Instruction Encoding Techniques for Area Minimization of Instruction ROM

T. Okuma, H. Tomiyama, A. Inoue, E. Fajar, and H. Yasuura
Department of Computer Science and Communication Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University
6–1 Kasuga-koen, Kasuga, 816-8580 Japan

Abstract

In this paper, we propose instruction encoding techniques for embedded system design, which encode immediate fields of instructions to reduce the size of an instruction memory. Although our proposed techniques require an additional decoder for the encoded immediate values, experimental results demonstrate the effectiveness of our techniques to reduce the chip area.

1. Introduction

A breakthrough in CAD and semiconductor process technologies facilitates designing the systems which are constructed as a combination of memories, dedicated circuits and microprocessors. In such systems, a chip area has a strong influence on a system cost, and memory size becomes dominant in the chip area. The minimization of the memory area is effective for that of the chip area and the system cost.

Memory is used two purposes, data memory and instruction memory. To reduce the area of memory, these are two approaches; reduction of the number of words and reduction of the width of words. In instruction memory, the number of words depends on efforts of programmer and compiler, and the width of words depends on system architectures and instruction coding techniques. We attempt to reduce the memory area for instruction coding.

In an instruction word, an immediate field usually determines the width (or length) of instruction words. However, there is no application that all values which can be represented by the immediate field are used. In this paper, we propose instruction encoding techniques, which encode immediate fields of instructions to reduce the size of the instruction memory.

The rest of the paper is organized as follows. In section 2, we discuss related work. In section 3, we propose immediate encoding techniques. In section 4, we discuss hardware implementation. Experimental results are presented in section 5, and we conclude this paper in section 6.

2. Related Work

In [4], Liao et al. proposed a machine instruction row encoding technique in embedded application program. The encoding technique is as follows. A common instruction row which appears in the program is extracted, and it is registered as a function. After that, the registered instruction row is replaced with the pointer to the function (function call instruction). Then, the instruction row is compressed by making a common part a function. It is reported that the area of ROM is reduced by at most 30%. This technique reduce the number of instructions stored in ROM, i.e. the height of ROM. While our proposed techniques reduce the instruction word length, i.e. the width of ROM. And, it is possible to combine with Liao's technique.

In [8], Yoshida et al. proposed an instruction encoding technique, which encodes each instruction word in such a way that the instruction ROM size is minimized. The main purpose of this technique is low power, but it is effective to reduce the system area too. However, this technique requires a large overhead in hardware to decode instructions.

In [3], Ishiura et al. proposed another instruction encoding technique. An original instruction word is partitioned into several fields so that the decoder for each field is kept within a reasonable size. After that, the each field is encoded.

3. Instruction Encoding Techniques

3.1. Basic Idea

In many processors, the longest field of instruction is immediate value which represents memory addresses of operand or operand values. The bit length of address is determined by the memory space used in the program and that of operand value is determined by the data size. Then, the longer field is required for the larger system. But there is a room to compress the field of immediate values, because not all addresses and/or values may be appeared in a program.

In embedded systems, application programs are stored in ROM, which will not be rewritten after the system is

shipped out. The set of immediate values which appear in the program is fixed in the stage of the system design. Thus these immediate values can be encoded and the length of the field of immediate values can be reduced dramatically. In CPU, the encoded values are decoded by a decoder and instruction format of CPU need not be changed. By this method, we can reduce the instruction length on ROM.

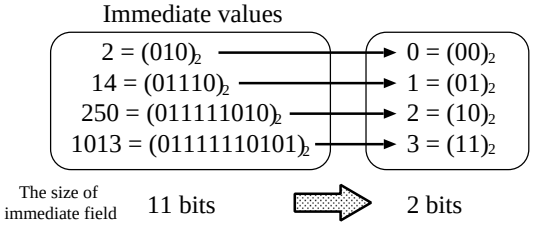

Figure 1. An example of the encoding

Figure 1 shows an example of the encoding techniques. In this example, there are four immediate values. The maximum value of the bit width in the immediate values determines the size of an immediate field, which is 11 bits here. When this values are encoded into codes from 0 to 3, the size of the immediate field is 2 bits. Therefore, the instruction word length can be reduced by at most 9 bits.

The idea of such encoding is not entirely new. It is actually a variation of the microcode compaction. See references[1, 5] for further particulars.

It is necessary to insert a decoder between the instruction memory and the instruction register, named an **immediate decoder**, which decodes the encoded immediate values to the original format of the CPU's instructions . The coding and the immediate decoder should be designed for each application program. But it is easy to be automated to this redesign phase.

3.2. Models of System

3.2.1 Instruction Format

We assume seven kinds of instruction format types, which is a simple model of popular RISC architecture (see Figure 2). In this figure, the field written "$\mathbf{O}_{\mathrm{p}}$" represents an opcode field. L_{op} is the size of this field, and it is shown as

$$L_{op} = \lceil \log |\mathbf{O}_{\mathrm{p}}| \rceil$$

where $\mathbf{O}_{\mathrm{p}}$ specifies the set of operations of the instructions which are used in an application. The field written "$\mathbf{R}_{\mathrm{eg}}$" is a register field. L_{reg} is the size of this field, and it is shown as

$$L_{reg} = \lceil \log |\mathbf{R}_{\mathrm{eg}}| \rceil$$

where $\mathbf{R}_{\mathrm{eg}}$ specifies the set of the registers which are used in an application. The fields written "$\mathbf{I}_2$", "$\mathbf{I}_4$", and , "$\mathbf{I}_7$" are immediate fields. L_{imm_i} is the each size of each field, and it is shown as

$$L_{imm_i} = \max_{\alpha \in \mathbf{I}_i}(\mathrm{len}(\alpha))$$

where $\mathbf{I}_i$ specifies the set of immediate values in the format type i which are used in an application program. $\mathrm{len}(x)$ means the bit width[1] of an immediate value x, which are either encoded or the original values. It is also called "effective bit size". All values in the immediate field should be expanded into the original values by the decoder, if it is encoded, before sending to data path. The length of each instruction format type L_i is shown as follows.

$$\begin{aligned} L_1 &= L_{op} + 3 \cdot L_{reg} \\ L_2 &= L_{op} + 2 \cdot L_{reg} + L_{imm_2} \\ L_3 &= L_{op} + 2 \cdot L_{reg} \\ L_4 &= L_{op} + L_{reg} + L_{imm_4} \\ L_5 &= L_{op} + L_{reg} \\ L_6 &= L_{op} \\ L_7 &= L_{op} + L_{imm_7} \end{aligned}$$

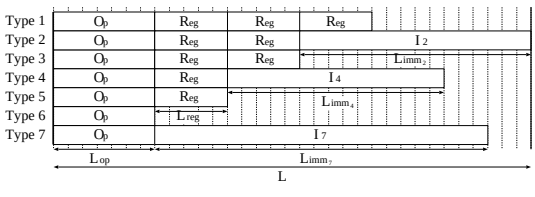

Figure 2. Instruction format

3.2.2 Instruction Word Length

The length of instruction L is fixed, that is the longest one of all types.

$$L = \max_i(L_i) \quad (1)$$

L is called the instruction word length.

The instruction word length determines the width of the instruction memory. Some instruction format types have the fields where the instruction memory is unused (see Figure 3). These fields are called an unused fields, and L_{uu_i} specifies the size of the unused field in the format type i. The longest instruction format type has no unused field, namely the length of unused field is 0. L_{uu_i} is shown as follows.

$$L_{uu_i} = L - L_i$$

[1]For example, $\mathrm{len}(0011011)$ is 6 , and $\mathrm{len}(0000110)$ is 4.

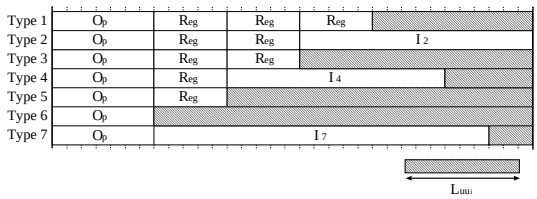

Figure 3. An unused field in the instruction format

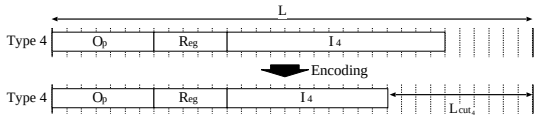

Figure 4. An example of $L_{cut_i}(i = 4)$

3.3. Encoding Techniques

In this paper, we propose six kinds of encoding techniques to reduce the instruction word length. We can define the reduction ratio of the ROM area including the immediate decoder for these techniques to indicate their effects. The reduction ratio R_{cost} is shown as

$$R_{cost} = \frac{N_{all} \cdot L_{red} - \delta}{N_{all} \cdot L}$$

where N_{all} is the number of all instructions in an application program, i.e. the hight of the instruction memory, and L is the instruction word length before encoding, i.e. the width of the instruction memory. δ is the cost of immediate decoder. L_{red} represents the instruction word length reduced by encoding, and it is shown as

$$L_{red} = \min_i(L_{cut_i}) \quad (2)$$

where L_{cut_i} specifies the difference between L (see Expression (1)) and an effective size of the format type i after encoding. Figure 4 shows an example of L_{cut_i}.

While the values of N_{all} and L do not change for all techniques, the values of L_{red} and δ are different. Since the cost of immediate decoder δ depends on the implementation, this chapter does not discuss about δ. An implementation of an immediate decoder is examined in the next chapter.

L_{red} is calculated by expression (2), and we only have to calculate L_{cut_i} for each technique to obtain the reduction ratio. In the rest of this chapter, each technique is precisely explained, and the equation of L_{cut_i} is derived.

a) Longest format/All Coding (LAC)

LAC is a method to encode only the immediate values of the longest instruction format type. Figure 5 shows LAC. In this figure, the shaded part is the immediate field eliminated by encoding.

L_{cut_i} of LAC is shown as

$$L_{cut_i} = \begin{cases} L_{imm_j} - \lceil \log |\mathbf{I}_j| \rceil & \cdots (i = j) \\ L_{uu_i} & \cdots (otherwise) \end{cases}$$

where j is the longest format type.

b) Individual format/All Coding (IAC)

IAC is a method to encode the immediate values of each field of type 2, type 4, and type 7 individually. Figure 6 shows IAC.

L_{cut_i} in IAC is shown as follows.

$$L_{cut_i} = \begin{cases} L_{imm_i} - \lceil \log |\mathbf{I}_i| \rceil + L_{uu_i} & \cdots (i = 2, 4, 7) \\ L_{uu_i} & \cdots (otherwise) \end{cases}$$

c) Union format/All Coding (UAC)

UAC is a method to encode the immediate values of each field of type 2, type 4, and type 7 altogether. Figure 7 shows UAC.

L_{cut_i} in UAC is shown as follows.

$$L_{cut_i} = \begin{cases} (L_{imm_i} - \lceil \log | \bigcup_{i=2,4,7} \mathbf{I}_i | \rceil) + L_{uu_i} & \\ & \cdots (i = 2, 4, 7) \\ L_{uu_i} & \cdots (otherwise) \end{cases}$$

d) Longest format/Partial Coding (LPC)

LPC is a method to encode only the immediate values of the longest instruction format type. However, it doesn't encode the values which can be represented in an immediate field after encoding. e.g. if 0 and 100 are the immediate value, 0 need not be encoded, 100 will be encoded as 1. The purpose of this method is to reduce the size of the immediate decoder. In this case, a flag bit to indicate whether the values is encoded or not is necessary. Figure 8 shows LPC. In this figure, the hatched part is the flag bit.

First of all, two function $\mathbf{Over}(\mathbf{I}, k)$ and $L_{over}(\mathbf{I})$ are defined before L_{cut_i} of this method is presented.

$$\mathbf{Over}(\mathbf{I}, k) = \{\alpha \in \mathbf{I} \mid \text{len}(\alpha) > k\} \quad (3)$$

$$L_{over}(\mathbf{I}) = \min\{\beta \mid |\mathbf{Over}(\mathbf{I}, \beta)| \leq 2^\beta\} \quad (4)$$

$\mathbf{Over}(\mathbf{I}, k)$ specifies the subset of $\mathbf{I}$ and has elements whose effective bit size is larger than k, where $\mathbf{I}$ is the set of immediate values. If β is defined as the size of immediate field, we only have to think about the following conditional expression.

$$|\mathbf{Over}(\mathbf{I}, \beta)| \leq 2^\beta$$

$L_{over}(\mathbf{I})$ specifies the minimum β which satisfying this requirement for $\mathbf{I}$. The immediate field can be shortened only to $L_{over}(\mathbf{I})$, even if it is minimum.

L_{cut_i} in LPC is shown as follows.

$$L_{cut_i} = \begin{cases} L_{imm_j} - (1 + L_{over}(\mathbf{I}_j)) & \cdots (i = j) \\ L_{uu_i} & \cdots (otherwise) \end{cases}$$

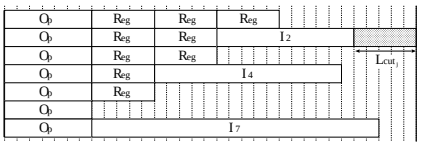

Figure 5. LAC

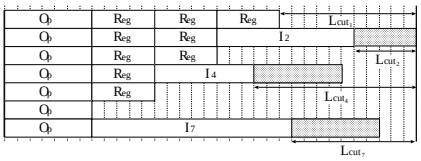

Figure 6. IAC

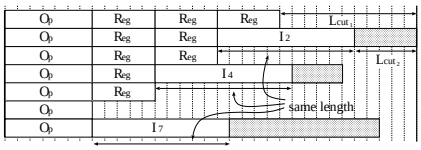

Figure 7. UAC

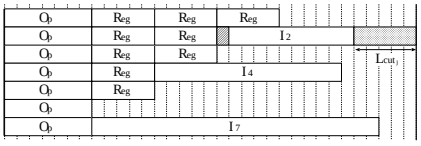

Figure 8. LPC

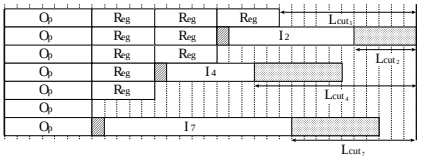

Figure 9. IPC

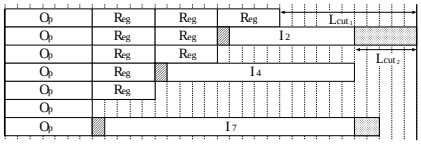

Figure 10. UPC

e) Individual format/Partial Coding (IPC)

IPC is a method to encode the immediate values of each field of type 2, type 4, and type 7 individually. Similar to LPC, it doesn't encode the values which can be represented in an immediate field after encoding, and the flag bit is necessary. Figure 9 shows IPC.

L_{cut_i} in IPC is shown as follows.

$$L_{cut_i} = \begin{cases} L_{imm_i} - (1 + L_{over}(\mathbf{I}_i)) + L_{uu_i} & \cdots (i = 2, 4, 7) \\ L_{uu_i} & \cdots (otherwise) \end{cases}$$

f) Union format/Partial Coding (UPC)

UPC is a method to encode the immediate values of each field of type 2, type 4, and type 7 altogether. Similar to LPC, it doesn't encode the values which can be represented in an immediate field after encoding, and the flag bit is necessary. Figure 10 shows UPC.

Because the number of encoded immediate values changes depending on the size of the fields $\mathbf{I}_2$, $\mathbf{I}_4$, and $\mathbf{I}_7$, we think about size by arranging a right edge of an immediate field as shown in Figure 10. Field $\mathbf{I}_4$ is L_{reg} bits longer than field $\mathbf{I}_2$, and Field $\mathbf{I}_7$ is $2 \cdot L_{reg}$ bits longer than field $\mathbf{I}_2$. The set of immediate values which can not be represented in these three immediate fields is defined as $\mathbf{Over}'(\beta)$, where β is the size of the field $\mathbf{I}_2$. $\mathbf{Over}'(k)$ can be calculated as follows.

$$\begin{aligned} \mathbf{Over}'(\beta) \quad = \quad & \mathbf{Over}(\mathbf{I}_2, \beta) \\ & \cup \ \mathbf{Over}(\mathbf{I}_4, \beta + L_{reg}) \\ & \cup \ \mathbf{Over}(\mathbf{I}_7, \beta + 2 \cdot L_{reg}) \end{aligned}$$

When $\mathbf{Over}'(\beta)$ is calculated, we only have to think about β such that the encoded values are represented in the field $\mathbf{I}_2$. Because the values which can be represented in this field is sure to be able to be represented in the field $\mathbf{I}_4$ and $\mathbf{I}_7$. Therefore, β which satisfies the following condition is appropriate as the size of field $\mathbf{I}_2$ after encoding.

$$|\mathbf{Over}'(\beta)| \leq 2^{\beta}$$

L_{over} specifies the minimum β which satisfying this requirement.

$$L_{over} \quad = \quad \min\{\beta \mid |\mathbf{Over}'(\beta)| \leq 2^{\beta}\}$$

L_{cut_i} in the UPC method is shown as follows.

$$L_{cut_i} = \begin{cases} (L_{imm_2} - (1 + L_{over}) + L_{uu_2} & \cdots (i = 2, 4, 7) \\ L_{uu_i} & \cdots (otherwise) \end{cases}$$

4. Discussion on Hardware Implementation

4.1. Decoder Cost

To decode immediate values in the above techniques, immediate decoders which decode the encoded values are required. There are two methods to implement the decoders, ROM decoders and logic circuits. In this paper, we consider the cost of the ROM decoders.

The cost of immediate decoder δ is shown as

$$\delta \quad = \quad |\mathbf{I}_{dec}| \cdot \max_{\alpha \in \mathbf{I}_{dec}} (\text{len}(\alpha)) \qquad (5)$$

where $\mathbf{I}_{dec}$ is the set of encoded immediate values. $|\mathbf{I}_{dec}|$ indicates the height of ROM for immediate decoder, and $\max_{\alpha \in \mathbf{I}_{dec}}(\text{len}(\alpha))$ indicates the width of it. If there are two or more sets of immediate values for encoding, the same number of immediate decoders are required. Thus, the cost of immediate decoders is the the sum of equation (5) with each set. In the rest of this chapter, the sets of immediate values for encoding are derived for all encoding techniques.

a) Longest format/All Coding (LAC)

All immediate values in the longest instruction format type are targeted for encoding in LAC. Therefore, when the longest instruction format type is assumed to be type j, the set of immediate values $\mathbf{I}_j$ in this type is targeted for encoding in this method.

b) Individual format/All Coding (IAC)

All immediate values in the each instruction format type are individually targeted for encoding in IAC. Therefore, the sets of immediate values $\mathbf{I}_2$, $\mathbf{I}_4$ and $\mathbf{I}_7$ are targeted for encoding in this method.

c) Union format/All Coding (UAC)

All immediate values in each instruction format types are collectively targeted for encoding in UPC. Therefore, the set of immediate values $\bigcup_{i=2,4,7} \mathbf{I}_i$ is targeted for encoding in this method.

d) Longest format/Partial Coding (LPC)

The values which can not be represented in immediate field of the longest format type are targeted for encoding in LPC. When the longest instruction format type is assumed to be type j, the set of encoded immediate values is calculated to L_{red} for LPC (see chapter 3.3). The size of the immediate field in the longest instruction format type shortens by $L_{red}+1$ (1 is the flag bit), when the instruction word length is reduced by L_{red} bits. Therefore, the size of the immediate field after encoding is shown as follows.

$$L_{imm_j} - L_{red} - 1$$

The set of the immediate values which is targeted for encoding in this method is shown as follows by the expression (3).

$$\mathbf{Over}(\mathbf{I}_j, L_{imm_j} - L_{red} - 1)$$

e) Individual format/Partial Coding (IPC)

The values which can not be represented in immediate field of each format type are individually targeted for encoding in IPC. Similar to LPC, the sets of encoded immediate values are calculated to L_{red} for IPC, and the size of the immediate field after encoding is shown as follows.

$$L_{imm_i} + L_{uu_i} - L_{red} - 1$$

The sets of the immediate values which are targeted for encoding in this method are shown as follows by the expression (3).

$$\mathbf{Over}(\mathbf{I}_i, L_{imm_i} + L_{uu_i} - L_{red} - 1) \quad (i = 2, 4, 7)$$

f) Union format/Partial Coding (UPC)

The values which can not be represented in immediate field of each format type are collectively targeted for encoding in UPC. Similar to LPC, the sets of encoded immediate values are calculated to L_{red} for UPC, and the size of the immediate field after encoding is shown as follows.

$$L_{imm_2} + L_{uu_2} - L_{red} - 1$$

The set of the immediate values which is targeted for encoding in this method is shown as follows by the expression (5).

$$\mathbf{Over}'(L_{imm_2} + L_{uu_2} - L_{red} - 1)$$

4.2. A Countermeasure for Delay

A clock period of CPU may become long, if it takes much time to decode the immediate values. The delay of a pipeline stage which performs such decoding may become dominant compared with other stages. One approach to avoid such a performance loss is to introduce a pipeline stage for the decoding can be added. However, in general, the deeper the pipeline, the worse the branch penalty in clock cycles[2]. We do not consider this approach in evaluations of next chapter.

5. Experimental Result

We applied our encoding techniques to three embedded applications, ghostscript, mpeg2 decoder, and mpeg2 encoder. A code size of these application programs are about 42,000, 6,500, and 6,800 lines, respectively. Our used compiler is gcc-dlx [2] which based GNU CC Ver. 2.7.2[6] for DLX architecture[2].

Table 2 shows analysis results of these application programs. The results indicates that the longest instruction format type is usually type 2, and a frequency in use of type 2 instructions is usually dominant for all other types.

Table 1 shows results of area reduction by proposed techniques to these application programs. The followings are summarized from the table.

1. An effective methods for ghostscript are LPC, IPC, and UPC.
2. An effective methods for mpeg2 decoder is IPC.
3. An effective methods for mpeg2 encoder is IPC.

In ghostscript, three kinds of partial coding have same results, because these were finally encoded only the immediate values in the longest format type. In mpeg2 applications, the highest reduction rate is achieved in IAC, though the immediate decoder size is the largest. This is because the IAC method can reduce the instruction word length most.

[2] http://www-mount.ee.umn.edu/~okeefe/mcerg/gcc-dlx.html

Table 1. Reduction of area by proposed techniques

ghostcript	LAC	IAC	UAC	LPC	IPC	UPC
Reduced size of instruction word length[bits]	4	4	3	4	4	4
Reduced area of instruction memory[bits]	69,936	69.936	52,451	69.936	69.936	69.936
Cost of immediate decoder[bits]	35,216	116,418	117,705	26,592	26,592	26,592
Reduction ratio[%]	11.5	10.1	7.05	**11.7**	**11.7**	**11.7**
mpeg2 decoder	LAC	IAC	UAC	LPC	IPC	UPC
Reduced size of instruction word length[bits]	5	6	4	5	5	5
Reduced area of instruction memory[bits]	146,545	175,854	117,236	146,545	146,545	146,545
Cost of immediate decoder[bits]	13,200	38,256	35,760	10,304	10,432	10,432
Reduction ratio[%]	13.8	**14.2**	8.4	14.1	14.1	14.1
mpeg2 encoder	LAC	IAC	UAC	LPC	IPC	UPC
Reduced size of instruction word length[bits]	5	6	4	5	5	5
Reduced area of instruction memory[bits]	181,480	217,776	145,184	181,480	181,480	181,480
Cost of immediate decoder[bits]	13,280	41,109	39,644	10,000	10,128	10,128
Reduction ratio[%]	14.0	**14.7**	8.8	14.3	14.3	14.3

Table 2. An analysis of application programs

	ghost-script	mpeg2 decoder	mpeg2 encoder
Number of			
total instructions	174,984	29,309	36,296
kinds of instructions	104	104	104
registers	29	29	32
immediate values (type 2)	2,201	825	830
immediate values (type 4)	432	346	374
immediate values (type 7)	3,910	1,220	1,285
immediate values (total)	5,753	2,089	2,170
Length of			
opcode field	7	7	7
register field	5	5	5
immediate field (type 2)	16	16	16
immediate field (type 4)	16	16	16
immediate field (type 7)	19	16	17
Effective bit width of			
type 1	22	22	22
type 2	33	33	33
type 3	17	17	17
type 4	28	28	28
type 5	12	12	12
type 6	7	7	7
type 7	26	23	24
Instruction word length	33	33	33
Frequency in use of	[%]	[%]	[%]
type 1	10.5	12.0	13.1
type 2	62.2	63.4	63.0
type 3	2.6	2.6	3.9
type 4	7.2	12.6	11.8
type 5	1.1	0.9	0.7
type 6	6.4	6.2	5.2
type 7	9.9	2.4	2.2
Unused ratio of ROM [%]	13.69	13.35	13.19

6. Conclusion

We presented six instruction encoding techniques for embedded system design which encode immediate values in instructions. We derived the equations which calculate the reduced size of instruction word length for all techniques, and the expression which calculate the reduction ratio of the ROM area including the immediate decoder for all techniques.

We can dramatically reduce the chip area, if our techniques are applied to a program of application specific VLIW processors.

References

[1] A. Grasselli. "The design of program-modifiable microprogrammed control units". *IRE Trans. on EC*, EC-11:336–339, 1962.

[2] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[3] N. Ishiura and M. Yamaguchi. "Instruction Code Compression for Application Specific VLIW Processors Based on Automatic Field Partitioning". In *Proc. of Synthesis and System Integration of Mixed Technologies (SASIMI'97)*, pages 105–109, 1997.

[4] S. Y. Liao, S. Devadas, and K. Keutzer. "Code Density Optimization for Embedded DSP Processors Using Data Compression Techniques". In *Proc. of ARVLSI*, 1995.

[5] R. F. Rosin, G. Frieder, and R. H. E. Jr. "An Environment for Research in Microprogramming and Emulation". *Comm. ACM*, 15(8):748–760, 1972.

[6] R. M. Stallman. *Using and Porting GNU CC for version 2.7.2*. Free Software Foundation, Inc., 1995.

[7] H. Yasuura, H. Tomiyama, A. Inoue, and F. N. Eko. "Embedded System Design Using Soft-Core Processor and Valen-C". *to appear in IIS Journal of Information Science and Engineering*, 14(3), September 1998.

[8] Y. Yoshida, B. Y. Song, H. Okuhata, T. Onoye, and I. Shirakawa. "Low-Power Consumption Architecture for Embedded Processor". In *Proc. 2nd International Conference on ASIC*, pages 77–80, Oct. 1996.