A stack-based processor for resource efficient embedded systems

Alongkot Burutarchanai, Phanupan Nanthanavoot, Chatchawit Aporntewan, Prabhas Chongstitvatana
Department of Computer Engineering Chulalongkorn University, Bangkok 10330, Thailand
g46abl@cp.eng.chula.ac.th, g45pnt@cp.eng.chula.ac.th u37cap@cp.eng.chula.ac.th, prabhas@chula.ac.th

Abstract

This work proposes the design of a 16-bit stack-based processor. It is aimed for a limited resource embedded system. The processor data path is simple. The instruction set is minimal. The proposed design has been realized on a FPGA device. The measurement on the required resource and performance based on the test suite aimed for the smart card applications indicates the suitability of the proposed design. The processor requires 6497 equivalent gates. Its maximum frequency is 45 MHz. In terms of code size, it is comparable to commercial 8-bit processors.

1. Introduction

In the last decade the progress of microprocessor design has been phenomenal. Performance rises according to Moore's law. The performance is double every 18 months. Performance is the key driving force for the progress of the last decade. However, the new applications have shifted the design landscape once again to low power, portable devices [1]. Embedded systems are emerging as a major driving force for computing devices. The much larger volume of demand makes embedded systems a dominant factor in industries. The constraints for embedded systems are power requirement. At the same time the amount of resource for a portable device is limited.

The aim of this work is to explore a design for a very limited resource constraint. The intended application for this processor is the smart card. The resource to implement a processor and the available memory are limited. The main consideration is to minimize the required resource. Performance issue is secondary. To reduce the requirement on memory, the machine code based on stack instruction is investigated.

Conventional machine code is not the most compact form to represent an executable code. The intermediate code for a virtual machine is usually much smaller because of its higher semantic content. One of the most popular form of intermediate code is based on stack addressing. A stack machine code is very compact due to its use of stack which does not required addressing bits. Majority of instructions thus do not required the operand in the instruction as it is implicit in the stack.

2. Processor design

2.1 Data path

The processor has 16-bit data width. It consists of four registers and a simple ALU. The registers are assigned special functions corresponded to the function of instruction set. The data path is purposefully constraint to be minimal and hence consumes very little resource.

2.2 Instruction Set

The instruction set is byte-coded. It consists of three types of instruction formats: zero, 8-bit and 16-bit operands. The opcode is 8 bits. There are 25 instructions divided into 4 groups: load/store from memory and local variables, arithmetic and logic, control flow. The arithmetic and logic group included usual arithmetic instructions, shift, and complete set of bitwise logic instructions. The control flow group contained a high level CALL and RET which create and destroy the activation record.

The four registers are: TOS (top of stack cache), SP (stack pointer), FP (frame pointer) and PC (program counter). The binary operators take two elements from top of stack. The unary operators operate on top of stack (TOS). LD and ST access to the global memory. GET and PUT access the local variable in the current activation record pointed by FP register. CALL has a high semantic content as it creates the new activation record and instantiates the actual parameters then passes the control to the caller. RET and RETV restore the previous activation record and return to the caller. RETV also pushes a return value to the top of stack of the caller. This instruction set tends toward minimalism.

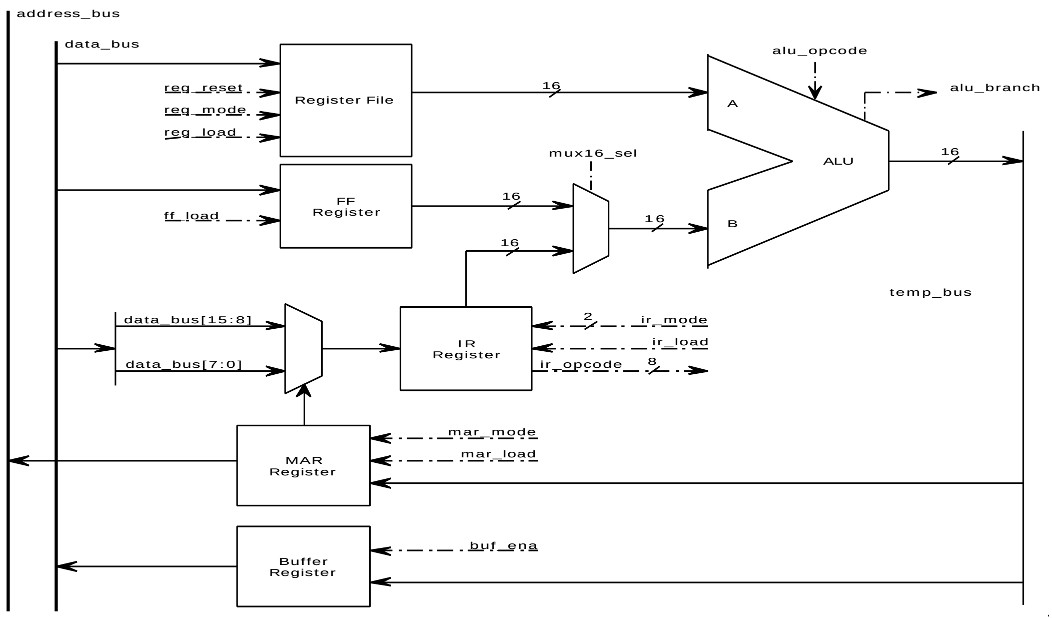

Figure 1 the data path to the stack-based processor

The choice of instruction set for a stack machine has a close relationship to a high level language. The "semantic gap" between its machine language and a high level language is narrow. One can almost write a stack machine language directly from a high level language source program. The higher semantic content, especially on the function call and parameter passing, helps to reduce the size of code.

See the following program:

```
//sum from a to b, s is a local
to sum a b | s =
  s = 0
  while a <= b
    s = s + a
    a = a + 1
  s
to main =
  print sum 1 10
```

It can be written in a machine code for the proposed stack machine as follows:

```
:sum
LIT 0, PUT s,
:loop
GET a, GET b, LE, JF exit,
GET s, GET a, ADD, PUT s,
GET a, LIT 1, ADD, PUT a, JMP loop
:exit
GET s, RETV

:main
LIT 1, LIT 10, CALL sum, SYS print
end
```

The instruction fetch operates on byte basis. The stack manipulation is on 16-bit data. The stack data structure is stored on the memory. The basic cycle of fetch and execute a simple instruction takes 18 clocks. The complex instruction takes a large number of clocks, such as CALL takes 59 clocks. (see Fig.2)

The design is realized on a FPGA. Using the Xilinx device and their synthesizer software, WebPack, the total equivalent gate count is 6497. The maximum frequency is 45 MHz. The control unit consumes more than half of the resource. This indicates the area for future optimization.

3. Measurements

To test the processor, it is used to run the test suite. The test suite is based on the intended application of this processor, the smart card. The test suite composed of AES (Advanced Encryption Standard) block cipher [2], OCB (offset code book encryption) [3], and fibonacci (heavily recursive call). The performance is measured on AES compared to the reported figures from [2] (see Table 1). The stack processor uses 317,346 clocks for (128, 128) bit key versus the 8-bit Intel 8051 processor 4,065 x 12 clocks and the 8-bit Motorola 68HC08 processor 8,390 x 1 clocks. In order to improve performance of the stack processor, two additional instructions are provided for array indexing. The result is 284,108 clocks for (128, 128) key, an improvement of 10% (see Table 1, stack2). Other tested program is the fibonacci program which uses CALL instruction heavily. The stack processor uses 54,089 clocks to calculate fib(7) (code size 52 bytes). In term of the code size, the stack processor has the advantage. The executable code for the stack processor is smaller than other processors in AES cipher (see Table 1). In the OCB application, the code size is 7,537 bytes versus the executable machine code size of 22,048 bytes on a PC.

In terms of resource, the proposed design is compared with two 16-bit general purpose processors from the opencores community: C16 and T80 [4]. All designs are implemented on Xilinx Spartan2 devices, XC2S100 (100,000 gates) in order to compare the required resource in the same metric. C16 cannot be synthesised on the chosen device due to lacking of RAM block, in other words, C16 requires more resource than available on the FPGA chip. T80 uses 15,009 equivalent gates. Its maximum frequency is 33 MHz. The proposed design fared very well as it uses less than half the resource of T80. (See Table 2)

Table 1 Performance Comparison of the stack processor to Intel 8051 and Motorola 68HC08 for (128, 128) bits key AES cipher

processor	Clocks	code size (bytes)
stack	317,346	677
stack2	284,108	650
8051	4,065 x 12	768
68HC08	8,390 x 1	919

Table 2 Resource comparison of each CPU

Processor	Frequency	Equivalence gate
Stack	45 MHz	6,497
C16	40 MHz	N/A
T80	33 MHz	15,009

4. Related work

The most well-known stack virtual machine is Java Virtual Machine (JVM) [5] as it is embedded into most browser. One commercial processor design that has its ISA based on stack-based instructions is PicoJava chip [6]. [7] Designed a hardware translation unit to run Java bytecode on a register-based machine with good performance. This is another alternative to using a stack-based machine like our proposal. There are volume of work on customized instruction set for specific applications for example [8, 9] including using programmable gate array for realizing these instruction set embedded in an ordinary processor [10]. The flexibility of customizing instruction set for specific applications has been commercialized by a number of companies for example Xtensa [11]. For multimedia work load the most well-known is MMX instruction from Intel [12]. These are aimed for high performance. The design proposed here has a different goals, it is aimed to be resource efficient.

5. Discussion

The design presented here of a stack-based processor achieved the intended objective, the efficient use of resource, adequately. Implementing on a Xilinx FPGA device, it consumes 6,497 equivalent gates with the maximum frequency of 45 MHz. In terms of code size, it is comparable to the class of commercial 8-bit processors. The advantages of a stack-based processor are its simplicity and its small code size has been demonstrated. In terms of performance, it is still lagged behind commercial register-based processors. However, the performance of the proposed processor can be improved in several ways. A number of special instructions can be added to its instruction set. Observing the profile of the frequency of each instruction execution, the additional addressing mode and multi-bit shift will improve the performance. Implementing load index and shift 2, 4 bits altogether results in running the AES benchmark with 277,157 clocks (12.6% faster). Performing code generation optimization will improve this figure by another 10-20%. Other special instruction can be considered. The prime candidate for AES applications is the multiplication in GF2 field which is used heavily. It can be implemented in hardware using a simple left shift and conditional bitwise exclusive-or. It can also be implemented as a sequence of control using the existing data path. The control unit is not optimized in this implementation. Its speed can be improved and its size can be reduced. Other advantage of customized instruction set is that many optimizations which are not feasible with a commercial processor can be performed. For example, to go for an extremely compact code size, a special instruction encoding can be used which can contain two instructions in one byte [13].

6. References

[1] C. Kozyrakis and D. Patterson, "A new direction for computer architecture research", IEEE computer, Nov. 1998, pp.24-32.

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

[3] P. Rogaway, M. Bellare, J. Black, And T. Krovetz, "OCB: A Block-Cipher Mode Of Operation For Efficient Authenticated Encryption", Proc. 8th CCS, pp. 196-205, ACM, 2001.

[4] www.opencores.org

[5] B. Joy (Ed), G. Steele, J. Gosling, G. Bracha, Java (TM) Language Specification (2nd Ed), Addison Wesley Pub., 2000.

[6] H. McGhan and M. O'Conner, 'PicoJava : a direct execution engine for Java bytecode", IEEE Computer, Vol.31 No. 10, 1998.

[7] R. Radhakrishnan, R. Bhargava and L. John, Improving Java Performance Using Hardware Trasnslation, In Proceedings of 15th ACM international Conference on Supercomputing, pages 427-439, 2001.

[8] R. Leupers and J. Elste and B. Landwehr, "Generation of interpretive and compiled instruction set simulators", Proc. of the Asia and South Pacific Design Automation Conference, Jan. 1999.

[9] S. Pees, A. Hoffmann, V. Zivojnovic and H. Meyr, "LISA - Machine Description Language for Cycle-Accurate Models of Programmable DSP Architectures", Design Automation Conference, 1999, pp. 933-938.

[10] T. Glokler and S. Bitterlich, "Power Efficient Semi-Automatic Instruction Encoding For Application Specific Instruction Set Processors", ICASSP, 1999.

[11] R. Gonzalez, "Xtensa: a configurable and extensible processor", IEEE Micro, March/April 2000, p.60

[12] MMX technology, http://developer.intel.com

[13] V. Kotrajaras, P. Chongstitvatana, "Nibbling Java byte code for resource-critical devices", National Conf. of Computer Science and Engineering, Thailand, 2003.

.

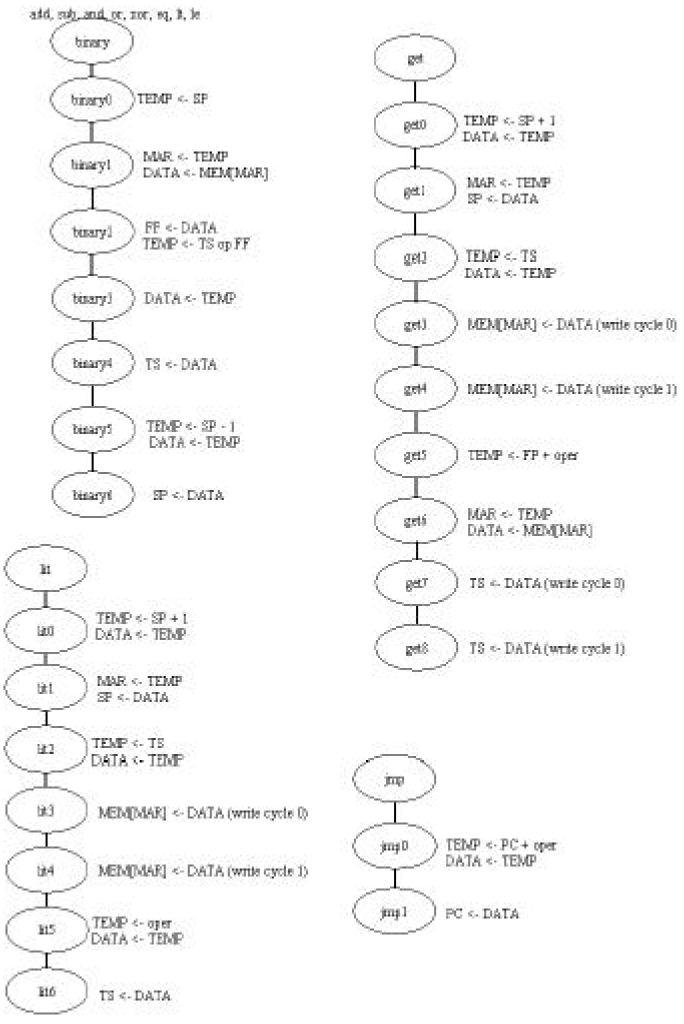

Fig. 2 the control unit states for some instructions