20 Jan 1975

# Instruction Design to Minimize Program Size

James F. Wade

Paul D. Stigall
*Missouri University of Science and Technology*, tigall@mst.edu

# INSTRUCTION DESIGN TO MINIMIZE PROGRAM SIZE*

James F. Wade
Computer Science Department

Paul D. Stigall
Electrical Engineering and Computer Science Departments
University of Missouri-Rolla
Rolla, Missouri 65401

## Introduction

With the introduction of dynamic microprogrammer computers, it becomes practical to reconfigure the architecture of a computer to more efficiently represent programs. A number of methods have been proposed and investigated for reducing the redundancy of computer op codes, address and data items. The greatest gains have come from frequency based encoding. Wilner[9] reports 40 to 70 percent reduction of program size using frequency based encoding for instructions and addresses combined with tailored instruction sets.

This paper examines the effect on program size of various architectural features assuming frequency based encoding. It is primarily oriented toward the reduction of program memory by selection of instructions and features to microcode, but applicable to the structure of the underlying micro machine and the compressed storage of any type of symbol string.

## Simplest Information Theory Model

A program is assumed to consist of a string of symbols from a given alphabet A of N symbols. Each symbol $S_i$ occurs with probability $P_i$. The probability of occurrence of a symbol is assumed to be independent of previous symbols.

From information theory[1] the information per symbol in bits is given by $-\text{Log}_2(P_i)$. The average information per symbol (entropy) $H = -\sum_{i=1}^{N} P_i \text{Log}_2(P_i)$. The lower bound on the number of bits needed to represent a string of L symbols under these assumptions is then LH. If symbols are coded individually as variable length bit strings with unique prefix encoding, the most efficient encoding is given by Huffman[3].

## Evaluation Method

It will be assumed that the symbol string is encoded minimally redundant assuming independence. After the technique under study is applied the resulting string is assumed to be re-encoded. Thus the method of evaluation becomes computing the entropy of the string after the reduction and comparing with the entropy of the string before reduction. If there is a change in the number of symbols required to represent the string, the entropy is adjusted to normalize to the length of the original string.

*Supported in part by a National Science Foundation Grant, GJ-32596.

## Replacement of Repeated Strings

The simplest and most natural process for reducing the length of a symbol string is the identification of repeated symbol strings and the introduction of a new symbol to replace the string. This is a simple model of subroutines, macros, and the introduction of new instructions.

If one assumed the string $S_1S_2\ldots S_k$ occurs with probability $P_s$ and is replaced by the new symbol S' wherever it occurs, then the new value of entropy is given by:

$$H' = -\frac{P_s}{1-(k-1)P_s}\text{Log}\left(\frac{P_s}{1-(k-1)P_s}\right) - \sum_{i=1}^{k}\frac{P_i-P_s}{1-(k-1)P_s}\text{Log}\left(\frac{P_i-P_s}{1-(k-1)P_s}\right) - \sum_{i=k+1}^{N}\frac{P_i}{1-(k-1)P_s}\text{Log}\left(\frac{P_i}{1-(k-1)P_s}\right). \quad (1)$$

The number of symbols needed to represent a string is reduced by the factor $1-(k-1)P_s$. The resulting change in entropy normalized to the length of the original string is given by:

$$Ha - H = (1-(k-1)P_s)\,H' - H. \quad (2)$$

substituting (1) in (2) gives

$$Ha - H = -P_s\,\text{Log}\left(\frac{P_s}{1-(k-1)P_s}\right) - \sum_{i=1}^{k}(P_i-P_s)\,\text{Log}\left(\frac{P_i-P_s}{1-(k-1)P_s}\right) - \sum_{i=k+1}^{N}P_i\,\text{Log}\left(\frac{P_i}{1-(k-1)P_s}\right) + \sum_{i=1}^{N}P_i\,\text{Log}\,P_i.$$

Which reduces to:

$$Ha - H = (1-(k-1)P_s)\,\text{Log}(1-(k-1)P_s) - \sum_{i=1}^{k}P_i\left(1-\frac{P_s}{P_i}\right)\text{Log}\left(1-\frac{P_s}{P_i}\right) + P_s\sum_{i=1}^{k}\text{Log}\,P_i - P_s\,\text{Log}\,P_s. \quad (3)$$

To determine the value of $P_s$ such that replacement results in a decrease of entropy, solve the inequality $H_a-H<0$ for $P_s$. For an approximate solution to the resulting nonlinear equation, use the Taylor expansion:

$$(1-x)\,\text{Log}_e(1-x) = -x + \frac{x^2}{2} + \text{H.O.T.}$$

to simplify the first two terms in (3). This gives

$$0 > P_s \left( \sum_{i=1}^{k} \text{Log}_e P_i - \text{Log}_e P_s + 1 + \frac{(k-1)^2 P_s}{2} - \sum_{i=1}^{k} \frac{P_s}{2P_i} + \text{H.O.T.} \right).$$

Rearranging yields:

$$\text{Log}_e P_s > \sum_{i=1}^{k} \text{Log}_e P_i + 1 + \frac{(k-1)^2 P_s}{2} - \sum_{i=1}^{k} \frac{P_s}{2P_i} + \text{H.O.T.} \quad (4)$$

For small $P_s$, the Log terms dominate giving:

$$P_s > e \prod_{i=1}^{k} P_i . \quad (5)$$

Thus a decrease in entropy results if $P_s > 2.72 \prod_{i=1}^{k} P_i$.

This result quantifies the intuituve idea that a rarely occurring long string of symbols can often be replaced by a new symbol and save memory, while a shorter string must occur more often to result in savings.

A somewhat less obvious result is that a rarely occurring symbol which can be replaced by a sequence of very frequently occurring symbols can result in a storage saving. This compares with Fosters' observation[8] that a few instructions make up the greatest percentage of most programs and elimination of infrequently used instructions causes little loss of programming power.

## Processor State Change

The second major technique of minimization is the setting of a state which modifies the interpretation of the symbols that follow it. This models changing processor states, changing control program, base register addressing, address bank selection, op code grouping, shift level coding, and other similar techniques.

Let a string of L symbols from an alphabet of size N be partitionable into K contiguous groups such that in each group there is a set $U_j$ of B symbols which appear in no other group. To encode the string, a new symbol is introduced to identify each group and inserted before each group in the string. Each symbol, $S_{ij} \varepsilon U_j$ is replaced by a new symbol $S'_j$. For notational convenience, let $P_{ij}$ the probability of occurrence of $S_{ij} \varepsilon U_j$, let $P_{io}$ for $1 \leq i \leq M = N - KB$ be the probability of occurrence of each symbol not in any set $U_j$. Note that

$$\sum_{i=1}^{B} \sum_{j=1}^{K} P_{ij} + \sum_{i=1}^{M} P_{io} = 1 . \quad (6)$$

The entropy before combining symbols is:

$$H = - \sum_{i=1}^{B} \sum_{j=1}^{K} P_{ij} \text{ Log } P_{ij} - \sum_{i=1}^{M} P_{io} \text{ Log } P_{io} .$$

To calculate the entropy after the encoding, note that the probability of occurrence of a symbol will be reduced by the factor $\frac{L}{L+K}$ by the introduction of the K switch symbols. Thus the probability of occurrence of $S'_j$ becomes

$$\sum_{j=1}^{K} \frac{L}{L+K} P_{ij} . \quad (7)$$

The entropy of the string after encoding is:

$$H' = - \sum_{i=1}^{B} \left( \sum_{j=1}^{K} \frac{LP_{ij}}{L+K} \right) \text{Log} \left( \sum_{j=1}^{K} \frac{LP_{ij}}{L+K} \right) - \sum_{i=1}^{M} \frac{LP_{io}}{L+K} \text{Log} \frac{LP_{io}}{L+K}$$

$$-K \left(\frac{1}{L+K}\right) \text{Log} \left(\frac{1}{L+K}\right) .$$

To compare, it is necessary to adjust $H'$ for the additional symbols introduced. Adjusted entropy is:

$$H_a = \frac{L+K}{L} H', \text{ or}$$

$$H_a = - \sum_{i=1}^{B} \left( \sum_{j=1}^{K} P_{ij} \right) \text{Log} \left( \sum_{j=1}^{K} \frac{LP_{ij}}{L+K} \right) - \sum_{i=1}^{M} P_{io} \text{ Log } \left(\frac{LP_{io}}{L+K}\right) - \frac{K}{L} \text{Log} \left(\frac{1}{L+K}\right) .$$

Isolating terms involving $\frac{L}{L+K}$ yields

$$H_a = - \sum_{i=1}^{B} \left( \sum_{j=1}^{K} P_{ij} \right) \text{Log} \left( \sum_{j=1}^{K} P_{ij} \right) - \left(\text{Log} \frac{L}{L+K}\right) \sum_{j=1}^{B} \sum_{j=1}^{K} P_{ij} - \sum_{i=1}^{M} P_{io} \text{ Log } P_{io} - \left(\text{Log} \frac{L}{L+K}\right) \sum_{i=1}^{M} P_{io} - \frac{K}{L} \text{Log} \left(\frac{1}{L+K}\right) . \quad (8)$$

Recalling (6) that $\sum_{i=1}^{B} \sum_{j=1}^{K} P_{ij} + \sum_{i=1}^{M} P_{io} = 1$, and substituting in (8) yields:

$$H_a = - \sum_{i=1}^{B} \left( \sum_{j=1}^{K} P_{ij} \right) \text{Log} \left( \sum_{j=1}^{K} P_{ij} \right) - \sum_{i=1}^{M} P_{io} \text{ Log } P_{io} - \text{Log} \frac{L}{L+K} - \frac{K}{L} \text{Log} \left(\frac{1}{L+K}\right) .$$

The last two terms may be regrouped to

$$\frac{K}{L} \text{ Log } L + \left(1+\frac{K}{L}\right) \text{Log} \left(1+\frac{K}{L}\right) .$$

The change in entropy becomes

$$H_a - H = -\sum_{i=1}^{B}\left(\sum_{j=1}^{K} P_{ij}\right) \operatorname{Log}\left(\sum_{j=1}^{K} P_{ij}\right) + \sum_{i=1}^{B}\sum_{j=1}^{K} P_{ij} \operatorname{Log} P_{ij} + \frac{K}{L} \operatorname{Log} L + \left(1 + \frac{K}{L}\right) \operatorname{Log}\left(1 + \frac{K}{L}\right). \quad (9)$$

Examining equation (9), we note that the first two terms represent the savings generated by combining symbols, and the last two terms represent the cost of introducing the K switch symbols. The first term is sensitive to the way the symbols in each independent group are paired. If the $S_{ij}$ are sequenced so that $P_{1j} \geq P_{2j} \geq \ldots \geq P_{3j}$ for every j, then the first term in (9) will be minimized.

Under the above ordering assumption a more usable expression can be derived by using an approximation. The ratios $P_{i1}:P_{i2}: \ldots :P_{iK}$ can be assumed to be approximately independent of i. Thus $P_{ij} \simeq Q_i\ (1 + D_j)$. Where $Q_i = \frac{1}{K}\sum_{j=1}^{K} P_{ij}$ = average of $P_{ij}$ over j. And

$$D_j = \frac{\frac{1}{B}\sum_{i=1}^{B} P_{ij}}{\frac{1}{BK}\sum_{i=1}^{B}\sum_{j=1}^{K} P_{ij}} - 1$$

Using this approximation:

$$H_a - H = -\sum_{i=1}^{B} KQ_i \operatorname{Log} KQ_i + \sum_{i=1}^{B}\sum_{j=1}^{K} Q_i(1+D_j) \operatorname{Log}(Q_i(1+D_j)) + \frac{K}{L} \operatorname{Log} L + \left(1 + \frac{K}{L}\right) \operatorname{Log}\left(1 + \frac{K}{L}\right).$$

Now $\sum_{i=1}^{B} Q_i = \frac{1}{K}\sum_{i=1}^{B}\sum_{j=1}^{K} P_{ij}$, and $\sum_{j=1}^{K} D_j = 0$.

Regrouping and substituting yields:

$$H_a - H = \frac{1}{K}\left(\sum_{i=1}^{B}\sum_{j=1}^{K} P_{ij}\right)\left(-K \operatorname{Log} K + \sum_{j=1}^{K}(1+D_j) \operatorname{Log}(1+D_j)\right) + \frac{K}{L} \operatorname{Log} L + \left(1+\frac{K}{L}\right) \operatorname{Log}\left(1+\frac{K}{L}\right). \quad (10)$$

Setting $H_a - H < 0$, and rearranging yields:

$$\sum_{i=1}^{B}\sum_{j=1}^{K} P_{ij} \geq \frac{\frac{K}{L} \operatorname{Log} L + \left(1+\frac{K}{L}\right) \operatorname{Log}\left(1+\frac{K}{L}\right)}{\operatorname{Log} K - \frac{1}{K}\sum_{j=1}^{K}(1+D_j) \operatorname{Log}(1+D_j)}. \quad (11)$$

Equation (11) gives the conditions for a savings in storage. Table 1 is a tabulation of equation (11). All $D_j$ are assumed to be 0. This gives a lower bound on the break even point. For distributions with $D_j$ of significant size, the values will be somewhat higher.

NUMBER OF GROUPS (K)

| L | 2 | 3 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|---|
| 10 | .98 | - | - | - | - | - |
| 20 | .58 | .56 | .59 | .69 | .80 | .91 |
| 50 | .28 | .27 | .29 | .33 | .38 | .43 |
| 100 | .16 | .15 | .16 | .19 | .22 | .25 |
| 200 | .09 | .09 | .09 | .11 | .12 | .14 |
| 500 | .04 | .04 | .04 | .05 | .06 | .06 |

Table 1. Break Even Point

It is seen that for a short string, the symbols involved in the switch must account for most of the probability before any savings in storage is realized. For longer strings the break even point is quite low.

$\Sigma\Sigma P_{ij}$

| L | .2 | .4 | .6 | .8 | 1.0 |
|---|---|---|---|---|---|
| 10 | - | - | - | - | .02 |
| 20 | - | - | .02 | .22 | .42 |
| 50 | - | .12 | .32 | .52 | .72 |
| 100 | .04 | .24 | .44 | .64 | .84 |
| 200 | .11 | .31 | .51 | .71 | .91 |
| 500 | .16 | .36 | .56 | .76 | .96 |

Table 2. Maximum Savings in Bits/Symbol for Number of Groups (K) = 2.

Table 2 is a tabulation of equation (10) for K = 2. Assuming $D_1 = D_2 = 0$ gives an upper bound on the savings. Using Log base 2 will give the savings in bits per symbol.

A savings of 0.5 bits/symbol represents about a 10 per cent reduction of a typical program. Thus we see that this type of feature saves significant amounts of storage only if the group sizes are on the order of 100 symbols.

## Limitations and Extensions

An important limitation is the ignoring of implementation cost. While one may rationalize that the cost of hardware is rapidly decreasing, the implementation of features that result in very small savings is not cost effective.

The derivations can be modified to more closely model specific architectural features for more precise results.

## Conclusions

This type of analysis helps evaluate the worth of a particular type of architectural feature. It shows what memory savings are possible over those obtained by frequency

based encoding. It finds the limits on memory savings for a particular type of feature, and the conditions under which the most savings are realized.

## References


1. Shannon, C. E. "A Mathematical Theory of Communication," Bell System Tech. J. 27 (1948), 379-423, 623-656.
2. Oliver, B. M., "Efficient Coding," Bell System Tech. J. 21, 4 (July 1952), pp. 724-750.
3. Huffman, D. A. "A Method for the Construction of Minimum Redundancy Codes," Proc. IRE 40 (1952) pp. 1098-1101.
4. Schwartz, E. S., "A Dictionary for Minimum Redundancy Encoding," J. ACM 10, 4 (Oct. 1963) pp. 413-439.
5. Schwartz, E. S. and Kallick B., "Generating a Cannonical Prefix Encoding," CACM 7, 3 (Mar. 1964).
6. Young, J. F., Information Theory, Wiley Interscience, New York, 1971.
7. Foster, C. C., and Gonter, R. H., "Conditional Interpretation of Operation Codes," IEEE Trans. Computers C-20, 1 (Jan. 1971) pp. 108-111.
8. Foster, C. C., Gonter, R. H., and Riseman, E. M., "Measures of Op-Code Utilization," IEEE Trans. Computers C-20, 5 (May 1971) pp. 582-584.
9. Wilner, W. T., "Burrough B1700 Memory Utilization," AFIPS Conference Proceedings 41, (Dec. 1972) pp. 579-586.


## Appendix

### Example of Frequency Based Coding

A tabulation of 1900 opcodes from text of programs for the MICRODATA 1621 computer was made. The MICRODATA 1621 has 8 bit opcode and variable length address and immediate fields. Of the approximately 250 defined opcodes, 132 appeared in the sample. A tabulation of results using Huffman coding with different number bases and encoding into fixed size bit fields is given. The split into bit fields is chosen to minimize average bits/symbol for the given maximum length.

| | Average Bits/Symbol | Max Length in Bits |
|---|---|---|
| Entrophy | 5.90 | ——— |
| Huffman | | |
| base 2 | 5.93 | 11 |
| base 4 | 6.02 | 12 |
| base 8 | 6.19 | 12 |
| base 16 | 6.32 | 12 |
| bit fields | | |
| 5 - 4 | 6.67 | 9 |
| 6 - 4 | 6.59 | 10 |
| 6 - 5 | 6.61 | 11 |
| 6 - 6 | 6.66 | 12 |
| 5 - 2 - 2 | 6.48 | 9 |
| 5 - 2 - 3 | 6.25 | 10 |
| 5 - 3 - 3 | 6.18 | 11 |
| 5 - 3 - 4 | 6.16 | 12 |
| 5 - 3 - 5 | 6.16 | 13 |
| 5 - 3 - 6 | 6.16 | 14 |