Reduce Power Consumption Of A High Performance Processor Through Gray Code Addressing

Ching-Long Su, Chi-Ying Tsui
and Alvin M. Despain

CENG Technical Report 93-44

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-9143

September 1993

Reduce Power Consumption of A High Performance Processor Through Gray Code Addressing

Ching-Long Su, Chi-Ying Tsui, Alvin M.

Advanced Computer Architecture Labora

University of Southern California

Los Angeles, CA 90089-2562, US

Phone: (213) 740-9143

E-mail: csu@usc.edu

September 15, 1993

93-44

Abstract

Traditional computer architectures are mainly designed for high performance. These high performance computer architectures usually consume considerably amounts of power. Computers designed for low power consumption are in high demand for the rapidly expanding market for portable computing. Design techniques for of low power consumption computers include lowering voltage, lowering capacity, and reducing bit switches.

In this paper, we focus on issues of an architecture design for reducing bit switches. We use a Gray code, in which consecutive memory addresses have only one-bit different, for memory addressing. Due to locality of program execution, this Gray code addressing can significantly reduce bit switches. For a typical program running on a general purpose RISC microprocessor, Gray code addressing has only 50~70% of the bit switches that binary code addressing has.

Binary Code Representation

The traditional number system used for memory addressing is binary coded. The binary code addressing system uses base 2. The two digits number used are 0 and 1. A binary representation 10010110 is interpreted as

$$1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 170$$

Table 1 shows 4-bit binary representations and their corresponding decimal equivalent.

Binary code	Decimal Equivalent	Binary code	Decimal Equivalent
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

Table 1 Binary code representation and decimal equivalent

Gray Code Representation

Gray code sequence is a set of numbers, represented as a combination of digits 1s and 0s, in which contiguous numbers have only one bit different, as shown in Table 2. A formal definition of Gray code sequence is described as follows [Hayes 88],

1. $G_1 = 0, 1$.

2. Let $G_k = g_0, g_1, \ldots, g_{2^k-2}, g_{2^k-1}$. G_{k+1} is formed by first preceding all members of the sequence G_k by 0, then repeating G_k with the order reversed and all members preceded by 1. In other words,

$$G_{k+1} = 0g_0, 0g_1, \ldots, 0g_{2^k-2}, 0g_{2^k-1}, 1g_{2^k-1}, 1g_{2^k-2}, \ldots, 1g_1, 1g_0$$

For example, $G_2 = 00, 01, 11, 10$ and $G_3 = 000, 001, 011, 010, 110, 111, 101, 100$. Clearly the foregoing construction ensures that consecutive members of a Gray code sequence differ in exactly 1 bit.

Gray code	Decimal Equivalent	Gray code	Decimal Equivalent
0000	0	1100	8
0001	1	1101	9
0011	2	1111	10
0010	3	1110	11
0110	4	1010	12
0111	5	1011	13
0101	6	1001	14
0100	7	1000	15

Table 2 Gray code representation and decimal equivalent

Conversion between Binary and Gray Code

The conversion between binary code and Gray code is defined as follows. Let B and G be two different representations of a number (B is a binary representation and G is a Gray code representation) with the same number of bits and the same decimal equivalent.

$$B = <b_{n-1}, b_{n-2},..., b_1, b_0>$$
$$G = <g_{n-1}, g_{n-2},..., g_1, g_0>$$

The conversion of binary code to Gray code is very simple. For any bit in B, said b_i, the converted bit in Gray code representation g_i is the exclusive or of b_i and the next most significant bit b_{i+1}. The most significant bits of binary representation and Gray code representation are the same.

Binary code --> Gray code

$$g_n = b_n$$
$$g_i = b_{i+1} \oplus b_i \quad (i = n-1, 0)$$

For example, let B be a binary number <1,1,0,1> whose decimal equivalent is 13. The values of b_3, b_2, b_1, and b_0 are 1, 1, 0, 1 respectively. The Gray code representation is then $<b_3, b_3\oplus b_2, b_2\oplus b_1, b_1\oplus b_0>$ which is equivalent to <1,0,1,1>. The decimal equivalent of Gray code <1,0,1,1> is 13, shown in Table 2.

Similar to the conversion of binary code to Gray code, the conversion of Gray code to binary code uses the exclusive or operation. However, it is more complex than the conversion from binary code to Gray code. For any bit in G, say g_i. the converted bit in binary code representation g_i is the exclusive or of g_i and all of the most significant bits of G, g_{i+1}, g_{i+2},..., g_n. The most significant bits of the binary representation and Gray code representation are the same.

Gray code --> Binary code

$$b_n = g_n$$
$$b_i = b_{i+1} \oplus g_i \quad (i = n-1, 0)$$

For example, let G be a Gray code number <1,1,0,1> whose decimal equivalent is 9. The values of g_3, g_2, g_1, and g_0 are 1, 1, 0, 1 respectively. The binary code representation is then <g_3, g3⊕g2, g3⊕g2⊕ g1, g3⊕g2⊕g1⊕ g0> which is <1,0,0,1>. The decimal equivalent of Gray code <1,0,0,1> is 9, shown in Table 1.

Bit Switches in Binary Code vs. Gray Code Representation

The bit switches of a sequence of numbers can be significantly different depending on the code representations. Figure 1 shows an example. For the sequence of numbers from 0 to 16, shown in Figure 1(a), there are 31 bit switches when the number are coded in binary representation. There are only 16 bit switches when these numbers are coded in Gray code representation. However, for the sequence numbers <1,3,7,15,14,12,13,9,8,10,11,2,6,4,5,0,16> which is shown in Figure 1(b), there are only 17 bit switches when they are coded in binary representation. There are 29 bit switches when these numbers are coded in Gray code representation.

From the above example, we see how the number of bit switches for a sequence of numbers coded in different representation. For random access patterns, Gray code and binary code have similar performance. Note that the sequence numbers in Figure 1(b) is careful selected favor to binary representation. In general, this special sequence is rather unlikely to happen. For consecutive access pattern, which is usually seen in a general processor which execute consecutive instructions in a basic blocks, Gray code addressing has outstanding performance.

3. How To Use Gray Code Addressing

In the previous section, we presented some examples of the usefulness of Gray code addressing. In this section, we show how to use Gray code addressing.

In a general uni-processor, an instructions is fetched from an addressed pointed to by the program counter. After this instruction is fetched, the program counter is increased by one for the next fetch. The program counter is also modified by branch instructions based on branch conditions. In the binary code addressing system, a binary counter is needed for incrementing the program counter. For branch instruction, the calculated target addresses are written directly into the program counter if the branch is taken. In Gray code addressing system, a Gray code counter is needed for incrementing the program counter. For branch instructions, the calculated target address is directly written into the program counter if the branch is taken. The only difference between binary code and Gray code addressing systems is the instruction field for target addresses. In Gray code addressing system, the instruction field for the target address in a branch

0	00000	00000
1	00001	00001
2	00010	00011
3	00011	00010
4	00100	00110
5	00101	00111
6	00110	00101
7	00111	00100
8	01000	01100
9	01001	01101
10	01010	01111
11	01011	01110
12	01100	01010
13	01101	01011
14	01110	01001
15	01111	01000
16	10000	11000
bit changed	**31**	**16**
(a)	*binary*	*Gray*

1	00001	00001
3	00011	00010
7	00111	00100
15	01111	01000
14	01110	01001
12	01100	01010
13	01101	01011
9	01001	01101
8	01000	01100
10	01010	01111
11	01011	01110
2	00010	00011
6	00110	00101
4	00100	00110
5	00101	00111
0	00000	00000
16	10000	11000
bit changed	**17**	**29**
(b)	*binary*	*Gray*

Figure 1 The bit switches of binary codes vs. Gray codes

instruction is modified such that the calculated target address is the correct target address in Gray code addressing system.

Figure 2 shows an example of branch instructions in binary code and Gray code addressing system. We present a binary code as $<b_n, b_{n-1},...,b_0>_2$ and a Gray code as $<g_n, g_{n-1},..., g_0>_g$, where the bit length is n+1. The correct execution sequence is (*A*,*B*,*br*,*C*,and *D*) where *A*, *B*, *C*, and *D* are instructions and *br* is a taken branch. The target address of the branch is the addition of current program counter and the offset which is specified in the branch instruction. In binary code addressing system, the offset of branch instruction is $<00111>_2$ and the value of the current program counter is $<01110>_2$ which is 14 in decimal representation. The target address of this branch instruction is the binary addition of $<01110>_2$ and $<00111>_2$, which is $<10101>_2$. In the Gray code addressing system, we can implement this branch operation by modifying the offset value to $<10110>_g$ specified in the branch instruction. Since the current program counter is $<01001>_g$ which is equivalent to 14 in decimal representation, the target address of this branch instruction is then $<11111>_g$, which is the binary addition of $<01001>_g$ and $<10110>_g$.

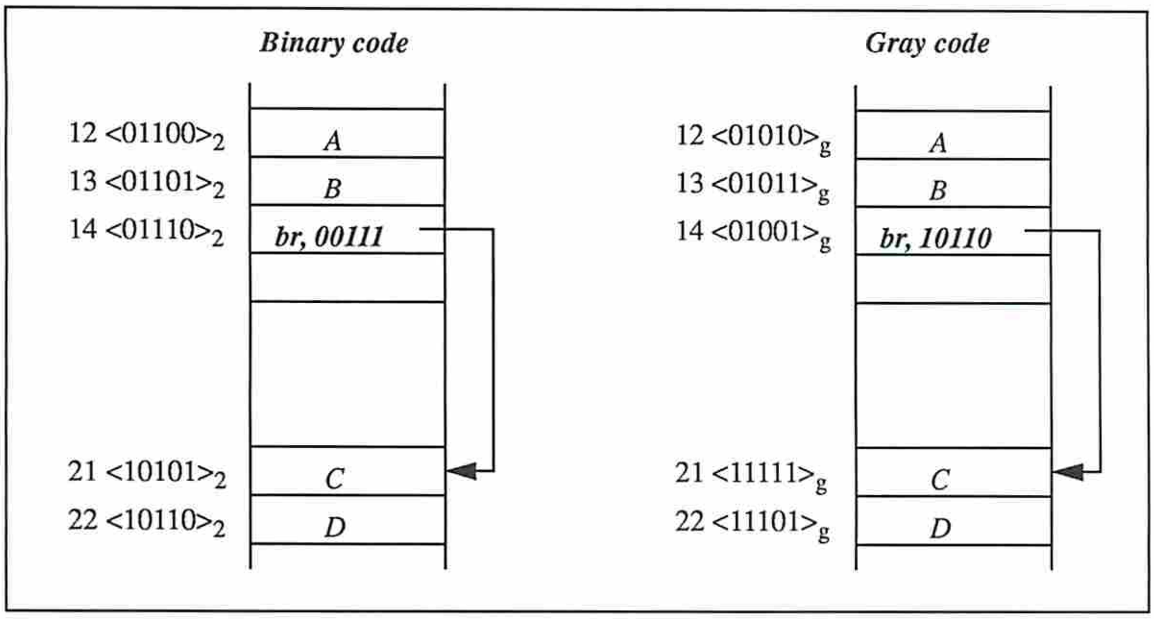

Figure 2 Branch instructions in binary code and Gray code addressing System

4. Results

To validate the benefit of Gray code addressing, we implement a Gray code addressing system on a RISC-like microprocessor, the VLSI-BAM [Holmer 90]. The instruction set of the VLSI-BAM is similar to the MIPS-2000 with some extensions for symbolic computation [MIPS 86].

Benchmark programs used in this paper, listed in Table 3, are selected from the Aquarius benchmark suite [Haygood 89]. Applications of these benchmark programs, include list manipulation, a data base query, a theorem prover, and a computer language parser. Benchmark programs are compiled through the Aquarius Prolog compiler [Van Roy 92] and an optimizing compile backend [Su 92].

Benchmark	Line	Instructions	Description
fastqueens	145	1,138,655	Naive reverse of a 30-element list
qsort	225	4,560	Quicksort of a 50-element list
reducer	397	1,064,197	A parser for language translation
circuit	1315	4,504,940	VLSI module generator
semigroup	4395	4,487,201	Query a data base
nand	985	350,761	A circuit generator
boyer	9918	27,494,723	An extract from a Boyer-Moore theorem prover
browse	1295	18,883,712	Build and query a database
chat	114312	3,303,153	A small subset of English for database querying

Table 3 Benchmark programs

A cycle-by-cycle instruction-level simulator has been built for collecting bit switch counts of memory accesses. We list these bit switch counts in Table 4. For instruction accesses, compared to traditional binary code addressing system, a Gray code addressing system significantly reduces the number of bit switches. For data accesses, bit switches in the binary code and the Gray code addressing system are quite close. This is because instruction addresses are more sequential than data addresses during regular program execution.

Benchmark	Instruction Address in Binary Code	Instruction Address in Gray Code	Data Address in Binary Code	Data Address in Gray Code
fastqueens	2,804,797	1,177,861	973,151	1,038,509
qsort	12,057	6,047	6,011	5,701
reducer	2,731,471	1,817,517	1,566,384	1,484,893
circuit	10,477,307	6,638,057	6,005,394	5,308,314
semigroup	12,028,773	8,933,613	6,174,504	6,010,871
nand	848,899	551,512	438,553	406,609
boyer	76,019,503	57,440,477	48,250,617	47,333,577
browse	47,335,397	30,897,487	24,885,247	26,393,123
chat	8,019,831	5,099,811	4,370,555	3,952,065

Table 4 numbers of Bit Switches for memory addressing

We measure the performance of address coding by the number of Bit switches Per executed Instruction, denoted as BPI. Figure 3 shows the BPI of instruction addresses in binary code and Gray code among benchmark programs. The benchmark program with the most significant reduction of bit switches in benchmark programs through Gray code addressing is *fastqueens*. The BPI of instruction addresses in binary code and Gray code are 2.46 and 1.03 respectively. The reduction of bit switches from Gray code addressing to binary address is more than half (58.13%). In other words, the more instruction locality of a program has, the more bit switch reduction will be. For the worst performance of Gray code addressing among benchmark programs, which is *boyer*, the reduction of bit switches from Gray code addressing to binary code addressing is still significant (24.28%). The average of BPI in Gray code addressing is 1.60. The average of BPI in binary code addressing is 2.53. The average reduction of bit switches from Gray code addressing out of binary addressing is then (1 - 1.60/2.53), which is 36.89%.

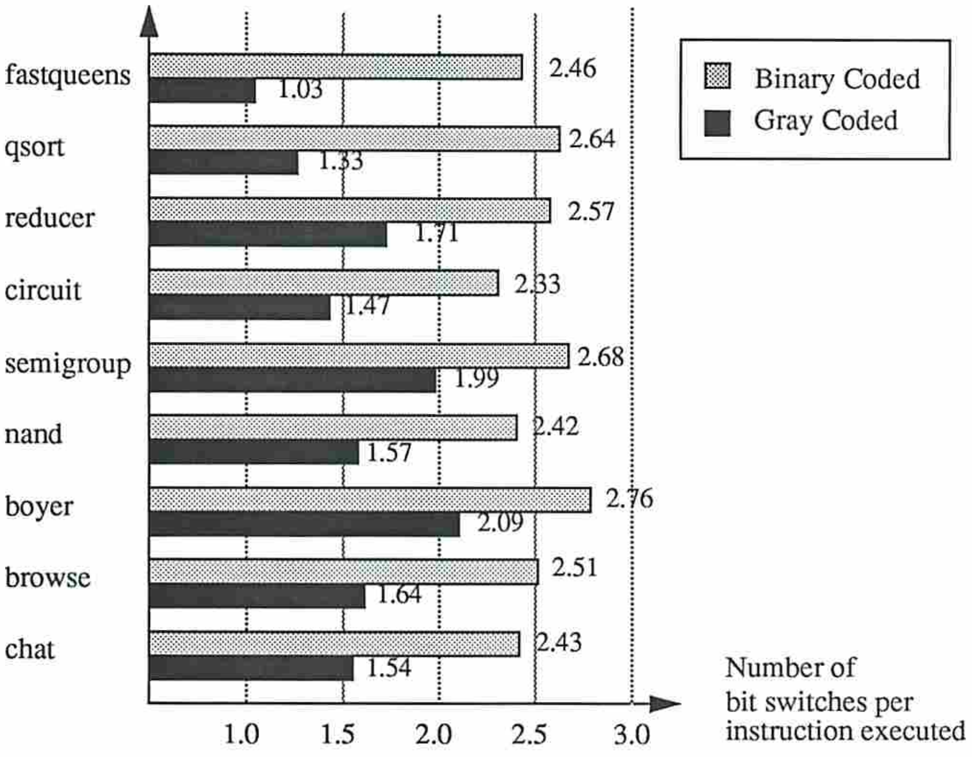

Figure 3. Bit switches of instruction addresses per executed instruction

Figure 3 shows the BPI of data addresses in binary code and Gray code among benchmark programs. Unlike the significant impact of bit switches for instruction addresses by using Gray code addressing, the impact of bit switches for data addresses is not obvious. Actually, the BPI of data addresses in binary code and Gray code are very close. Among benchmark programs which we use in this paper, we find that two out of nine benchmark program (*fastqueen and browse*) have lower BPI in binary code than in Gray code. The other seven benchmark program have slightly larger BPI in binary code than in Gray code. The average of BPI in Gray code addressing is 1.28. The average of BPI in binary code addressing is 1.39. The average reduction of bit switches from Gray code addressing out of binary addressing is then 7.91%.

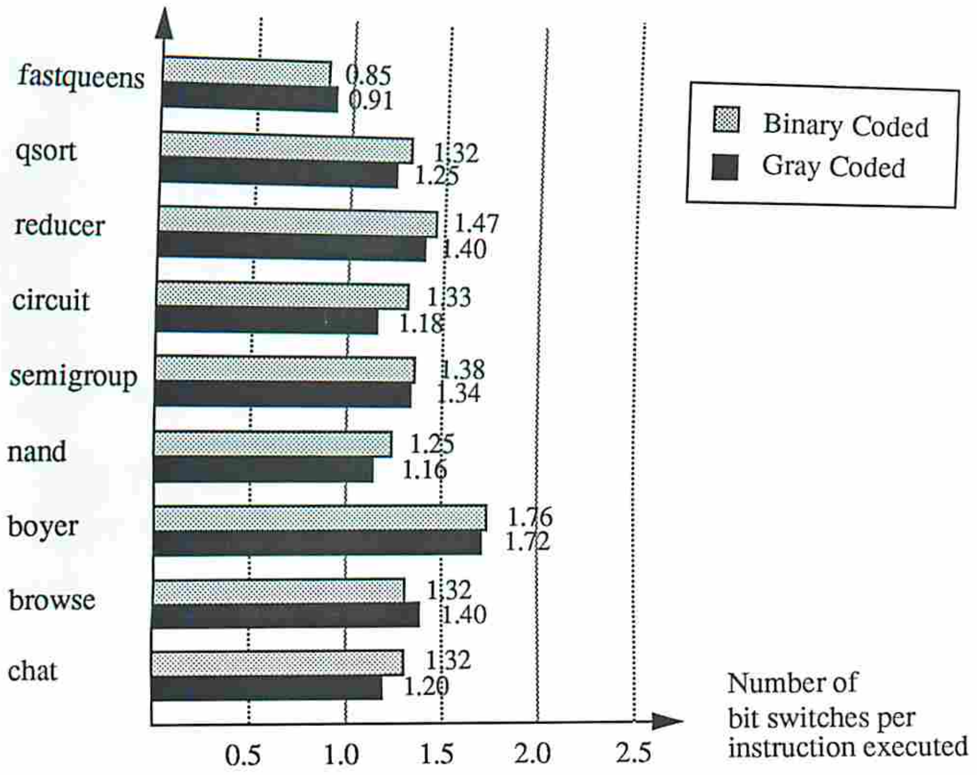

Figure 4. Bit switches of data addresses per executed instruction

5. Conclusion

In this paper, we provide a novel memory addressing system, Gray code addressing, for instruction set processors. Compared to traditional binary code addressing, the Gray code addressing system significantly reduces the bit switches of memory addressing in the execution of programs. The more consecutive addresses referenced, the more bit switches are saved. In order to effectively use the Gray code system, a Gray code counter and an optimizing compiler backend are needed. We demonstrate a simple scheme to adopt the current optimizing compiler backend for the Gray code addressing system.

While portable computing becomes more and more popular, the designs of processors for low power consumption becomes more important. The Gray code addressing takes advantage of consecutive memory accesses during the execution of programs. A significant number of bit switches can be saved.

References

[Haygood 89] R. Haygood, "A Prolog Benchmark Suite for Aquarius," Technical Report, Computer Science Department, University of California, UCB/CSD 89/509, 1989.

[Hayes 88] J.P. Hayes, "Computer Architecture And Organization," McGraw-Hill Int. Editions, 1988.

[Holmer 90] B. Holmer, B. Sano, M. Carlton, P. Van Roy, R. Haygood, W. Bush, and A. Despain. "Fast Prolog with an Extended General Purpose Architecture," The 17th Annual International Symposium on Computer Architecture, May 1990.

[Ghosh 92] A. Ghosh, S. Devadas, K. Keutzer, and J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuit," the 29th DAC, 1992.

[MIPS 86] "MIPS language programmer's guide," MIPS Computer Systems, Inc., 1986

[Su 92] C-L Su, "An instruction Scheduler and Register Allocator for Prolog Parallel Microprocessors," International Computer Symposium, 1992

[Tsui 93] C.Y. Tsui, M. Pedram, and A.M. Despain, "Technology Decomposition and Mapping Targeting Low Power Dissipation, " the 30th DAC, 1993

[Van Roy 92] P. Van Roy and A. M. Despain, "High-Performance Logic Programming with the Aquarius Prolog Compiler," Computer, January 1992.