A Two-Pass Assembler for the AMIR CPU Microprocessor

M.N. Ibrahim, M.Idroas, J.Kassim N. Azhari, A. Baharum E.B.L. Eline

ABSTRACT--- ***Today's demands for high performance computing on embedded devices call for high performance microprocessor. There are two families of most popular microprocessors namely Intel and ARM. However, these microprocessors have lots of limitations due to their architectures, which are inherited from older versions. The results are their assembly language become too complicated to be used for writing useful applications, hence high level languages were promoted, which is inefficient for hardware access. The AMIR CPU, which was developed at the Advanced Microprocessor Research Laboratory, Universiti Teknologi Malaysia was designed after careful study of weaknesses of present microproceesors both in terms of hardware architecture Instruction Set. A new two-pass assembler was developed to convert the AMIR CPU assembly language to its corresponding machine codes. A few application programs were written and successfully tested working on a DE0 FPGA board containing the AMIR CPU.***

Keywords - AMIR CPU, Microprocessor, Two-Pass Assembler, FPGA

INTRODUCTION

A microprocessor is an electronic component that acts as the brain of many moderns. electronic equipments. The history of microprocessor began in 1974 with the introduction of of the 4-bit Intel 4004, which was soon improved to 8-bit 8080, followed by 8085 [Z. Stachniak 2010]. Within a few years later a few other other companies emerged with their own microprocessors. Among them include Zilog's Z80, MOS Technology's 6502, Motorolas 6800 [Betker, Fernando & Whalen 1997]. The introduction of IBM PC, displacing many mainframe computers revolutionized microprocessor industry, with Intel immediately became one of the world's biggest corporation as IBM chose Intel microprocessor family known as x86 for their PC. The x86 family was continually being ugraded over many years, benefited by large user base of PCs. Even the present highly promoted Intels multicore such as i7 and i9 are derivatives of older x86 family traced back to 1970's, due to Intel's strategy of maintaining downward compatability. As a consequence, their microprocessor are based on old architecture, unfit for future software demands.

There is another type of microprocessor applications, a non-PC ones, broadly known as embedded systems, which include among others home and medical equipments, automation, handphones, and instrumentation. In this area the ARM is the microprocessor which control about 90% of world's usage. The ARM microprocessor was developed in 1990, much later than the first product by Intel. Many improvents were made in terms of power efficiency, number of registers, Triadic type instruction, and other aspects. However, the ARM processor still lacks features essential for future software programming. The IEEE Computer Society has introduced a family of standard for future software called the Portable Operating System Interface (POSIX) in 2008 to ease software developers among different organizations. [Xiocong Fan, 2015]. Some of the issues covered include mulitasking, multithreading and handling of many daemons lke Windows. ARM processor was developed before the POSIX standard was published hence become inefficient to be compatible with POSIX. The Intel x86, produced much earlier, hence more incapable.

There is another class of processor called softcore microprocessor. These processors are not fabricated on integrated circuits rather resides on Field Programmable Gate Array (FPGA). Their disadvantage are mostly tied to fixed type of FPGA, hence cannot be easily transferred to other FPGA. Examples of softcore microprocessor include Altera Nios, Xilinx MicroBlaze, ARM Cortex and LEON.

When AMIR CPU began its development in 2010, many of the weakness of previous processor were avoided, their best features included, and POSIX compatibility taken into account. As a result the AMIR CPU was built based on a very good hardware architecture and excellent Instruction Set [Ibrahim, Azhari, and Baharum, 2017 (2)]. Its hardware advantages include:

- Large linear registers
- Fast I/O – memory transfer eliminating DMA
- Multi-platform capability

Whereas, its software advantages include:

- Fully orthogonal instructon set
- Burst transfer mode
- Small set of easy to use assembly language.

[Ibrahim, Azhari, and Baharum, 2017 (1)].

PROBLEM STATEMENT

In order to prove the capabilities of the AMIR CPU, one need to interface it with outside world, program the embedded software, and test the output in real time. There is however no assembly or high level language to program the processor even though the Instruction Set is completely well-defined, together with corresponding machine codes.

Revised Manuscript Received on February 14, 2019.

M.N. Ibrahim, Faculty of Engineering, Universiti Teknologi Malaysia.

M.Idroas, Faculty of Engineering, Universiti Teknologi Malaysia.

J.Kassim N. Azhari, Faculty of Engineering, Universiti Teknologi Malaysia. (E-mail: mdnasir@utm.my)

A. Baharum E.B.L. Eline, Faculty of Engineering, Universiti Teknologi Malaysia.

To design high level language is not a good choice since it slows down the processor. Moreover, its assembly language as claimed by the designers are optimized yet easy to use thus making the high level language unnecessary.

THE AIM OF RESEARCH

The research aims to develop a new 2-pass assembler for the AMIR CPU. Once completed, application software will be be written to test the capabilities of the AMIR CPU.

METHOD OF RESEARCH

For the software part, the design of the two pass assembler is developed by using high level language, C++ and compiled by using Dev C++ software. The program are divided into two parts, which are the first pass and the second pass. For the first pass, the assembler will find out the label as well as its address of each instructions and a table will be created in intermediate.txt. While for the second pass, the assembler will generate the machine code and displayed in MIF which is denoted as x.mif. The flowchart of Figure 1 illustrates working concept of the two-pass assembler.

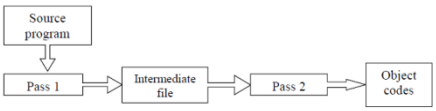

Figure 1

Basically, in pass 1, each line of code in the source file is being scanned in order to find out the labels within the codes. If a label is found, then the label's name and its value will be stored in the symbol table. If normal instruction is found, the assembler will determine its corresponding size of the instruction. In order to keep track of instruction locations, the assembler uses a memory word called a location counter (LC). LC stores the value of the memory location assigned to the instruction or operand presently being processed. After the size of instruction is being determined, the LC increases by the size of the instruction. Then, pass 1 ends with the source line and other information being written onto an intermediate file which is denoted as intermediate.txt. This process loops until the end of source file is detected. Figure 2 shows the operation of pass 1 of the assembler.

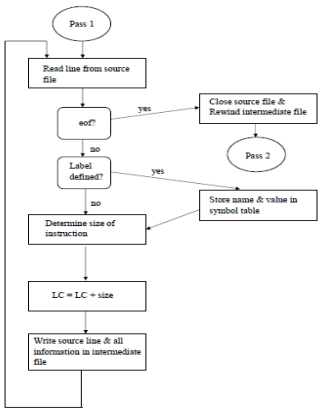

Figure 2

In the second pass, the assembler reads each line from the intermediate file produced in the first pass until the end of line of the file. All the instructions will be assembled where the machine code will be written in x.mif. A file with the extension .mif is an ASCII text file that defines the initial content of a memory block which means the initial values for each address and this file is used during the project compilation in the hardware part. This process will repeat until all the lines in the intermediate file has been read.

Figure 3 shows Pass 2 of the assembler.

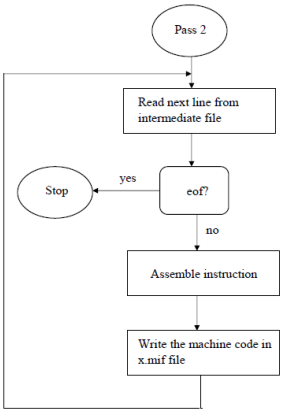

Figure 3

As for the hardware part, a DE0 board containing Altera Cyclone III was used. Three application programs were written and tested. The first one involves turning ON and OFF of the LED. The second one use a 2-digit 7-segment display to show counter values in decimal. Finally for more difficult level application, the GPIO port on the DE0 board was connected to a DC motor via motor driver IC, L293D. Interfacing to the DE0 board was done according to the board's specificatios, as shown in Figure 4

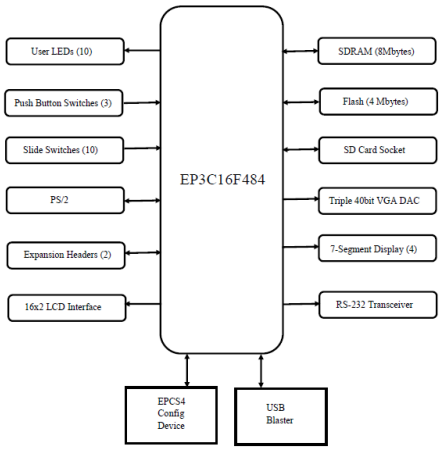

Figure 4. [Altera Corporation]

RESULTS AND DISCUSSION

A 2-pass assembler for the AMIR CPU was successfully developed using Dev C++ compiler. Each of 33 assembly language instructions was tested, the results of which matched with manual assembly based on. An example of Input file, Intermediate file, and resultant machine codes are shown in Figure 5, 6, and 7, respectively.

```
mil r6=0xFF0F          --16-bit port output address
mi rA0=0x03F0          --load 32-bit sp value to A0, 3F0 =1008
m rA0,sp               --initialize stack pointer
mi r1=0x0001           --initialize constant
mi r2=0x0201           --initialize constant
AGAIN: xor r9,r9,r9    --initialize r9=0
DISP_COUNT: m r9,#r6   --Display LED
call DELAY
add r9,r1,r9           --increment r1, store result in r9
sub r2,r9,r9           --decrement r9, store result in r9
x=DISP_COUNT           --jump to branch DISP_COUNT
j AGAIN
DELAY: mi r10=0x100000 --load 32-bit DELAY counter
sub r10,r1,r10         --decrement counter
x=0xFFFF               --if x= (-1), return to the subroutine
ret
```

Figure 5

Label	Mnemonic	Operand	Address
	mil	r6=0xFF0F	0
	mi	rA0=0x03F0	2
	m	rA0,sp	3
	mi	r1=0x0001	5
	mi	r2=0x0201	7
AGAIN	xor	r9,r9,r9	8
DISP_COUNT	m	r9,#r6	9
	call	DELAY	A
	add	r9,r1,r9	B
	sub	r2,r9,r9	C
	x	DISP_COUNT	D
	j	AGAIN	E
DELAY	mi	r10=0x100000	11
	sub	r10,r1,r10	12
	x	0xFFFF	13
	ret		14

Figure 6

```
WIDTH=32;
DEPTH=1024;
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN

  000  :  0306FF0F;
  001  :  05A00000;
  002  :  000003F0;
  003  :  06A00000;
  004  :  05010000;
  005  :  00000001;
  006  :  05020000;
  007  :  00000201;
  008  :  10090909;
  009  :  0A090600;
  00A  :  16000005;
  00B  :  12090109;
  00C  :  13020909;
  00D  :  1BFFFFFC;
  00E  :  1AFFFFFA;
  00F  :  05100000;
  010  :  00100000;
  011  :  13100110;
  012  :  1BFFFFFF;
  013  :  17000000;
  [014..3FF]  :  00000000;
END;
```

Figure 7

The AMIR CPU Instruction Set Architecture contains 4 groups of instructions as listed below [Ibrahim, Azhari, Baharum (2), 2017]:

1. Data Movement.
2. Logical & Arithmetic.
3. Procedure Call.
4. Execute or Branch.

As an example of how the machine code is generated, refer to to Figure 1 Line1 which is:

mil r6=0xFF0F

which means “Move immediate 0xFF0F into the low 16-bit word of register r6 without affecting its upper 16-bits)”. According to AMIR CPU ISA, this instruction can be assembled according to Figure 8.

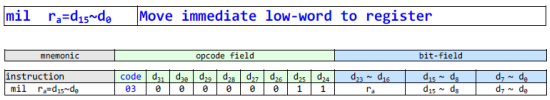

Figure 8 [Ibrahim. Azhari, Baharum]

The machine code (in binary) for this instruction should be as follows:

opcode = 0b00000011

bit field for Register a = 0b0110

bit field for data = 0b1111111100001111.

By combining the 3 values above, the machine code generated in hexadecimal should be 0x0306FF0F, placed at memory location 0. The assembler produced the expected result as shown on line 1 of Figure 7.

The rest of other instructions follow the same explanation. For the j (“jump”) instruction the offset from the label was calculated using 2’s complement number and put as machine code accordingly.

CONCLUSION

This research aims to develop a new assembler for the AMIR CPU. A new assembler was built and tested on the AMIR CPU residing in a DE0 FPGA board.The objectives were successfully achieved. This work helps future application developers of the AMIR CPU applications.

ACKNOWLEDGEMENTS

The authors wish to acknowledge Universiti Teknologi Malaysia for supporting this project through an internal research grant Vot No. 14J98.

REFERENCES

1. Betker M. R., Fernando J. S. and. Whalen S. P, "The history of the microprocessor," in Bell Labs Technical Journal, vol. 2, no. 4, pp. 29-56, Autumn 1997.
2. Ibrahim, M.N, Azhari, N, Baharum,A. (1) 2017, AMIR-1 32-Bit Microcontroller Instruction Set Architecture, Pahlawan Micropemproses Sdn. Bhd.
3. Ibrahim, M.N, Azhari, N, Baharum,. (2), 2017, AMIR-1 32-Bit Microcontroller Technical Reference, Pahlawan Micropemproses Sdn. Bhd.
4. Stachniak Z.. Oct.-Dec. 2010, "The MIL MF7114 Microprocessor," in IEEE Annals of the History of Computing, vol. 32, no. 4, pp. 48-59
5. Xiaocong Fan, 2015, “Chapter 3 – POSIX and RTOS”, Real-Time Embedded Systems – Design Principles and Engineering Practices: 339-368.