Comparison of emulation oriented 8-bit ISA with 6502 ISA for an ARM emulator

Hao Xu[1] Yuko Hara-Azumi[1] Yasuhiko Nakashima[1]

Abstract: In order to implement processors using new-materials like Thin Film-Transistors (TFTs), we have defined an 8-bit ISA for 32-bit ARM emulator. This paper compares two emulators for our 8-bit ISA and 6502. By executing 32-bit ARM programs on the emulators, we demonstrate that our instruction emulator outperforms the 6502 instruction based emulator. Moreover, some of our instructions that are implemented in a single instruction, instead of several instructions in 6502 lead to reduce the size of the emulator. Especially, our self-defined table branch instructions that have 16-way jump and 2-way checking function reduce the emulation cycles for one ARM instruction.

1. Introduction

Recently, Oxide Thin Film-Transistors (TFTs) have emerged as devices that exhibit higher mobility compared to amorphous silicon transistors. Among Oxide TFTs, ZnO-based TFTs have attracted increasing attention for flexible display applications because highly uniform, large-area displays can be fabricated on plastic substrates at low temperature, resulting in low production costs [1]. With the rapid development of this fabrication technology in integrated circuits, a new material for ZnO-based TFTs is being studied by the Material Department of NAIST. Using that technology, an integrated circuit will be printed on the new material sheet just as with inkjet printing.

However, when we use the inkjet printing technology to implement the processors on the new material, the incidence of circuit malfunction is higher than that of traditional silicon material. In order to achieve higher reliability employing conventional modular redundancy techniques, it is essential to realize a simple architecture processor that has a small scale. To solve this issue, processors using smaller bits (e.g., 8 bits) are required for decreasing the incidence of malfunctions. There have been multiple 8-bit processors in the late 1970s and early 1980s, such as Motorola 68XX and Intel 80XX. However, these 8-bit processors are not efficient to use today as most of today's software programs are implemented for 32/64-bit processors, e.g., ARM processors, and thus, they are not portable to be executed on such 8-bit processors. Consequently, we have proposed an 8-bit processor which can emulate 32-bit ARM applications [2]. We introduced several self-defined instructions to efficiently execute 32-bit ones, with which we demonstrate that it can successfully execute several ARM applications. However, we have not yet compared our proposed processor with aforementioned existing 8-bit ones for quantitatively evaluating its effectiveness.

In this paper, we compare our proposed processor with an existing 8-bit one for emulating 32-bit ARM applications. 6502 ISA was selected as a representative counterpart for the evaluation. We evaluate our self-defined emulator against the 6502 ISA emulator in order to see its effetiveness in reducing the emulator size and step numbers. From the results, we observe that our self-defined instructions are more useful than several 6502 instructions. Especially, table jump instructions play a crucial role in reducing the step numbers.

The remainder of this paper is organized as follows: Section 2 gives an overview of related work. The study of self-defined instruction set and comparison on building an emulator between the self-define ISA and 6502 ISA will be shown in Section 3. In Section 4, we evaluate the performance of the two processors executing a test program. Finally, Section 5 provides summary and directions for future work.

2. Related Technology

In this section, the ARM ISA, 6502 ISA and emulator technology are explained in three subsections.

2.1 ARM ISA

Fig. 1 shows the typical instruction format of ARM architecture. The "S" bit of calculation instruction set (i.e., ALU (+Shift) and Multiply) decides if the conditional code is updated or not. The branch instructions, which use the lower part of the instruction, can reduce the branch misprediction penalty. The operand Sft in Load/Store and ALU (+shift) instructions represents a shift operation. The instructions with Sft operand can make one ARM instruction as useful as two RISC instructions. In addition, the load/store instructions can update the base register in one instruction. In a RISC instruction set, this takes three instructions. In short, an ARM instruction which has four bytes can execute the multiply function in one instruction [3]. The ARM processor is widely used in various systems, e.g., mobile phones, these years.

[1] Nara Institute of Science and Technology

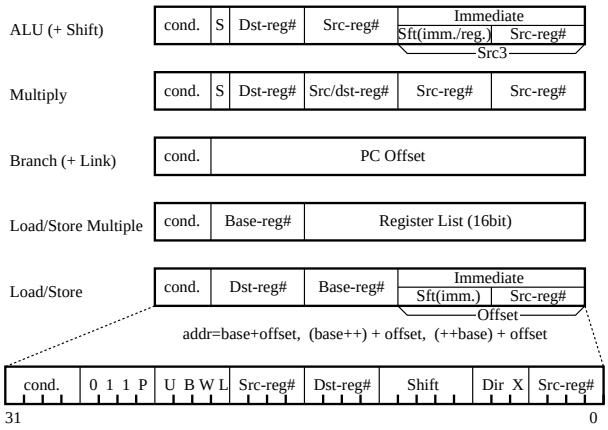

Fig. 1 ARM Instruction Set

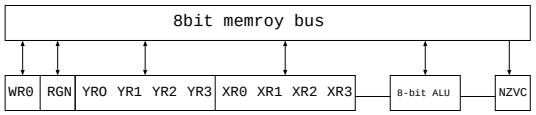

Fig. 2 Partial Block of Self-defined ISA Processor

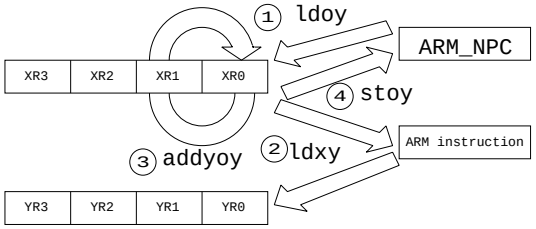

Fig. 3 Procedure of Fetching Guest Instructions

2.2 6502 ISA

The 6502-CPU is a relatively simple design microprocessor architecture for embedded systems. The 6502-CPU has a 16-bit address bus and can access up to 64Kb of memory. The "zero page" address mode accesses memory locations from address 0x00 to 0xFF. These addresses are used as well as the registers to load and store the operands.

From a structure point of view, the 6502 processor is a stack-based processor. The stack range based on the first-in-last-out role is from 0100 to 01FF in memory. The accumulator is the only register that is able to do the arithmetic and logic operation. The two index registers X and Y can be treated as auxiliary-general purpose registers, having the added ability of being incremented and decremented by the normal operations[4].

2.3 Emulator Technology

Emulator is a technology to accurately imitate another computer and run software from that computer. Virtually every home computer system that has been ever created has been emulated. Emulator is also similar to an interpreter and works in a similar way though an emulator does far more than just interpret instructions.

In [2], we have proposed an emulator to emulate ARM processors on an 8-bit processor such as our proposed 8-bit processor and 6502 processor. As mentioned before, ARM instruction format has 32-bit and can be divided into four 8-bit registers.

When 8-bit processors emulate ARM processors, 4-byte load/store and arithmetic calculations are necessary. The 6502 ISA registers, which are 1-byte, are not able to load 4-byte ARM instruction in one cycle. Thus, several methods are employed to load the 4-byte instructions in 6502 ISA. For example, the zero page mode is treated as registers to load the ARM instruction temporarily. The zero page mode instructions are used to decode the ARM instructions. This method takes the large number of cycles to load and store the 4-byte ARM instructions. Another method is to use the stack in order to load and store the ARM instruction temporarily. In this case, it also takes many instructions to control the stack pointer, but can reduce the cycle number than using zero page at the cost of increasing the emulator size.

3. Fundamental Improvement of Key Components

In this section, we briefly review our self-defined ISA which was proposed in [2]. Additionally, we show comparisons of our self-defined ISA and 6502 ISA for emulation.

3.1 Self-defined ISA

Fig. 2 shows the partial block diagram of our 8-bit processor. The 4-byte X and Y index registers are used to load the ARM instructions in one cycle. The index registers are performed singularly by four 1-byte registers. The work register (WR0) is similar to the accumulator on which operations are performed in 6502. The RGN register is a 1-byte register that stores the ARM register number. The conditional register contains 4-bit flags that are Negative, Zero, overflow and Carry. The 8-bit ALU gets operands from registers through an 8-bit memory bus. To make a clear distinction between the ARM processor and our self-defined processor, hereafter, we call the ARM processor "guest" and our self-defined processor "host". Using this register structure, our processor is comfortably able to load the guest PC (4-byte) and guest instructions (4-byte). **Fig. 3** shows the procedure of fetching guest instructions:

- STEP1: The 4-byte registers YR load the 32-bit guest PC
- STEP2: According to the guest PC, fetch the guest instructions in XR registers.
- STEP3: YR registers are added to update guest PC
- STEP4: Copy the value in YR registers to ARM_NPC.

In the case of ARM, a plurality of instructions such as load/store or calculation instruction operand codes position are often completely different, while operands are the same. In the original ARM instruction, the opcode and shift immediate operand span two 8-bit registers when we divide the 32-bit instruction to four 8-bit registers. In this case, more host instructions are requested to get the opcode and operands in execution stage. **Table 1** shows a method to convert the internal instruction order when the guest instructions are decoded once, which is able to reduce the host instruction numbers in execution stage, in other words, reducing the size of the emulation program.

For detail information of self-defined ISA, please refer to [2].

Table 1 Internal Instruction Format by Decoding Once

Address	Field(bits)	Field(bits)	Field(bits)	Field(bits)
0x00	src1 (4)	src2 (4)		
0x01	dst1 (4)	dst2 (4)		
0x02	sftc (1)	wb (1)	pre (1)	sop (1)
	opcd (4)			
0x03	type (3)	updt (1)	cond (4)	
0x08	list (16)	as same as Register List in Fig.1		
0x0c	src3 (32)	as same as Src3/Offset in Fig.1		

3.2 Comparison of Self-defined ISA and 6502 ISA on Emulator

As mentioned in Section 2.3, we use 6502 ISA as a counterpart of our self-defined ISA as emulating ARM applications (version 4 ARM ISA). The ARM instructions are handled in five stages: instruction fetch stage, instruction decode stage, instruction execution stage, memory access stage and write back stage. In the following, only the first three stages will be discussed and compared between the 6502 ISA and our self-defined ISA.

3.2.1 Instruction Fetch Stage

In this stage, the host register loads and procedures the guest PC to load the guest instructions. Then, the guest PC is updated by 4. In **Table 2**, the columns B, C, L, I and O represent byte numbers, cycle numbers label name, instruction name and operand name, respectively. 10 columns from the left shows the behavior of a partial fetch stage of 6502 ISA emulator, and the remaining five columns represent that of our self-defined ISA emulator. The address mode of ARMNPC, TEMPI and TEMPII is zero-page. We defined the ARMNPC to store the next PC of ARM. The TEMPI and TEMPII are used to load the guest instructions as registers. The 6502 ISA emulator takes 35 bytes, 54 cycles and 19 steps to implement this partial fetch stage. In the right side of Table 2, our self-defined ISA takes 7 bytes, 16 cycles and 4 steps to implement the same partial fetch stage.

3.2.2 Instruction Decode Stage

In this stage, our self-defined ISA is used to perform the decode stage of ARM processor and build the instruction format as shown in Table 1. In case of ARM, firstly the decoding process estimates the instruction type by the value of the main opcode. For example, the bit [27-25] is decoded to estimate the instruction type (e.g., data processing type, load/store type, software interrupt, etc). Then, the decoding process searches the sub-opcode position. For example, if the instruction type is the data processing type, the bit [24-21] is decoded to estimate the operation. In order to make the decode work more efficiently, we divide the instruction that is stored in XR register into eight 4-bit fields. In addition, we have introduced a group of instruction set called "table jump" that is able to use the 4-bit field to implement 16-way table jump operations. The table jump instruction also has the function that is able to implement a 2-way select operation by checking one bit of the field. **Fig. 4** shows the details of the table jump function.

In the figure, the instruction tbx2h means: get the high 4-bit of XR2 register for table jump. The hex number A corresponds the options of table jump. There are four options as follows.

- A is 0, so B is 0. At this time, the high 4-bit of XR2 leads to 16-ways table jump.
- When A is 1, the high 4-bit of XR2 does the logic OR arithmetic with hex number B. The way[1] is selected when the result is F. Otherwise, the way[0] is selected. This option is used for checking the identical position of each bit in high 4-bit of XR2.
- When A is 2, the high 4-bit of XR2 does the logic AND arithmetic with hex number B. The way[1] is selected when the result is 0. Otherwise, the way[0] is selected. This option is used for checking a single bit in high 4-bit of XR2.
- When A is 3, the high 4-bit of XR2 does the logic EOR (Exclusive OR) arithmetic with hex number B. The way[1] is selected when the result is 0. Otherwise, the way[0] is selected. This option is used for checking multiple bits in high 4-bit of XR2.

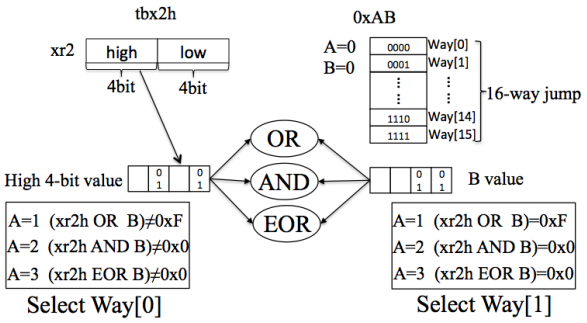

Fig. 4 Procedure of Table Jump Instructions

Table 3 shows a part of emulator operation in decode stage. The format of this ALU operation is shown in Fig. 1. The instruction type has already been decoded by table jump instructions in the earlier time of this stage. Table 3 shows the decoding process of the logic shift ALU operations in accordance of bit 6 and bit 5 in XR0. The method that builds the internal format instruction by decode once is also shown in Table 3. The left side uses the 6502 ISA, and the right side uses the self-defined ISA. In the right side, the instruction "mx32ow 7" means:

- shift the XR3 index register by the left 7 bits
- shift XR2 index register by 1 bit
- logic-OR the shifted XR3 index register and XR2 index register
- store the result of logic-OR to work register (WR0)

In the left side of Table 3, the first twelve 6502 instructions are used to perform the operation of "mx32ow" instruction. The ARM instruction is loaded from the "zero page" address TEMPII+0 to the "zero page" address TEMPII+3. TEMPII+3 represents the high 8 bits of ARM instruction which are equal to the 8 bits ARM instruction in XR3 index register. Firstly, the accumulator loads the 8-bit and shifts it by left 7 bits through seven "ASL" 6502 instructions. Secondly, the shifted result in the accumulator is stored to TEMP. TEMP is also a "zero page" address, which stores the operated result from the accumulator. Thirdly, the accumulator loads and shifts the 8-bit in "zero page" address TEMPII+2. Finally, the accumulator performs logic-OR with the "zero page" address TEMP by the "ORA" 6502 instruction. The operation of instruction "mx10ow" is generally the same as "mx32ow", but the registers are changed from XR2 and XR3 index registers to XR0 and XR1 index registers. The instruction

Table 2 Partial Fetch Stage Comparison

6502 ISA										Self-define ISA				
B	C	L	I	O	B	C	L	I	O	B	C	L	I	O
2	2		LDX	4	1	2		CLC		2	6		ldoy	ARMNPC
2	4	LOOP1	LDA	ARMPC-1	2	3		LDA	ARMNPC+0	1	5		ldyx	
2	4		STA	TEMPI-1,X	2	3		STA	ARMNPC+0	2	6		addyoy	4
2	4		STA	TEMPI-1,X	2	3		LDA	ARMNPC+1	2	6		stoy	ARMNPC
1	2		DEX		2	3		STA	ARMNPC+1					
2	2		BPL	LOOP1	2	3		LDA	ARMNPC+2					
2	4	LOOP2	LDA	TEMPI-1,X	2	3		STA	ARMNPC+2					
2	4		STA	TEMPII-1,X	2	3		LDA	ARMNPC+3					
1	2		DEX		2	3		STA	ARMNPC+3					
2	2		BPL	LOOP2										

Table 3 Partial Decode Stage Comparison

6502 ISA										Self-define ISA				
B	C	L	I	O	B	C	L	I	O	B	C	L	I	O
2	3		LDA	TEMPII+3	1	2		DEX		2	2		mx32ow	7
1	2		ASL		2	2		BPL	LOOP3	2	2		andwow	0xF0
1	2		ASL		2	3		LDA	TEMPII+0	2	3		sto2w	ARMDECODE0
1	2		ASL		2	2		AND	0xF0	2	2		mvx1ow	0xF0
1	2		ASL		1	2		LSR		2	2		srlwow	4
1	2		ASL		1	2		LSR		2	3		sto1w	ARMDECODE0
1	2		ASL		1	2		LSR		2	2		mvx2ow	0x0F
1	2		ASL		1	2		LSR		2	3		sto0w	ARMDECODE0
2	3		STA	TEMP	1	2		LSR		2	2		mvx0ow	0x0F
2	3		LDA	TEMPII+2	1	2		LSR		3	3		call	GETRGN
1	2		LSR		1	2		LSR		1	5		ldny	
2	3		ORA	TEMP	2	3		STA	TEMP	2	2		mx10ow	1
2	2		AND	0xF0	2	3		LDA	TEMPII+1	2	2		andwow	0x1F
2	3		STA	ARMDECODE0+2	1	2		ASL		4		tbx0l	0x00	
2	3		LDA	TEMPII+1	2	3		ORA	TEMP				.byte	D2ALULSL
1	2		LSR		2	2		AND	0x1F				.byte	D2ALULSL
1	2		LSR		2	3		STA	SHIFT5				.byte	D2ALULSR
1	2		LSR		2	3		LDA	TEMPII+0				.byte	D2ALULSR
1	2		LSR		1	2		LSR					.byte	D2ALUASR
2	3		STA	ARMDECODE0+1	2	2		AND	0x03				.byte	D2ALUASR
2	3		LDA	TEMPII+2	2	2		CMP	0x00				.byte	D2ALUROR
2	2		AND	0x0F	2	2		BEQ	D2ALULSL				.byte	D2ALUROR
2	3		STA	ARMDECODE0+0	2	2		CMP	0x01				.byte	D2ALULSL
2	3		LDA	TEMPII+0	2	2		BEQ	D2ALULSR				.byte	D2ALULSL
2	2		AND	0x0F	2	2		CMP	0x10				.byte	D2ALULSR
2	3		STA	ARMREGNUM	2	2		BEQ	D2ALUASR				.byte	D2ALULSR
3	6		JSR	GETARMREG	2	2		CMP	0x11				.byte	D2ALUASR
2	2		LDX	3	2	2		BEQ	D2ALUROR				.byte	D2ALUASR
2	4	LOOP3	LDA	ARMREG-1,X									.byte	D2ALUROR
2	4		STA	TEMPI-1,X									.byte	D2ALUROR

"ldny" represents that YR index register loads the data from the ARM register based on the ARM register number. The function of the "ldny" instruction in our self-defined ISA corresponds with that of the "LOOP3" loop instructions in the 6502 ISA. The operation of "ldny" is to transfer a 4-byte value between "zero page" address operations. The index register X is used as an address counter. It counts back from 3 to 0. Data is fetched from memory at the "zero page" address "ARMREG plus the value X" and placed in accumulator. The data is then written from accumulator to memory at the "zero page" address "TEMPI plus the value of Register X". Register X is decremented by one. If the decremented value is not negative, as the determined by BPL (branch on plus), the program loops back to LOOP3. The table jump instruction "tbx0h 0x00" in our self-defined ISA correspond with the last eleven instructions in the 6502 ISA. For calculating the byte numbers, the cycle numbers, and the step numbers of the same part of decode stage, our self-defined ISA takes 17 bytes, 28 cycles and 30 steps. Comparing with the byte numbers, the cycle numbers and step numbers in 6502 ISA, which are 95bytes, 142 cycles and 58 steps, respectively. Our self-defined ISA has significant benefits.

3.2.3 Instruction Execution Stage

In this stage, the format of Table 1 is used to perform the executing operation. As mentioned before, the original ARM processor has many 32-bit arithmetic operations. To implement these operations with the small number of bytes and cycles, a sequence of our self-defined instructions is created for 32-bit arithmetic logic operation. **Table 4** shows a comparison about a 32-bit substraction between 6502 ISA and our self-defined ISA. The instructions such as "addyoy" and "oryoy" are almost the same as the substraction. The 4-byte register with a 1-bit shift operation is the original logic shift operation of the ARM processor. **Table 5** shows an example of a 1-bit rotation instruction. The vacated-bit from rotation is temporarily stored in the carry flag of conditional register.

In addition, the ARM processor needs to write back the processor status to the conditional code register when the operation completes. At this time, it is necessary to get the status register.

Table 4 A 32-bit Calculation Comparison

6502 ISA					Self-define ISA									
B	C	L	I	O	B	C	L	I	O	B	C	L	I	O
1	2		SEC		2	3		LDA	TEMPI+2	2	4		sbcyoy	ARMREG[RGN]
2	3		LDA	TEMPI+0	2	3		SBC	TEMPII+2					
2	3		SBC	TEMPI+0	2	3		STA	TEMPI+2					
2	3		STA	TEMPII+0	2	3		LDA	TEMPI+3					
2	3		LDA	TEMPI+1	2	3		SBC	TEMPII+3					
2	3		SBC	TEMPII+1	2	3		STA	TEMPI+3					
2	3		STA	TEMPI+2										

Table 6 Get Status Register as ARM Processor by 6502 ISA

B	C	L	I	O	B	C	L	I	O	B	C	L	I	O
1	3		PHP		1	2		ASL		1	4		PLA	
1	4		PLA		1	3		PHA		2	3		ORA	STATUSARM
2	3		STA	STATUS	2	3		LDA	0x40	2	3		STA	ATATUSARM
2	3		LDA	STATUS	1	2		LSR		1	4		PLA	
2	2		AND	0x80	1	3		PHA		2	3		ORA	STATUSARM
1	3		PHA		2	3		LDA	STATUS	2	3		STA	STATUSARM
2	3		LDA	STATUS	2	2		AND	0x01	1	4		PLA	
2	2		AND	0x20	1	2		ASL		2	3		ORA	STATUSARM
1	2		ASL		1	2		ASL		2	3		STA	STATUSARM
1	2		ASL		1	2		ASL						
1	2		ASL		1	2		ASL						
1	2		ASL		2	3		STA	STATUSARM					

Table 5 32-bit Rotation Comparison

6502 ISA				Self-Define ISA			
B	C	I	O	B	C	I	O
2	5	LSR	SRC+3	1	4	ror0y	
2	5	ROR	SRC+2				
2	5	ROR	SRC+1				
2	5	ROR	SRC+0				

Table 7 Static Step Comparison

	6502 ISA	Self-defined ISA	Reduction rate(%)
Fetch	64	66 (18)	–
Decode	915	323 (166)	46.6
Execution	2009	620 (80)	65.2
Total	2998	909 (254)	61.2

Due to the fact that the 6502 status processor is different from the ARM status processor, the flag position needs to be changed by the following 6502 instructions as shown in **Table 6**. But in our self-defined ISA, the "mvccwh" instruction is created to move the current conditional code to work register with 1 byte and 1cycle.

4. Evaluation

In this section, we conduct an evaluation of our self-defined ISA and the 6502 ISA processors in emulating a 32-bit ARM program. An 8-Queen problem program is used as a test bench. We compare the two processors in terms of the step numbers and the size of the emulator.

4.1 Static Steps Comparison

Table 7 shows the detail of static steps between 6502 and Self-defined emulators. The second and third columns represent the step numbers of the 6502 ISA and our self-defined ISA, respectively. The fourth column show the reduction rate. The 5-stages are divided to three parts: the fetching part (Fetch), the decoding part (Decode) and the executing access part (Execution). Fetch includes instructions in the fetching stage. Decode includes the instructions during the decode stage. Execution includes the instructions of execution stage, memory access stage and write back stage. In the third column, the numbers in the parentheses are the address numbers of table jump instructions.

As can be seen from Table 7, our self-defined ISA reduces the steps of emulating the 8-Queen program by 46.6% compared with the 6502 ISA emulator in Decode. Also, in Execution, our self-defined ISA reduces the step numbers by 65.2%. In total, our self-defined ISA reduces the steps in emulating the 8-Queen program by 61.2% than 6502 ISA.

In addition, the size of 6502 ISA emulator is 5.7Kbyte, while that of our self-defined ISA emulator is 2.5Kbyte, which means the reduction in the size of the emulator by 38.6%.

4.2 Discussion

Comparing the results of our self-defined ISA and 6502 ISA, we have demonstrated the effecitiveness of our self-defined emualtor against the 6502 emualtor. More specifically, our self-defined ISA that has two 4-byte index registers is useful for loading and storing 4-byte guest instructions, while the 6502 ISA that contains two 1-byte index registers is not able to load the guest instruction in 2 cycles. Especially, our self-defined ISA working register that gets data from index registers is only performed for the arithmetic, logic or shift calculations. The data transfer between working register and index register is done by 1-byte self-defined ISA. On the other hand, the accumulator of the 6502 ISA loads the data from zero-page address, which takes 2 bytes and 3 cycles.

In addition, our table jump instruction that has 2 bytes and 4 cycles helps the decode work efficiently. The 2-way table jump helps the decode work for checking every bit of guest instructions flexiblely. On the other hand, the 6502 ISA needs to decode the instructions steps by steps, which takes more execution cycles because the flag position of the status register in the 6502 ISA is different from the ARM status register (conditional code register). The 6502 ISA takes 34 steps to have the same position with the ARM status register, which takes a number of cycles on the

execution stage. In conslusion, the structure of our self-defined ISA processor is more effecitive than the 6502 ISA processor , for emulating ARM processors.

5. Conclusion

This paper evaluated our self-defined ISA processor against the 6502 ISA processor for emualting 32-bit ARM applications. Our proposed 8-bit processor consists of a 2.5KB emulator that has the same function as ARM ISA by our self-defined instruction set architecture (ISA). Through an evaluation using a test program in our experiment, our self-defined ISA processor achieved the reduction of emulation steps by 61.2% and that of the emualtor size by 38.6% compared with the 6502 ISA processor.

Our proposed ISA may still have room for improvement, for example, for reduction the size of the emualtor. This will be studied in our future work.

Acknowledgments This work is partly supported by ALCA, challenging Exploratory Research 24650020 and 2012 STARC joint research.

References

[1] Young-Je Cho, Ji-Hoon Shin, S.MBobade, Young-Bae Kim, Duck-Kyun Choi.: Evaluation of Y2O3 gate insulators for a-IGZO thin film transistors, *Thin Solid Film*, pp. 4115– 4118(2009).

[2] Yasuhiko Nakashima.: A Study Of Emulator Oriented Small CPU for Realizing Film Computers, *CPSY2012-13 SwoPP*, pp. 25–30, AUG. (2012).

[3] *ARM Architecture Reference Manual*, ARM limited, ARM DDI0100E.

[4] *MCS6500 Microcomputer Family Programming Manual*, Second Edition, MOS TECHNOLOGY. INC (1976).