An Embedded Processor with Instruction Packing

C. Satayavibul, P. Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Phayathai road, Bangkok 10330, Bangkok, THAILAND
g49cst@cp.eng.chula.ac.th , prabhas@chula.ac.th

***Abstract* - This work proposes a design of, SX4, a resource efficient 32-bit processor. SX4 is designed for a limited resource embedded system. The performance of SX4 has been improved by the "instruction packing" method. The measurement on the performance improvement is evaluated. Instruction packing reduces the code size by 33.0% at the same time it improve the speed of the processor by 17.1%. The proposed design has been realized on a FPGA device. SX4 processor requires 10,020 equivalent gates. Its maximum frequency is 63 MHz. In term of cycle consumed, it compares very well to commercial 32-bit Xilinx's microprocessor, Micro Blaze.**

I. INTRODUCTION

Nowadays, the embedded devices are playing important role in daily life. Computer exists in many things, such as mobile phone and automatic vending machine. As a result, the research in this area has many objectives: to improve the performance, to minimize code size, to minimize power consumption or to reduce cost of production depend on their functional purposes.

According to its working environment, the embedded devices have to be built to fit its surrounding conditions. A modern mobile phone needs a high performance processor for its multimedia applications, while most of the other devices don’t require such high performance.

This is the main motivation of the work, to develop a resource efficient processor with adequate performance. The main focus of this work is to apply instruction packing method to reduce the size of executable code and also to improve the performance of a processor. The performance improvement is possible due to the reduction in number of cycle in fetching instructions. Such method is done by putting more than one instruction into same memory address space, which reduces the code size. As the code segment and data segment of the proposed processor share the same memory bus, instruction packing eases the memory bottle neck problem by reducing instruction traffic between processor and memory. Moreover, the proposed method requires little amount of additional resource so that the enhanced processor is still very resource efficient. Pipelining is another option for improving the processor’s performance but it requires large modification on processor’s data path which can violate the resource efficient criteria.

II. PROCESSOR DESIGN

SX4 is a 32-bit processor with no pipelined execution. Its design is based on the design of stack-based processor, SMC [1]. Fig. 1 demonstrates system overview of the proposed processor. Additional perpherals are attached to the system via memory-mapped I/O. SX4's instruction set is also based on SMC's ISA. SMC's zero-address instructions, which operate with data on top of stack, are replaced by one-address instructions in SX4 so that there is no stack-based operation in SX4. The processor operates with memory as shown in Fig. 1. One part of memory is writable; another part is read only (ROM). The ROM stores the code segment.

A. *Data path*

This processor data path is improved from the previous stack-based processor, SMC. (see Fig. 2) The processor has 32-bit data width. It consists of five registers: PC (program counter), IR (instruction register), AC (Accumulator), FP (frame pointer) and BUFFER.

B. *Instruction Set Architecture*

SX4 uses three instruction formats: Short, Long1 and Long2.(see Fig. 3) The Short format instruction, which is 16-bit wide, consists of two parts: 8-bit opcode and 8-bit operand. The long instruction formats are 32-bit wide. Long1 format has 8-bit opcode and 24-bit operand. Long2 format has 4-bit opcode, 20-bit operand and another extra 8-bit operand.

SX4 instructions can be divided into four categories: arithmetic&logic, data manipulation, control and misc (see Table I). Unary instruction operates on AC. The binary instruction needs one more element, which depended on the addressing mode. It can be either an immediate value of the operand or a local variable stored in the activation record. LD and ST access global memory. GET and PUT access local variables. CALL creates a new activation record. RET restores the previous activation record and returns to the caller.

The processor's operation is similar to an accumulator machine. This design uses an activation record, instead of general purpose registers. Although the activation record is slower than register, but there is some benefit of using the activation record. Firstly, it reduces size of the processor. Moreover, the activation record eliminates register saving and restoring during a function call.

Normally, the SX4 processor takes one cycle for instruction fetch and one more cycle for executing the instruction. But for some complex instruction, execution needs two cycles to

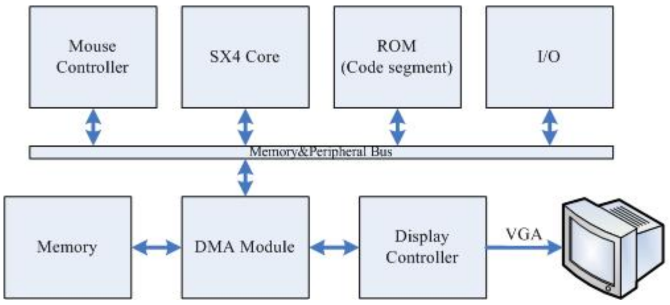


Figure 1. System overview of the proposed processor.

complete. As a result, overall performance of SX4 is around 2 to 3 cycles per instruction.

III. Instruction Packing

As mentioned earlier, the embedded devices are designed to fit their working environment. In this paper, we introduce an instruction packing method, which causes little impact to the resource of a processor. The resource increased from including the instruction packing is less than 10% of the original design while it reduces code size and cycle consumed in program execution.

Instruction packing is done by putting two short instructions in the same memory address space (see Fig. 4). As two instructions are stored in one word, the processor will have two instructions after one instruction fetch. There is a case in which two short instructions can not be packed together. The

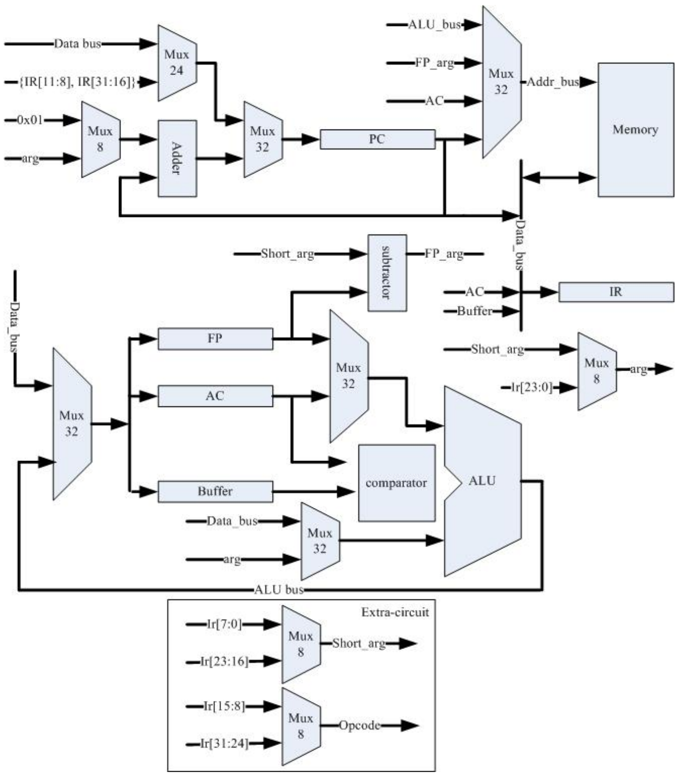


Figure 2. The data path of the proposed processor.

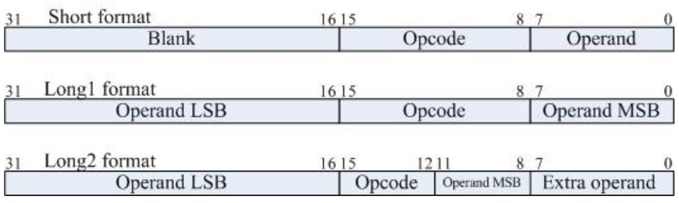
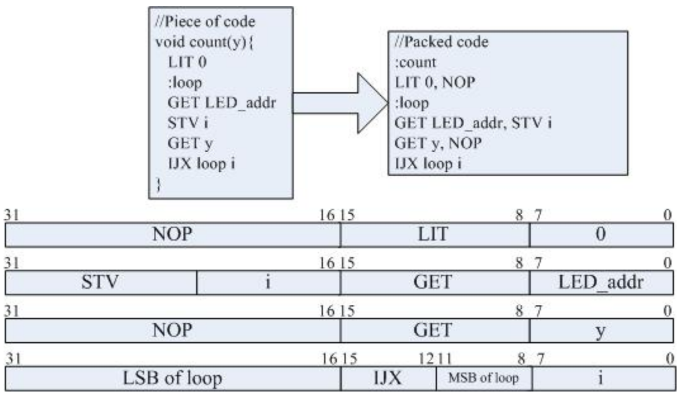

31 Short format 16 15 8 7 0
Blank
Opcode
Operand
31 Long1 format 16 15 8 7 0
Operand LSB
Opcode
Operand MSB
31 Long2 format 16 15 12 11 8 7 0
Operand LSB
Opcode
Operand MSB
Extra operand

Figure 3. The proposed ISA format.



```
//Piece of code
void count(y){
  LIT 0
  :loop
  GET LED_addr
  STV i
  GET y
  IJX loop i
}
```

```
//Packed code
:count
LIT 0, NOP
:loop
GET LED_addr, STV i
GET y, NOP
IJX loop i
```

Figure 4. Example of instruction packing.

instruction, which is the destination of CALL or JMP, cannot be placed in the most significant half word (31st – 16th bit). This is due to the proposed design's architecture does not suport half word directional in CALL and JMP instruction. Says, the program can not jump to the instruction placed in the most significant half word. As a result, such instructions can not be packed with the earlier instruction. Fig. 4 illustrates example of such case. The GET LED_addr instruction is the destination of the later IJX instruction, therefore it can not be packed with the earlier LIT 0 instruction. Hence, NOP instruction is packed with the LIT 0.

In the best case, instruction packing can reduce the program size by 50% and improve processor's throughput by 25%. This method also reduces program's code size. To include instruction packing on SX4, there is a small modification of the control unit. The data path needs two 8-bit MUX to select which half word is used. The extra circuit is shown in Fig. 2.

A complier plays a role in packing the code for the processor. Improvement in performance also depends on how well the code has been packed.

TABLE I
The List of The Proposed Processor's Instructions.

Instruction category	**Instruction**
Arithmetic&logic	ADD, SUB, MUL*, DIV*, BOR, BAND, BXOR, MOD*, ADDI, SUBI, MULI*, DIVI*, BORI, BANDI, BXORI, MODI*
Data manipulation	LD, ST, LDL, STL, LDX, STV, GET, PUT, INC, DEC
Control	JMP, JT, JF, JMPL, JTL, JFL, IJX, CALL, RET
Misc.	LIT, LITL, NOP

* indicates the instruction has not yet been implemented.

See the following function written in a machine code as follow:

```
add:
    get a  // 2 cycles consumed
    add b // 2 cycles consumed
    ret 3  // 3 cycles consumed
```

The code above can be optimized by packing GET and ADD instruction together as follow:

```
add:
    get a add b // 3 cycles consumed
    ret 3  // 3 cycles consumed
```

As a result, the packed code runs one cycle faster than the unpacked code, hence the performance improved by 14%.

IV. Measurement

In this section we test the processor with a suite of benchmarks. There are seven programs used in the measurement. There are seven test suites used in this paper: Bubble, Merge, Quick, Fibonacci, Hanoi, Sieve and AES.

Bubble: bubble sort 100 items of data. The initial data is ordered in descending order.

Merge: merge sort 100 items of data. The initial data is ordered in descending order.

Quick: quick sort the descending ordered data. The following number indicates number of data items.

Fibonacci: calculate the value of Fibonacci 10. The program is written in recursive function.

Hanoi: solves the 6 disks Hanoi problem.

Sieve: finds all prime numbers which less than 100.

AES: AES (Advance Encryption Standard) (128, 128) bit key block cipher [2].

To test the processor, we divide the measurement into three steps. Firstly, the instruction packed SX4 is compared to the original SX4 (see Table II, III). The test result shows that the SX4 executes packed code faster than unpacked code by 17.1% averaging over all benchmarks. Also packed program code size is 33.0% smaller than unpacked programs.

Secondly, code size comparison between packed SX4 program and other processors' (see Table IV, V). One of them is the stack-based processor [1]. The stack-based ISA is widely known to achieve a compact executable code [3].

TABLE II
THE COMPARISON BETWEEN
PACKED CODE SIZE AND UNPACKED CODE SIZE.

Test program	Unpacked code size (Byte)	Packed code size (Byte)	Size reduce
Bubble	96	64	33.3%
Merge	388	224	42.3%
Quick	200	168	16%
Fibonacci	100	68	32%
Hanoi	180	116	35.6%
Sieve	152	92	39.5%
AES	1,336	912	31.7%

The total equivalent gates needed in Micro Blaze system, the processor and 8kbytes of block ram is 315,528 gates. The Micro Blaze alone needs about 55,000 gates. The test program is written in C programming language and compiled by gnu C compiler. We use Xilinx On-Chip Peripheral Timer/Counter to measure the number of cycle consumed during execution. The numbers of cycle shown in Table V are the number of cycle that Micro Blaze takes to finish the test program and stop the timer. In term of cycle, the SX4 compares very well with Xilinx Micro Blaze. If running at the same frequency, the proposed processor will be about 15% slower than Micro Blaze, except for Fibonacci and Hanoi test suite. Because of the use of activation record, SX4's function call is very simple and fast. Therefore, SX4 defeats Micro Blaze on heavily recursive call programs. SX4 code size is about 60% of the Micro Blaze code size. In term of performance, Micro Blaze's maximum frequency is 91 MHz while the proposed design's is 63 MHz.

V. Related Work

Processor proposed in paper [1] uses stack-based instruction approach to achieve the resource efficient criteria. Its operations take elements on the top of stack so it does not require general purpose registers. Performace is the major weakness of the stack-based processor. Many methods, include instruction packing [4] , are applied to improve the performance. But the stack-based operations, which require a lot of memory accesses, have limitted the processor's performance.

There are many approaches in instruction packing. One of them is the use of Instruction Register File (IRF) [5, 6]. IRF

TABLE III
THE PERFORMANCE COMPARISON BETWEEN
PACKED CODE AND UNPACKED CODE.

Test program	Unpacked code Cycle consumed	Packed code Cycle consumed	Speedup
Bubble	327,706	282,953	13.7%
Merge	34,712	27,888	19.7%
Quick			**14.8%**
20 items	5,388	4,575	15.1%
60 items	32,948	28,116	14.7%
100 items	82,908	70,856	14.5%
Fibonacci	3,875	3,112	19.7%
Hanoi	4,398	3,708	15.7%
Sieve	2,341	1,859	20.6%
AES	59,710	50,640	15.2 %

TABLE IV
THE COMPARISON BETWEEN
PACKED CODE SIZE AND THE STACK-BASED PROCESSOR CODE SIZE.

Test program	Stack-based code size (Byte)	Packed code size (Byte)	Comparison
Bubble	124	64	51.6%
Quick	187	168	89.8%
Fibonacci	50	68	136%
Hanoi	153	116	75.8%
Sieve	137	92	67.2%
AES	650	912	140.3%

keeps the frequently occurring pair of instructions in the program. This approach indicates multiple instructions by indexing the IRF.

Similar to instruction packing, code compression aimed for reducing the code size. The well-known code compression examples are ARM Thumb [7] and MIPS16 [8]. The special instruction set is designed to offer small code size. Number functions are unavailable in code compression mode. As a result, the performance of the processor drop while operating in code compression mode.

TABLE V
THE COMPARISON BETWEEN
THE PROPOSED PROCESSOR AND XILINX MICRO BLAZE.

Category	Xilinx Micro Blaze	Proposed processor	Comparison
Circuit size (Equivalent gate)	315,528*	10,020	3.2%
	55,000**		18.2%
Code size (byte)			
Bubble	172	64	37.2%
Merge	396	224	56.6%
Quick	264	168	63.6%
Fibonacci	112	68	60.7%
Hanoi	168	116	69.0%
Sieve	132	92	69.7%
AES encryption	1,524	912	60.0%
Performance (cycle consumed)			
Bubble	248,354	282,953	114.0%
Merge	23,389	27,888	119.2%
Quick			
20 items	4,131	4,575	111.0%
60 items	24,439	28,116	115.1%
100 items	73,442***	70,856	96.5%
Fibonacci	4,810	3,112	64.7%
Hanoi	4,102	3,708	90.4%
Sieve	1,195	1,859	155.6%
AES	43,500	50,640	116.4%
Maximum frequency (MHz)	91	63	0.69

* 8kbyte of Block ram is included in the equivalent gate count.

** approximated equivalent gate count of Micro Blaze processor. Block ram is excluded.

*** indicates external memory is used to hold the Micro Blaze program's Stack/Heap section.

VI. CONCLUSION

The design presented here of a 32-bit processor with instruction packing achieves the main objective, to design a resource efficient adequate performance processor. We divide the conclusion into three points of view. First of all, in term of resource consumed, the proposed design has been realized on a Xilinx Spartan3 FPGA device. It consumes 10,020 equivalent gates. Secondly, in term of code size, the result is still satisfactory. SX4's code size is obviously smaller than the conventional 32-bit processor. The proposed processor code size also comparable to the stack-based processor [1], which is a 16-bit processor. Finally, the number of cycle of SX4 executing the benchmark suites are comparable to a commercial processor, Micro Blaze. However, the maximum frequency of the SX4 is significantly lower than Micro Blaze's. To be fair, these results are based on small set of benchmarks and should be taken as preliminary result. Our current work is expanding on this result to include larger and more diversed benchmark suite and better code generation quality.

VII. REFERENCES

[1] A. Burutarchanai, P. Nanthanavoot, C. Aporntewan, and P. Chongstitvatana, "A stack-based processor for resource efficient embedded systems," Proc. of IEEE TENCON 2004, Thailand, 21-24 November 2004.

[2] J. Daemen and V. Rijmen, "The Rijndael Block Cipher: AES proposal," 1999.

[3] P. Nanthanavoot and P. Chongstitvatana, "Code-Size Reduction for Embedded Systems using Bytecode Translation Unit," Conf. of Electrical/Electronics, Computer, Telecommunications, and Information Technology (ECTI), Thailand, 13-14 May 2004.

[4] P. Nanthanavoot, A. Burutarchanai and P. Chongstitvatana, "Instruction packing for a 32-bit resource efficient processor," National Science and Technology Development Agency (NSTDA) Annual Conference, Thailand, 27-30 March 2005 .

[5] S. Hines, G. Tyson and D. Whalley, "Reducing instruction fetch cost by packing instructions into register windows," Microarchitecture, 2005. MICRO-38. Proceedings. 38th Annual IEEE/ACM International Symposium on Microarchitecture MICRO 38, 12-16 Nov. 2005. Page(s):11 pp.

[6] S. Hines, J. Green, G. Tyson and D. Whalley, "Improving Program Efficiency by Packing Instructions into Registers," ACM SIGARCH Computer Architecture News , Proceedings of the 32nd Annual International Symposium on Computer Architecture ISCA '05, May 2005.

[7] Advanced RISC Machines Ltd., "An Introduction to Thumb," March 1995.

[8] K. D. Kissell, "MIPS16: High-density MIPS for the Embedded Market," In Proc. of Real Time Systems '97 (RTS97), 1997.