Journal of Computer Science 10 (3): 411-422, 2014
ISSN: 1549-3636
© 2014 Science Publications
doi:10.3844/jcssp.2014.411.422 Published Online 10 (3) 2014 (http://www.thescipub.com/jcs.toc)

A CASE FOR HYBRID INSTRUCTION ENCODING FOR REDUCING CODE SIZE IN EMBEDDED SYSTEM-ON-CHIPS BASED ON RISC PROCESSOR CORES

[1]Govindarajalu Bakthavatsalam and [2]K.M. Mehata

[1]Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, Irungattukottai, India
[2]School of Information and Computer Sciences, B S Abdur Rahman University, Chennai, India

Received 2013-10-13; Revised 2013-11-12; Accepted 2013-11-23

ABSTRACT

Embedded computing differs from general purpose computing in several aspects. In most embedded systems, size, cost and power consumption are more important than performance. In embedded System-on-Chips (SoC), memory is a scarce resource and it poses constraints on chip space, cost and power consumption. Whereas fixed instruction length feature of RISC architecture simplifies instruction decoding and pipeline implementation, its undesirable side effect is code size increase caused by large number of unused bits. Code size reduction minimizes memory size, chip space and power consumption all of which are significant for low power portable embedded systems. Though code size reduction has drawn the attention of architects and developers, the solutions currently used are more of cure than of prevention. Considering the huge number of embedded applications, there is a need for a dedicated processor optimized for low power and portable embedded systems. In the study, we propose a variation of Hybrid Instruction Encoding (HIE) for the embedded processors. Our scheme uses fixed number of multiple instruction lengths with provision for hybrid sizes for the offset and the immediate fields thereby reducing the number of unused bits. We simulated the HIE for the MIPS32 processors and measured code sizes of various embedded applications of MiBench and MediaBench benchmarks using an offline tool developed newly. We noticed up to 27% code reduction for large and medium sized embedded applications respectively. This results in reduction of on-chip memory capacity up to 1 mega bytes that is very significant for SoC based embedded applications. Considering the large market share of embedded systems, it is worth investing in a new architecture and development of dedicated HIE-RISC processor cores for portable embedded systems based on SoCs.

Keywords: Chip Space, Code Size, Instruction Encoding, Instruction Set Architecture, SoC

1. INTRODUCTION

An embedded system is not a general purpose computer. Instead, it is a preprogrammed system to perform one or more dedicated functions. In most embedded systems, size, cost and power consumption are critical than performance (Hennessy and Patterson, 2012). A large number of embedded systems such as cellular phones, cameras, toys are portable and battery operated and their design is based on System-on-a-Chip (SoC). As applications become increasingly complex, code memory consumes a large portion of the area in SoC architectures. Apart from increased chip space and cost, the power consumption also increases due to larger code memories. Hence minimizing code size is an essential requirement in Battery Operated Portable Embedded Systems (BOPES). In the study, we deal with reduction of code size at processor Instruction Set Architecture (ISA) level so that the code generated by the compiler is shorter.

Corresponding Author: Govindarajalu Bakthavatsalam, Department of Computer Science and Engineering, Sri Venkateswara College of Engineering, Irungattukottai, India

The RISC processors such as ARM and MIPS are widely used in the embedded SoCs, due to high performance offered by the RISC Architecture. The Fixed Instruction Encoding (FIE) of RISC processors helps in simpler instruction decoding and easy pipeline design (Hennessy and Patterson, 2012). But the FIE increases the code size as some fields are either unused or underutilized in several instructions. In embedded SoCs, the code memory is integrated with the processor and the other system hardware on a single chip (Fisher *et al*., 2005). This limits the available space for the application memory for the SoC architectures. Although embedded systems typically cost far less than desktop computers, several billion embedded SoCs are sold annually compared to a few hundred million desktop processors (Vahid and Givargis, 2006). Our paper proposes replacing the ‘uniform instruction size’ feature by ‘hybrid instruction size’ in the embedded RISC cores used in BOPES so as to reduce the code memory space, for embedded programs.

The main contributions of this work can be summarized as follows. The study proposes replacement of FIE with Hybrid Instruction Encoding (HIE) with two modifications to RISC Architecture: Multiple instruction sizes and hybrid lengths for the offset and immediate fields. We designed a HIE-ISA for the MIPS processor as a modification to MIPS32 ISA for evaluating the HIE-ISA and developed an offline tool, that converts the object codes from MIPS ISA to HIE-ISA. This tool measures the code size savings for embedded applications in MiBench and MediaBench benchmark suites.

1.1. RISC Instructions and Code Density

RISC processors generally have three types of instructions: ALU, Load or store and Branch and Jump (Hennessy and Patterson, 2012). **Figure 1** summarizes the basic formats of MIPS32 integer instructions (other than floating-point instructions) with examples. All the instructions are 32-bits and the most significant 6 bits contains the opcode. In the I-type and J-type instructions, the opcode itself indicates the exact operation. In the R-type instructions, the op field identifies the instruction type and the fn field (least significant bits 0-5) indicates the exact operation. The R-type is for register-to-register operations. The I-type is for data transfers, branches and immediate operations. In load/store type instructions, the offset field is added to the contents of the *rs* register, usually an address, to form the effective address for one of the operands, either the source or the destination. The major drawbacks of RISC instruction formats causing increased code size are listed below using MIPS32 as example.

Five bits are unused in most R-type instructions as illustrated in **Fig. 2** for the and instruction.

In most immediate type instructions, 8 bits are sufficient for the operand and the remaining 8 bits are redundant. **Figure 3-6** illustrate the four different cases of immediate field patterns out of which only in one case, both bytes of immediate field are non-zero.

In branch instructions such as beq, the offset field is underutilized in those cases where the offset required can be specified in 8 bits.

1.2. Related Work

There have been significant efforts at system design level to compensate for the code size increase caused by the FIE., Several techniques (Heikkinen *et al*., 2009) have been implemented to minimize the object code. These are classified into three types (Xie *et al*., 2006), Offline Code compression, Compiler techniques and ISA modification. The first two techniques retain the original ISA but require software/hardware additions by the system developers, whereas the third technique involves supporting a new instruction set that is a subset of the original ISA.

In ISA modification cases, such as ARM Thumb and MIPS16, the original ISA is modified with shorter instructions, limited instruction set, smaller operand fields and fewer GPRs. This results in code size reduction by 30 to 40%, but reduces performance by 15 to 20% (Bonny and Henkel, 2008) and also requires a decoder and de-compressor inside the processor to support both ISAs. The other drawback of this approach (Benini *et al*., 2004) is the performance penalty caused by lack of several instructions in the dense instruction set. This approach customizes the existing RISC instruction set architecture with narrow instructions supporting fewer operations, smaller operand fields and fewer registers. A variation of this approach is used by microMIPS (ITGPLC, 2009; 2010) that is a recent addition to MIPS architecture. It offers a new ISA that supports both 16-bit and 32-bit instructions in a single program. However, its new instructions have restrictions on using certain registers. Some of the 16-bit microMIPS instructions can access only 8 of 32 GPRs. RISC-V project at University of Berkeley is somewhat similar to microMips architecture permitting 32-bit base instructions and 16-bit extensions of compressed instructions. It hopes to achieve up to 30% savings in static and dynamic memory space. Though the researchers term it as variable instruction decoding, it offers a two instruction length feature similar to microMIPS.

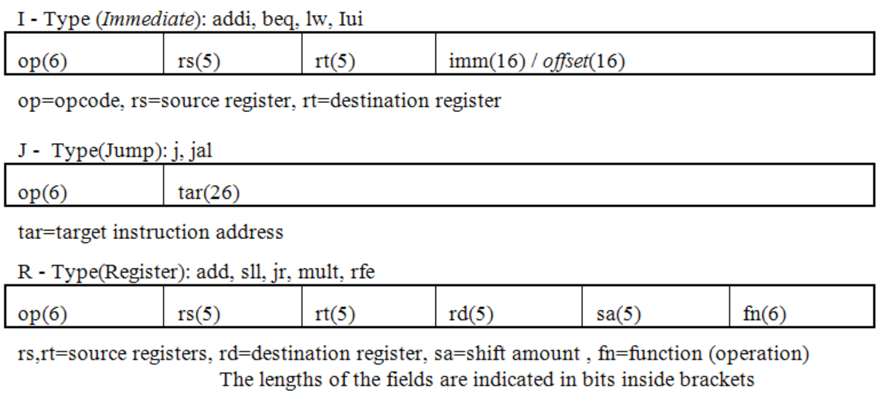

I - Type (*Immediate*): addi, beq, lw, Iui
op(6)
rs(5)
rt(5)
imm(16) / *offset*(16)
op=opcode, rs=source register, rt=destination register
J - Type(Jump): j, jal
op(6)
tar(26)
tar=target instruction address
R - Type(Register): add, sll, jr, mult, rfe
op(6)
rs(5)
rt(5)
rd(5)
sa(5)
fn(6)
rs,rt=source registers, rd=destination register, sa=shift amount , fn=function (operation)
The lengths of the fields are indicated in bits inside brackets

Fig. 1. MIPS32 basic instruction formats

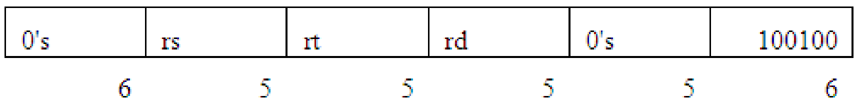


Fig. 2. Format of and instruction in MIPS32 ISA

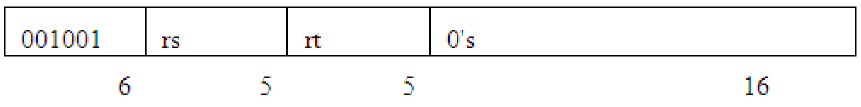


Fig. 3. Format of addiu instruction with immediate field containing zero value

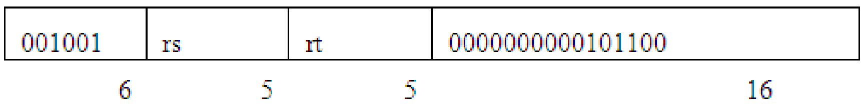


Fig. 4. Format of addiu instruction with only most significant byte of immediate field containing zero value

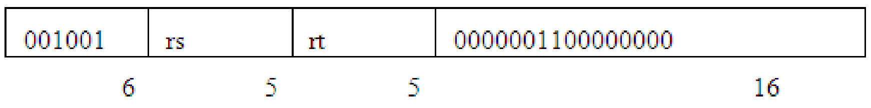


Fig. 5. Format of *addiu* instruction with only least significant byte of immediate field containing zero value

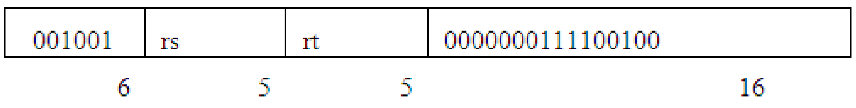


Fig. 6. Format of addiu instruction with both bytes of immediate field containing non-zero value

A mixed approach is also followed (Bonny and Henkel, 2008) by re-encoding unused bits in the instruction format for a specific application, using Huffman Coding algorithm. The compressed code and the decoding table are stored in the code memory. During execution of the program, a hardware decoder external to the processor decodes the compressed instructions.

The study presents an architectural solution that is application independent and recommends fixing the length of various instructions to 1, 2, 3 or 4 bytes instead of uniform size of 4 bytes. Though compiler and processor modifications are required to existing RISC architectures, these are one time efforts by the processor manufacturers/compiler developers and there is no burden on embedded system developers as required in other approaches. Also, it is a program independent solution for embedded applications. However, this strategy does not prevent inclusion of other methods for achieving additional amount of code size reduction for specific applications.

The organization of the study is as follows. Section 2 discusses the behavior of RISC processors for embedded applications and describes the HIE-ISA proposed by us for BOPES as a modification to existing MIPS32 ISA. Section 3 describes the architecture of the offline tool developed by us for static simulation of HIE-ISA and details the experiments carried out with this tool using MiBench and MediaBench applications for comparing the object code sizes for MIPS32 ISA and the proposed HIE-ISA. Section 4 discusses the results. Section 5 presents conclusions.

2. MATERIALS AND METHODS

2.1. Behavior of Embedded Applications on RISC Processors

In order to estimate the extent of wastage in RISC object codes, we analyzed the MIPS32 object codes (Patterson and Hennessy, 2008) for the embedded benchmark suits, MiBench and MediaBench. The MiBench (Guthaus *et al*., 2001) is a set of benchmark programs in C, for six embedded applications: Automotive and Industrial control, Consumer Devices, Office Automation, Networking, Security and Telecommunications. **Table 1** lists the MiBench programs used by us for evaluating the HIE for MIPS32. Typical applications of Automotive and Industrial Control are air bag controllers, engine performance monitors and sensor systems. These benchmarks perform mathematical, calculations, bit counting, sorting and image recognition. The typical examples of consumer devices are scanners, digital cameras and Personal Digital Assistants (PDAs).

The benchmarks mainly consist of multimedia applications with the representative algorithms for jpeg encoding/decoding, image color format conversion, image dithering, color palette reduction, MP3 encode/decoding and HTML typesetting. The typical examples of network devices are switches and routers. The work done by the embedded processors in these devices involves shortest path calculations, tree and table backups and data input/output.

The algorithms used in these benchmarks are finding a shortest path in a graph and creating and searching a Patricia *trie* data structure. There are some benchmarks common to network, security and telecommunication classes. The Telecommunications benchmarks have algorithms for voice encoding/decoding, frequency analysis and checksum calculation. The Office applications are primarily text manipulation algorithms. The typical examples of office automation are printers, fax machines and word processors. The security benchmarks have algorithms for data encryption, decryption and hashing.

The MediaBench suite (Lee *et al*., 1997) is composed of multimedia applications. MediaBench 1.0 contains 19 applications collected from image processing, communications and DSP applications. Certain applications such as jpeg and gsm are common to MiBench and MediaBench suites. A short note on the selected applications in MediaBench suite is given in **Table 2**.

We cross-compiled the MiBench and MediaBench programs on Intel PC and analyzed the compiler output using our tool MIDACC, an offline code analyzer tool. Given a MIPS object code, this tool produces the instruction count for each instruction type. It also analyzes the utilization of the offset and immediate fields in the instructions and lists extent of wastage in terms of percentage of total program size. Analysis of MIPS object codes using this tool reveals two interesting behaviors.

Four instructions, *addu*, *addiu*, *lw* and *sw*, dominate the embedded programs consuming as high as 60% of the code, as shown in **Fig. 7**. Applying 80-20 rule, any technique to improve the density of these four instructions will reduce the code size.

The extent of wastage due to underutilization of the offset and immediate fields varies from 10 to 20% of the code size (**Table 3**) for the embedded applications.

The largest program of Automotive applications of MiBench suite is the *susan* occupying 51,000 bytes of memory. It is an image recognition package used for a vision based quality assurance application. **Figure 8** shows that in *susan*, the immediate/offset field is fully used in 80% of the cases only. This amounts to wastage of 10,200 bytes, i.e., 20% of the code memory. Our proposed HIE-ISA for MIPS processor gives overall code reduction of *susan* and *mpeg2* by 27 and 21% respectively due to hybrid instruction length feature and hybrid length provision for the offset and immediate fields. Though the HIE-ISA does not eliminate the wastage totally, it minimizes the wastage to a major extent. The extent of code size reduction achieved with HIE-ISA is also indicated in **Table 3**.

Table 1. MiBench benchmarks used for evaluation of FIE

Auto/Industrial applications	
Program	Functions
basicmath	Simple mathematical calculations such as cubic function solving, integer square root and angle conversions from degrees to radians
bitcount	Tests the bit manipulation abilities of a processor by counting the number of bits in an array of integers
qsort	Sorts a large array of strings into ascending order using the quick sort algorithms
susan	An image recognition package for recognizing corners and edges and typically used for a vision based quality assurance application
Consumer applications	
Program	Functions
jpeg	An algorithm for image compression and decompression; used to view images embedded in documents
lame	An MP3 encoder that supports constant, average and variable bit-rate encoding
typeset	A general typesetting tool with a front-end processor for HTML; representative of a core component of a web browser that might be used in a consumer device
Office applications	
Program	Functions
stringsearch	Searches for given words using a case insensitive comparison algorithm
ispell	A spelling checker supporting contextual spell checking, correction suggestions and non English languages
rsynth	A text to speech synthesis program that integrates several pieces of public domain code into a single program
Network applications	
Program	Functions
dijkstra	Constructs a large graph in an adjacency matrix representation and calculates the shortest path between every pair of nodes using dijksra's algorithm
patricia	Creates and searches a patricia trie structure
CRC32	Same as CRC32 in telecom
sha	Same as sha in security
blowfish	Same as blowfish in security
Security applications	
Program	Functions
blowfish	A symmetric block cipher with a variable length key.
sha	A secure hash algorithm that produces a 160-bit message digest for a given input; used in the secure exchange of cryptographic keys and for generating digital signatures
rjindael	A block cipher with the option of 128-, 192- and 256-bt keys and blocks.
Telecommunications applications	
Program	Functions
CRC32	Perform a 32-bit Cyclic Redundancy Check (CRC) on a file. Useful to detect errors in data transmission
FFT	Performs a fast fourier transform and its inverse transform on an array of data; useful in digital signal processing to find the frequencies contained in a given input signal
ADPCM	Adaptive differential pulse code modulation; takes 16-bit linear PCM samples and converts them into 4-bit samples, yielding a compression rate of 4:1
GSM	Global standard for mobile communications. A standard for voice encoding/decoding data streams

Table 2. MediaBench benchmarks used for evaluation of FIE

Program	Functions
jpeg	A standardized compression method for full colour and gray-scale images. JPEG is lossless, meaning that the output image is not exactly identical to the input image. Two applications are derived from the JPEG source code; cjpeg does image compression and djpeg does decompression
MPEG	A dominant standard for high quality digital video transmission. The important computing kernel is a discrete cosine transform for coding and the inverse transform for decoding. The two applications used are mpeg2enc and mpeg2dec for encoding and decoding respectively
GSM	European GSM 06.10 provisional standard for full rate speech transcending, pry-ETS 300 036, which uses residual pulse excitation/long term prediction coding at 13 Kbit/s. GSM 06.10 compresses frames of 160 13-bit samples (8 kHz sampling rate, i.e., a frame rate of 50 Hz) into 260 bits.
G.721	Voice compression**:**Reference implementations of the CCITT (International Telegraph and Telephone Consultative Committee) G.711, G.721 and G.723 voice compressions
PEGWIT	A program for public key encryption and authentication
EPIC	An experimental image compression utility. The compression algorithms are based on a bi-orthogonal critically sampled dyadic wavelet decomposition and a combined run-length/Huffman entropy coder. The filters have been designed to allow extremely fast decoding without floating-point hardware
ADPCM	Adaptive differential pulse code modulation is one of the simplest and oldest forms of audio coding

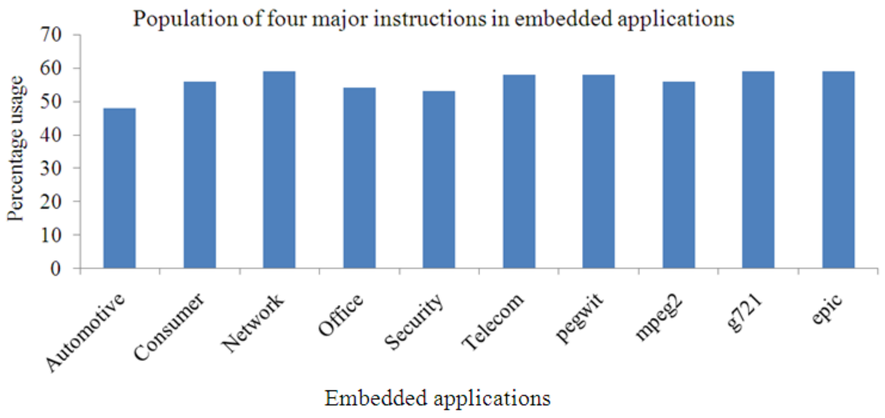
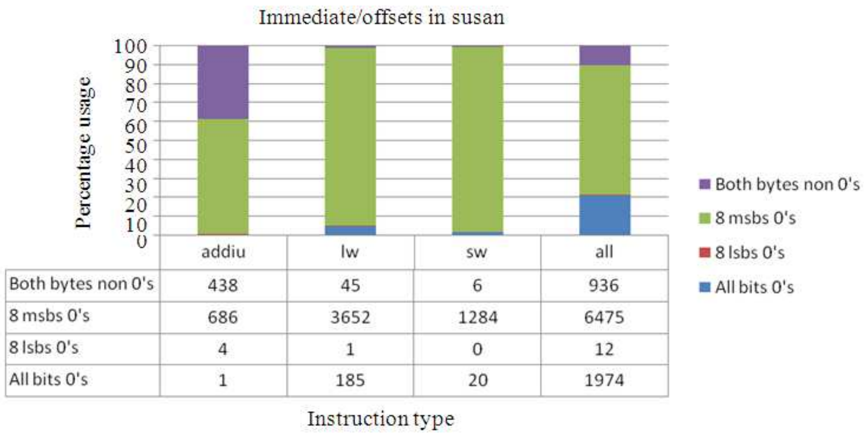


Fig. 7. Distribution of four frequent instructions in MiBench and MediaBench benchmarks



	addiu	lw	sw	all
Both bytes non 0's	438	45	6	936
8 msbs 0's	686	3652	1284	6475
8 lsbs 0's	4	1	0	12
All bits 0's	1	185	20	1974

Fig. 8. Usage of immediate/offset fields in susan

Table 3. Impact of four major instructions and offset/immediate fields in Embedded object codes for MIPS32

MiBench/MediaBench Program	Percentage usage of four major instructions	Percentage underutilization of offset/immediate fields	Percentage reduction of code size in HIE
basicmath	33	10	21
bitcnts	58	16	27
qsort	57	13	24
susan	65	21	27
jpeg	63	20	26
typeset	62	19	21
lame	45	16	18
dijkstra	59	16	22
patricia	59	16	22
rijndael	59	16	22
blowfish	59	16	22
sha	42	18	23
adpcm	59	16	22
CRC32	59	16	22
FFT	56	16	21
gsm	58	16	21
ispell	53	14	19
rsynth	47	13	22
stringsearch	59	16	22
pegwit	58	16	22
mpeg2	56	15	21
G721	59	16	22
epic	59	16	21

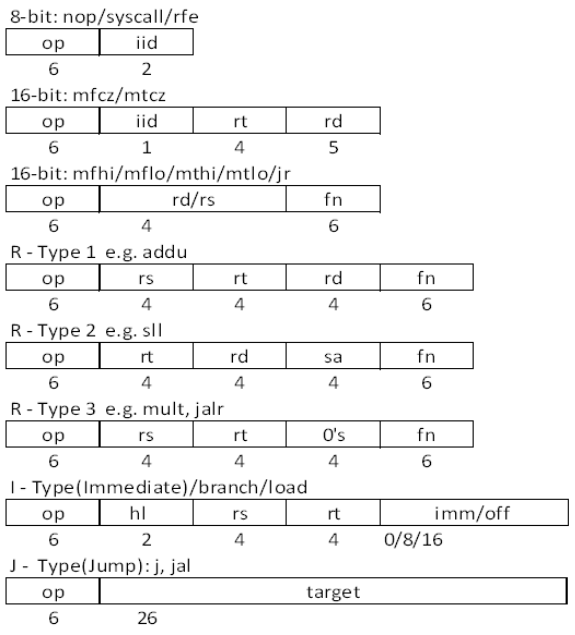


Fig. 9. HIE-RISC instruction formats

2.2. HIE-Methodology For MIPS32

Our goal for the HIE-ISA is minimizing unused fields within instructions and improving the utilization of the offset and immediate fields. Based on our analysis of all the 66 integer instructions of MIPS32 ISA, we finalized on 8 different types of instructions for the HIE-MIPS ISA.

2.2.1. HIE-MIPS Instructions

For the HIE-ISA, we decided on four sizes for the integer instructions: Three 8-bit, seven 16-bit, twenty one 24-bit, three 32-bit and thirty two with three options: 16/24/32 bits. **Figure 9** shows the proposed instruction formats for HIE-MIPS. To evaluate the effectiveness of our proposed HIE-ISA, we modeled it for the MIPS32 ISA. Basically, for every integer instruction of MIPS32 ISA, we provide an equivalent HIE-ISA instruction. Out of 66 integer instructions, j, jal and break, are retained as 32 bits as in MIPS32 ISA. The remaining instructions are translated into one of the 8 types. In several ALU instructions, there are five zeroes. If three more bits are made free, these instructions can be reduced to 24 bits. Hence we reduced the register fields by 1 bit each. This restricts the number of GPRs to 16; however, it will not strain the compiler as graph coloring technique for register allocation works satisfactorily for 16 GPRs, (Hennessy and Patterson, 2012). Popular RISC Processors such as

ARM and SH4 have only 16 registers. In addition to reducing the length of register fields, the shift amount (sa) field (used in the shift instructions) is reduced by 1-bit. The nop, rfe and syscall are 8-bit instructions with a common opcode and a 2-bit iid field to identify the instruction. The 16-bit instructions are jr, mfhi, mflo, mthi, mtlo, mfcz and mtcz. In mfcz and mtcz, the rd field is retained as 5 bits since it refers to coprocessor registers. The iid bit differentiates between mfcz and mtcz. The mthi, mflo, mthi and mtlo have a common format and the register field is shared between rd and rs. In mfhi/mflo/mthi/mtlo format, the rd/rs field denotes rd for mfhi and mflo. For mthi and mtlo, it denotes rs.

The 24-bit instructions that form three different R-types, are add, addu and div, divu, mult, multu, nor, or, sll, sllv, sra, srav, srl, srlv, sub, subu, xor, slt, sltu and jalr. In type1, there is no sa field. In type2, there is no rs field. In type3, there are four zeroes to maintain byte alignment. The remaining 32 instructions have three length options: 16, 24, or 32 bits. The offset and immediate fields are encoded in a unique way in our proposal. If the value of the offset/immediate is zero, these fields are omitted. When one of the bytes in the offset/immediate is zero, that byte is omitted and the hybrid length identifier hl is formed. **Table 4** shows a typical example using hexadecimal notation. All the four cases have a common opcode.

2.2.2. Mapping MIPS32 ISA to HIE-MIPS

MIPS Instructions are converted into new HIE instructions of 8 different types and the conversion depends on the opcode and immediate/offset fields. **Table 5** indicates the length of each converted instruction. All unconverted instructions are retained as 32 bits.

3. RESULTS

3.1. HIE-Simulator Tool-MIDACC

We developed a standalone software tool for simulating the HIE for MIPS32 and measuring the code size reduction. Since we need to simulate a new ISA, it will be a complex process if we were to use any existing simulator for the HIE-MIPS. Our objective is not executing any program, but measuring static code sizes of HIE-MIPS, for various embedded applications and comparing with static code sizes of MIPS32 for the same applications. Hence we decided to develop a simple offline tool that can convert the object codes of MIPS32 into object codes of HIE-MIPS. We built the tool, MIPS Instruction Distribution Analyser And Code Converter (MIDACC), in C#, with twin functions: Code analysis and code compression. The first module performs analysis of given MIPS32 object code and identifies the distribution of 66 integer instructions in the object code. This module also provides details on utilization of the immediate and offset fields by different instructions in the application programs (**Table 3 and Fig. 8**). The second module is a code converter that converts each instruction in the object code, from MIPS32 ISA to HIE-MIPS ISA, as per the HIE-MIPS methodology. The integer instructions of MIPS32 are converted into nine groups in HIE-ISA (**Table 5**). The software tool was developed under Windows XP on Intel PC and occupies 2 MB memory and runs in. NET Framework 3.5.

3.2. Estimating WASTIO Percentage

WASTIO refers to wastage due to unused (underutilized) bits in immediate and offset fields in the MIPS32 code. The wastages are classified into four types A, B, C and D based on the number of immediate/offset bytes that are redundant in the code; type A: 2 bytes wastage; type B: 1 byte wastage of,least significant byte; type C: 1 byte wastage of,most significant byte; and type D: no wastage. WASTIO percentage is calculated using the formula below:

$$\text{WASTIO percentage} = 100 \times (\text{WASTIO/code size})$$

We observed varying extent of reduction for embedded programs as shown in **Fig. 10**. Since certain applications contain multiple benchmarks, the figures use geometric means for the reduction percentages.

Table 4. Encoding Offset/Immediate field in HIE-MIPS

MIPS32 encoding	HIE-MIPS encoding	hl bits	Instruction size in HIE (bits)
0000	-	00	16
000F	0F	01	24
0F00	0F	10	24
0F0F	0F0F	11	32

Table 5. MIPS32 ISA Vs HIE-ISA mapping

HIE group	No. of instructions	HIE size (bits)	Instructions	MIPS32 type	Remarks
A	3	8	rfe, syscall, nop	Exception and interrupt	A common OP field with two-bit iid field to differentiate
B	2	16	mfcz, mtcz	Data movement with coprocessor	A common OP field plus one-bit iid. The rt denotes the CPU register and the rd (5 bits) the coprocessor register
C	5	16	jr, mfhi, mflo, mthi, mtlo	jr is jump register instruction; others are data movement type	The OP and fn fields are similar to MIPS32. The 4-bit register field is rs for jr, mthi and mtlo. For mfhi and mflo, the register field is rd.
D	13	24	add, addu and, nor, or, sllv, srav, srlv,sub, subu, xor,slt, sltu	R- type; slt and slu are comparison instructions; others are arithmetic and logical instructions	HIE R-type1. All fields are similar to MIPS32 except that unused zeroes are deleted and the register fields are 4 bits in HIE.
E	3	24	sll, sra, srl	R- type; shift instructions	HIE R-type2. All fields are similar to MIPS32 except that the unused rs field is deleted and the register fields are 4 bits in HIE.
F	5	24	jalr, div, divu, mult, multu	R-type; arithmetic instructions	HIE R-type3. All fields are similar to MIPS32 except that 6 unused zeroes are deleted and the register fields are 4 bits in HIE; 4 zeroes maintain byte alignment. In jalr, the register fields are rs and rd; in other instructions, these are rs and rt.
G	32	3 options; 16/24/32	addi, addiu andi, ori, xori, lui, slti, sltiu, bczt,bczf,beq, bgez, bgezal, bgtz,blez, bltzal, bltz, bne, lb,lbu,lh, lhu, lw, lwcz, lwl, lwr,sb,sh, sw, swcz, swl, swr	I-type /branch/load/store. A mixture of arithmetic/logical, constant manipulation, compare, branch, load and store type instructions. Most are of I-type with 16-bit immediate field.The branch / load/store instructions have 16-bit offset field.	HIE I- type. All fields are similar to MIPS32 except that the immediate/ offset *field* can take three different lengths; 0/8/16 bits as indicated by the hl field.In lui, the rs field contains 4 zeroes. All register fields are 4-bits.
H	2	32	j,jal	jump type.	Exactly similar to MIPS.
I	1	32	break	break is interrupt and exception type.	Exactly similar to MIPS.

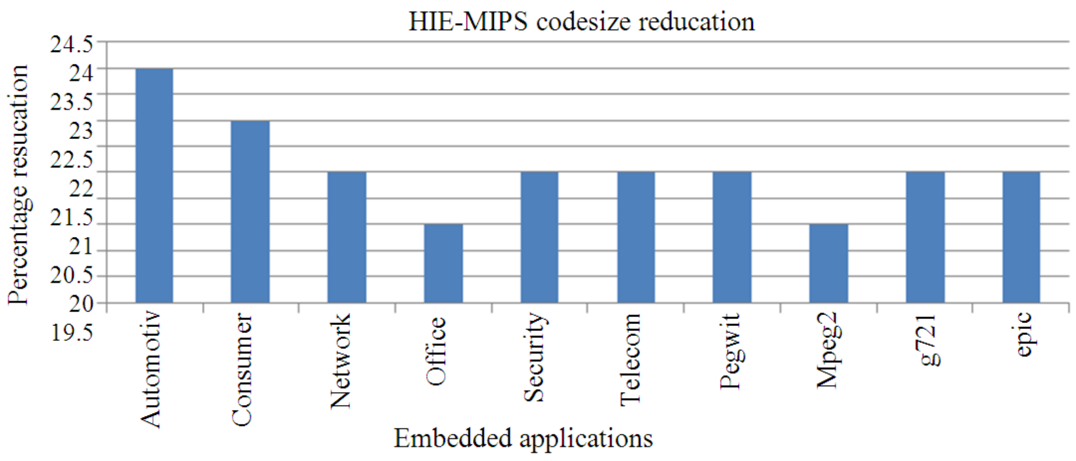


Fig. 10. Extent of code size reduction with HIE for mibench and mediabench

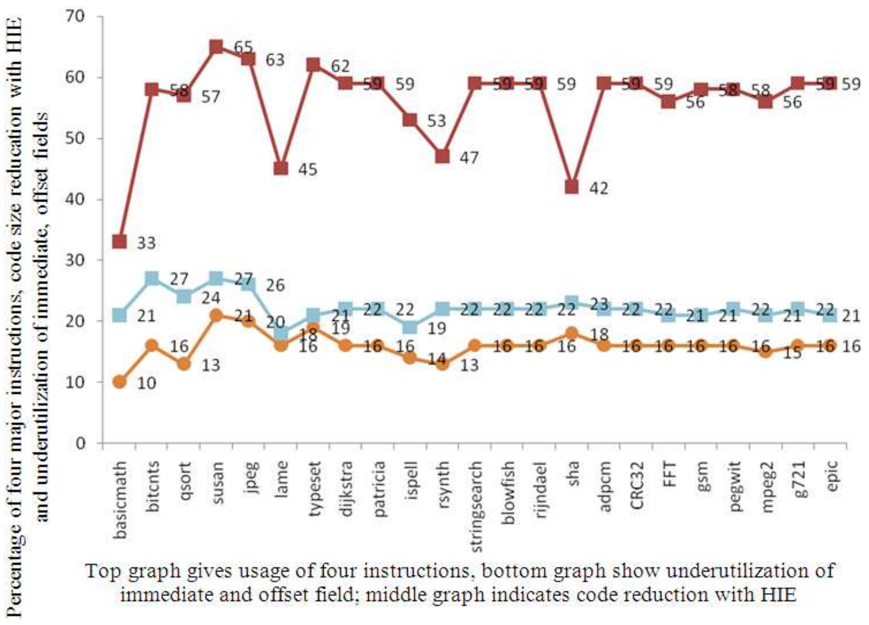


Fig. 11. Impact of instruction mix and immediate/offset fields on code size

Table 6. Typical code size reduction of embedded applications for HIE-MIPS

% Reduction	Small (≤8KB)	Medium (8KB-32KB)	Large (≥32KB)
Below 20	-	-	Lame, ispell
20-25	basicmath, qsort, sha	rsynth	typeset, fft, CRC 32, dijkstra, patricia, blowfish, rijndael, adpcm, gsm, stringsearch , pegwit, rnpeg2, g721, epic
Above 25	bitcount	-	jpeg, susan

4. DISCUSSION

It is observed that the Automotive and Consumer applications gain maximum with HIE-ISA; the *mpeg2* and the office applications gain least. The other applications get medium reduction. The Automotive and Industrial Control benchmarks show reduction varying from 21 to 27%. The image recognition program, *susan*, gets best reduction and the *basicmath* program gets the least reduction. Though 65% of *susan* code consists of the four major instructions, the poor result for the Automotive group is due to *basicmath* in which the four major instructions form only 33% of the code. The consumer benchmarks, get reasonably good reduction. The *jpeg* gets maximum reduction whereas the *lame* gets the least. The network benchmarks, *dijkstra* and *patricia*, get equal amount of reduction. In both benchmarks, 59% of the code is made up of the four common instructions. All the Telecommunications benchmarks undergo almost equal extent of reduction. In the office automation benchmarks, the *rsynth* and *stringsearch* programs get the maximum reduction and the *ispell* the least. The security benchmarks get medium reduction. The MediaBench programs also get medium reduction.

There is a wide variation in the sizes of the benchmark programs. Out of the 23 embedded applications, four are small (≤ 8KB), one is medium (8KB-32KB) and eighteen are large (≥ 32KB). **Table 6** summarizes the extent of code size reduction by HIE for the 23 benchmark programs classified according to their sizes. It is obvious that HIE offers satisfactory extent of code reduction for majority of the embedded applications. A relationship is found between the code size reduction in HIE-MIPS and two properties of MIPS32 object codes: One is quantum of four major instructions and the other is percentage underutilization of immediate and offset fields. This is visible from **Fig. 11**.

It is noticed from **Fig. 11** that code size reduction is higher for those programs that have higher amount of four major instructions and higher amount of under utilization of immediate and offset fields. This behavior forms the backbone of our HIE methodology. However, there are marginal exceptional behaviors by some programs. For instance, the *sha* has only 42% of four major instructions and only 18% of the code is wasted due to under utilization of immediate and offset fields. However, there are marginal exceptional behaviors by some programs. For instance, the *sha* has only 42% of four major instructions and only 18% of the code is wasted due to under utilization of immediate and offset fields. In spite of this, there is 23% code size reduction with HIE for the *sha*. This could be due to increased number of R-type instructions in the MIPS32 code for the *sha*. These instructions have been reduced to 24 bits in the HIE-MIPS code.

The instruction fetch and decode logics need to manage hybrid instruction lengths and multiple sizes of offset and immediate fields. These hardware changes do not need much additional space in the processor. However, reduced number of registers in HIE-RISC saves chip space. The processor itself occupies lesser area than the on-chip memory in embedded SoCs and hence the HIE reduces the overall chip area for SoCs. The study has estimated the static code size reduction for HIE based ISA and dynamic simulation is to be done for evaluating performance and power consumption. Marginal performance reduction can be tolerated for BOPES in view of savings in chip space and power consumption.

5. CONCLUSION

In The study, we have proposed a modified Hybrid Instruction Encoding in place of Fixed Instruction Encoding so as to reduce the code memory size in SoCs. We have established that four major instructions dominate the embedded applications occupying up to 65% of the code and up to 20% of the code size is wasted due to underutilization of the offset and immediate fields. This is in addition to wastage due to unused bits in other fields of the instructions.

An HIE-ISA has been proposed for RISC processors supporting multiple instruction sizes and four options for immediate and offset fields. We simulated HIE with four instruction sizes for MIPS32 processor and the results show code size reduction up to 27%. We experimented with twenty three benchmark programs collected from MiBench and MediaBench suites, using an offline static simulator developed by us. We noticed that except for two programs all others got reduced by more than 20%. Whereas two large programs got reduced by more than 25%, only two large programs got reduced by less than 20% in HIE code and the remaining 14 large programs got reduced between 20-25%. Considering the significant extent of savings in code memory and chip space in SoCs, we recommend development of dedicated HIE-RISC processor cores for the embedded market.

6. REFERENCES

Benini, L., F. Menichelli and M. Olivieri, 2004. A class of code compression schemes for reducing power consumption in embedded microprocessor systems. IEEE Trans. Comput., 53: 467-482. DOI: 10.1109/TC.2004.1268405

Bonny, T. and J. Henkel, 2008. Instruction re-encoding facilitating dense embedded code. Proceedings of the Conference on Design, Automation and Test, Mar. 10-14, IEEE Xplore Press, Munich, pp: 770-775. DOI: 10.1109/DATE.2008.4484772

Fisher, J.A., P. Faraboschi and C. Young, 2005. Embedded Computing: A VLIW Approach to Architecture, Compilers and Tools. 1st Edn., Elsevier, ISBN-10: 1558607668, pp: 671.

Guthaus, M.R., J.S. Ringenberg, D. Ernst, T.M. Austin and T. Mudge *et al*., 2001. MiBench: A free, commercially representative embedded benchmark suite. Proceedings of the 4th Annual Workshop on Workload Characterization, Dec. 2-2, IEEE Xplore Press, pp: 3-14. DOI: 10.1109/WWC.2001.990739

Heikkinen, J., J. Takala and H. Corporaal, 2009. Dictionary-based program compression on customizable processor architectures. Microproc. Microsyst., 33: 139-153. DOI: 10.1016/j.micpro.2008.10.001

Hennessy, J.L. and D.A. Patterson, 2012. Computer Architecture: A Quantitative Approach. 1st Edn., Elsevier, San Francisco CA., ISBN-10: 012383872X, pp: 493.

ITGPLC, 2009. microMIPS instruction set architecture-32-bit performance. Minimum System Cost.

ITGPLC, 2010. Beyond the Hype: MIPS-the Processor for MCUs.

Lee, C., M. Potkonjak and W.H. Mangione-Smith, 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. Proceedings of the 13th Annual IEEE/ACM International Symposium on Microarchitecture, Dec. 1-3, IEEE Xplore Press, Research Triangle Park, NC, pp: 330-335. DOI: 10.1109/MICRO.1997.645830

Patterson, D.A. and J.A. Hennessy, 2008. Computer Organization and Design. 4th Edn., Morgan Kaufmann, Amsterdam, ISBN-10: 0080922813, pp: 912.

Vahid, F. and T. Givargis, 2006. Embedded System Design: A Unified Hardware/Software Introduction. 1st Edn., Wiley India Pvt. Limited, New Delhi, ISBN-10: 812650837X, pp: 348.

Xie, Y., W. Wolf and H. Lekatsas, 2006. Code compression for embedded VLIW processors using variable-to-fixed coding. Proceedings of the 15th International Symposium on System Synthesis, Oct. 2-4, IEEE Xplore Press, USA., pp: 525-536. DOI: 10.1109/TVLSI.2006.876105