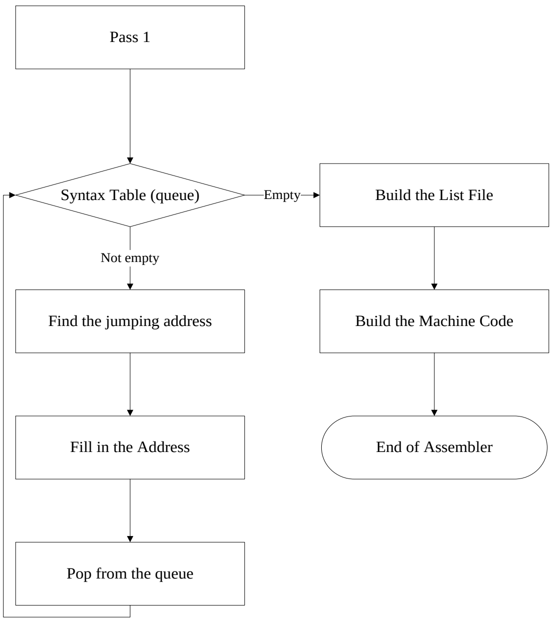

Figure 2. Pass 2 of the assembler

Pass 1 (see Fig. 1).

Step 1. Parsing the source codes.

Step 2. Find the keyword "Begin", it means the beginning of the source codes.

Step 3. Assemble the source codes. If this assembler found that it has read the end of the file, then go to the Pass 2. Otherwise, go to the step 4.

Step 4. Find syntax. If it read "if", "else-if", "else", "end-if", "while", "end-loop", it would record the address and syntax.

Step 5. Find operator. It would find match the operator and branch table.

Step 6. Assemble the source codes to hex codes, then go to step 3. (If it finds the codes are error, it would not assemble and record the line of the source codes. Finally, it would let user know that where and what it happened.)

Pass 2 (see Fig. 2).

Step 1. Get the syntax table from the queue.

Step 2. Build the list file.

Step 3. Build the machine codes.

Step 4. End of the algorithm.

TABLE IV. STEP 1 SOURCE CODES VS MACHINE CODES

Line	Address	Machine Codes	Source Codes
0	00004970	2D E9 00 40	push.LR
1	00004974	45 F2 FC 32 C4	pf.3=0
2	00004984	08 23	R3=8
3	00004986	56 21	R1=56
4			While z=1
5	00004988	FF F7 EC FF	delay.R1
6	0000498C	5F EA 36 06	T rrc.r
7	00004990	08 D2	If c>=0
8	00004992	45 F2 FC 32 C4	pf.3=0
9	000049A2	07 E0	Else
10	000049A4	45 F2 FC 32 C4	pf.3=1
11			End If
12	000049B4	01 3B	R3-1
13	000049B6	E7 D1	End Loop
14	000049B8	FF F7 D4 FF	delay.R1
15	000049BC	45 F2 FC 32 C4	pf.3=1
16	000049CC	FF F7 CA FF	delay.R1
17	000049D0	01 26	T=1
18	000049D2	00 F0 25 F9	#emit
19	000049D6	00 F0 73 FB	+!
20	000049DA	00 BD	ret

TABLE V. TO ILLUSTRATE THE STEP 1'S SOURCE CODES

Line 0	Let the next instruction push into return stack.
Line 1	Output the first bit of RS232.
Line 2	The value "8" is ASCII-codes (8-bits).
Line 3	Set the count of delay.
Line 4	The while-loop start.
Line 5	Start the delay routine, the time is set by the register 1 (R1).
Line 6	Let register T shift right
Line 7	If the carry is 0.
Line 8	Output low potential from f bit 3 port.
Line 9	Else.
Line 10	Output high potential from f bit 3 port.
Line 11	End of If.
Line 13	End of while-loop.
Line 14	Start the delay routine, the time is set by the register 1 (R1).
Line 15	Set the stop bit to high potential.
Line 16	Start the delay routine, the time is set by the register 1 (R1).
Line 17	Set the register T.
Line 18	(System variable, to remember the number of the output byte)
Line 19	Add and push back.
Line 20	Return

D. Token and Scan Operator and Syntax

When the assembler read the source codes, if will first find the operator whether it would match the language table or not. If it matched the instruction, it would assemble to the machine codes directly. If it does not, it will compare whether it is “if”, “else if”, “else”, “end if”, “while loop”, “end loop” keywords. If it is a keyword, it would record the line of the codes and push the data into the syntax queue. [12]

E. About Example and Tiny Forth System

These source codes in forth language is to build and set up a tiny forth system.

Step 1. Generate the command “emid” from RS232 through the I/O hight/low, and then use the port “TX” signal for example (from the Table IV and Table V).

Step 2. Design the command “key” (with the pin RX from RS232). Because the “key” and “emit” are matched, it should be set the same delay time (from the Table VI and Table VII).

This part of source codes in forth language is for using tiny-forth system. And then, this tiny forth system can be used to build a huge, large forth system. Some of umbilical or tethered systems in business used this concept to accomplish the enormous and extremely complex systems. Also, part of architectures through the tiny system communicates others. This system looked small but it can be a break point for developing and researching in computer, or microprocessors.

TABLE VI. STEP 2 SOURCE CODES VS MACHINE CODES

Line	Address	Machine Codes	Source Codes
0	00004A58	04 3F 3E 60	{dup}
1	00004A5C	00 B5	push.LR
2	00004A5E	B6 1B	T=0
3			While z=1
4	00004A60	FF F7 F2 FF	c=rx
5	00004A64	00 28	R0:0
6	00004A66	FB D1	End Loop
7	00004A68	56 21	R1=56
8	00004A6A	48 08	R0=R1>>1
9	00004A6C	FF F7 7C FF	delay.R0
10	00004A70	08 23	R3=8
11			While z=1
12	00004A72	FF F7 77 FF	delay.R1
13	00004A7A	4F EA 70 00	R0>>>1
14	00004A7E	76 08	T>>1
15	00004A80	46 EA 00 06	T|R0
16	00004A84	01 3B	R3-1
17	00004A86	00 2B	R3:0
18	00004A88	F3 D1	T>>18
19	00004A8A	36 0E	delay.R1
20	00004A8C	FF F7 6A FF	ret
21	00004A90	00 BD	nop
22	00004A92	00 BF	c=rx
23	00004A48	45 F2 04 00	R0=5004
24	00004A4C	C4 F2 02 00	R0t=4002
25	00004A50	00 68	R0=(R0+0)
26	00004A52	00 F0 01 00	R0=R0&1
27	00004A56	70 47	ret

TABLE VII. TO ILLUSTRATE THE STEP 2’S SOURCE CODES.

Line 0	To get the byte data and record to stack.
Line 1	Let the PC counter point to next instruction.
Line 2	Set register T to 0.
Line 3	The while-loop start.
Line 4	Let the port TX (from desktop) to the microprocessor’s RX input port.
Line 5	Compare R0.
Line 6	End of while-loop.
Line 7	Set the count of delay.
Line 8	Shift right.
Line 9	Start the delay routine, the time is set by the register 0 (R0).
Line 10	Set the value of R3.
Line 11	The while-loop start.
Line 12	Start the delay routine, the time is set by the register 1 (R1).
Line 13	Let the R0’s the bit 0 to set the carry flag.
Line 14	Shift right.
Line 15	Or operator.
Line 16	Subtract.
Line 17	Compare R3.
Line 18	Shift right.
Line 19	Start the delay routine, the time is set by the register 1 (R1).
Line 20	Call the return.
Line 21	Cell for 4 address.
Line 22	Set R0. Because most of CPU let the import from the RS232 store in the R0.
Line 23	Store the value into register (lower bytes).
Line 24	Store the value into register (higher bytes).
Line 25	Add.
Line 26	And operator.
Line 27	Return.

IV. CONCLUSION

The traditional assembly languages (such as x86, ARM, MIPS) need to cost more time than to learn forth assembly language. Someone who has ever learned about C, C++, or high-level programming language, it would understand the forth language in a short time. Also, the traditional assembly languages are difficult for the beginners who have never learned about computer science.

From now on, there are countless assembly languages and instructions. If it could be established the standard of assembly language, this will help beginners to learn, programmers to develop, and teachers to teach. In the future, it can be designed a general compiler to compile the high level languages to the forth language. Therefore, it is not necessary to develop compilers for processors, respectively. If the goal is achieved, it will create a new computer research area.

ACKNOWLEDGMENT

The authors would want to thank Hsing-Yao Huang (received the master degree from National Taiwan University, ME in 1955) and Po-Chun Tsou to help the author complete this paper and give illustrations. Especially the ARM cortex M0 vs Forth Language table, he gave a good suggestion for establishing and editing this.

REFERENCES

[1] J. R. Hayes, M. E. Fraeman, R. L. Willians, and T. Zaremeba, “An architecture for the direct execution of the forth programming language,” in *Proc. ASPLOS II*, Oct. 1987, pp. 42-49.
[2] Leo Brodie, *Starting Forth Run on iForth and SwiftForth*, 1st ed. Forth Inc., 2007, ch. 2
[3] Leo Brodie, *Thinking Forth: A Language and Philosophy for Solving Problems,* 1st ed., Forth Inc., 2007, ch. 2.
[4] K. R. Irvine, *Assembly Language for Intel-Based Computer*s, 5th ed., Pearson, Sep. 2006, ch.4, 6, 7.
[5] L. Wittie and G. Shute. (Sep. 2010). *MIPS Instruction Coding*. [Online]. pp. 1-4. Available: http://www.cs.sunysb.edu/~lw/spim/MIPSinstHex.pdf
[6] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 4th ed., Elsevier Taiwan LLC., Oct. 2010, ch. 2.
[7] Standard Forth, ANSI INCITS 215-1994.
[8] H. M. Martin, “Developing a tethered forth model,” *ACM SIGFORTH Newsletter*, vol. 2, no. 3, pp. 17-19, Mar. 1991.
[9] Gbadev. (Sep. 2001). *ARM Thumb Reference*. [Online]. Available: http://re-eject.gbadev.org/files/ThumbRefV2-beta.pdf
[10] *Atmel 8051 Microcontrollers Hardware Manual*, Atmel Corporation, 2007, ch. 1.
[11] L. L. Beck, *System Software: An Introduction to Systems Programming*, 3rd ed., 1996, ch. 5.
[12] M. T. Goodrich and R. Tamassia, *Algorithm Design: Foundations, Analysis, and Internet Examples*, John Wiley & Son, 2006, ch. 2.

Gang-Jeng Huang is studying at the department of information engineering, Chung Cheng Institute of Technology (CCIT), National Defense University (NDU). His experience includes the web site designer and database administrator for the 22nd National Defense Science and Technology Conference. He also got the special award from the Taiwan Summer of Code (TSOC), and its project is simplified universal assembler.

Shiuh-Ku Weng received Ph.D degree from Chung Cheng Institute of Technology (CCIT) of National Defense University (NDU) in EE in 1997. Now, he is an associate professor in Department of Information Engineering, CCIT, NDU. His research interests include embedded system, image processing, information security and system programming.

Jyi-Jinn Chang received BS degree from National Taiwan Institute of Science and Technology, in EE. He is working at Science Based Park, Hsichu, Taiwan. His expertise includes Instrument design and research, microcomputer related facilities control firmware design, and the forth language development for using in large vacuum coating deposition system.