On Comparing Apples and Oranges, or,

My Machine is Better Than Your Machine

B. W. Kernighan and P. J. Plauger
Bell Laboratories, Murry Hill, N. J.

D. J. Plauger
Bucknell University, Lewisburg, Pa.

Reprinted from

Performance Evaluation Review of the
SICME of the ACM

September 1972

ON COMPARING APPLES AND ORANGES, or, MY MACHINE IS BETTER THAN YOUR MACHINE

B.W. Kernighan and P.J. Plauger
Bell Laboratories, Murry Hill, N.J.

D.J. Plauger
Bucknell University, Lewisburg, Pa.

ABSTRACT

In a recent comparison test, six computer manufacturers were asked to code a particular program loop to run as quickly as possible on their machine. Presumably conclusions about the merits of the machines were to be drawn from the resulting code.

We have reduced the number of Instructions for the loop by an average of one instruction per machine, a 15% decrease. It appears that conclusions might more appropriately be drawn about manufacturers' software.

INTRODUCTION

A letter by Jacob Palme in the June 1972 issue of Performance Evaluation Review (titled "Beware of the Gibson Mix") contains the program fragment:

```
e:=p;
loop:  if e ≠ 0 then
begin f:= (t and area(e+1));
   if f ≠ 0 then area(e+2) := area(e+2)+s;
   e := area(e); goto loop;
end;
```

According to Palme, computer manufacturers were asked to code this in assembly language so the inner loop would run as fast as possible. The number of instructions in this loop was listed as:

DEC PDP-10	4
Honeywell Bull G 625	7
IBM 360/370	7
Xerox Sigma 5	6
Scanips RC 4000	7
Univac 1106	7

Palme concludes that "mean execution time per instruction" is not a valid comparison parameter given such wide variation in numbers of instructions.

We find this conclusion rather weak. Past experience with manufacturer-supplied software alerted us to the possibility that the numbers cited might not necessarily be gospel. Any red-blooded programmer should rise to the challenge of improving on those numbers. Since we had an intimate knowledge of two of the machines listed, a nodding acquaintance with two others, and reference manuals for all, we took up the gauntlet. The results proved interesting.

All of the machines are essentially word-binary with multi-functional registers, so the general approach is consistent. With suitable initialization, coding the loop falls into three pieces - computing and testing f (which is a local variable), conditionally updating area (e+2) and chaining via e until a null pointer is found. If only indexed loads and conditional branches are available, eight instructions are required:

```
LOOP    LOAD, TEMP      AREA + 1, E          piece 1
        AND, TEMP       T
        BZ              LINK
        LOAD, TEMP      AREA +2,E            piece 2
        ADD,TEMP        S
        STORE,TEMP      AREA+2,E
LINK    LOAD,E          AREA+0,E             piece 3
        BNZ             LOOP
```

The trickery comes in using peculiar instructions to do two or more of these standard operations at once. In the examples that follow, we may present assembly code of a curious cast, since we don't know all the languages, but the intent should be clear in each case. Let us take the specimens in order.

DEC PDP-10

(Includes PDP-6) This one is indeed optimum; there is no way to beat four instructions:

```
LOOP    TDNE,RT     AREA+1,E        piece 1
        ADDM,RS     AREA+2,E        piece 2
        MOV,E       AREA+0,E        piece 3
        JN          LOOP
```

TDNE skips if the AND of storage and register RT is zero, without changing RT. Since ADDM likewise implements piece two without destroying RS, the skip can leap the entire clause. Piece three offers no savings.

Honeywell Bull G 625

(Includes GE/Honeywell 600/6000 series) The straightforward way is to put T in the A register, S in Q, and E in X1. Then the loop requires five instructions:

```
LOOP    CANA        AREA+1,1        piece 1
        TZE         LINK
        ASZQ        AREA+2,1        piece 2
LINK    LDX1        AREA+0,1        piece 3
        TNZ         LOOP
```

We can think of no plausible way to stretch this to seven instructions.

If execution speed is truly the dominant requirement, however, one should use a Repeat Link (RPL) Instruction, a baroque concoction that can be made to chain down a linked list executing Comparative AND's at data-fetch speed. T

must now be 18 bits long, which seems to be within the rules of the game: and the links must be absolute, which may not be sporting. S may be anything up to 36 bits. The basic loop requires four instructions, but is limited to lists of length less than 257; to make it truly general takes more dressing up than we care to write down.

IBM 360/370

Here we have two choices for implementing each piece. We present the brute force sequence first:

```
LOOP    L       X,AREA+4(E)     piece 1
        NR      X,RT
        BZ      LINK
        L       X,AREA+8(E)     piece 2
        AR      X,RS
        ST      X,AREA+8(E)
LINK    L       E,AREA+0(E)     piece 3
        LTR     E,E
        BNZ     LOOP
```

Note that an extra instruction is required because load from memory does not set the condition code. A concise (five instruction) version of the loop is:

```
LOOP    TM      **,AREA+4(E)     piece 1
        BZ      LINK
        AP      AREA+8(,E),S     piece 2
LINK    L       E,AREA+0(E)      piece 3
        BXH     E,LOOP,RC
```

Piece one requires that T be at most eight bits, which are written or compiles into the appropriate part of the Test Under Mask Instruction. Piece two requires that S be packed decimal. Piece three requires that RC be an odd-numbered register which is initialized to zero to permit a rapid decrement and jump if high on register E.

One can argue strongly against rewriting instructions these days, but there can be no quibble about the mixed data types, since AREA clearly contains types POINTER, LOGICAL and ARITHMETIC cheek by jowl to begin with. We conclude that the manufacturer-supplied version likely consists of our brute force pieces one and three, using packed decimal for piece two to keep the instruction count down. The opposite set of choices also adds to seven instructions, however, and it is possible that these represent the manufacturer-supplied code. A third possibility uses the Load Multiple instruction to fetch AREA+1 and AREA+2 simultaneously; it is left as an exercise to verify that this also takes seven instructions.

Xerox Sigma 5

(Includes Sigma 7, Sigma 9) There is a straightforward five-instruction solution:

```
LOOP    CW,RT     AREA+1,E      piece 1
        BAZ       LINK
        AWM,RS    AREA+2,E      piece 2
LINK    LW,E      AREA+0,E      piece 3
        BNEZ      LOOP
```

The Compare Word nicely performs an AND on the side while doing the algebraic compare, and the Add Word to Memory instruction behaves as in the PDP-10. Again one can only guess at how the manufacturer found a need for one more instruction.

Scanips RC 4000

This machine is handicapped by having only an unconditional branch and an incomplete set of conditional skips. Putting T in W1 and E in W2, brute force thus gives:

```
LOOP    SZ,W1     *AREA+2,W2    piece 1
        JL,0      ADD
        JL,0      LINK
ADD     RL,W0     AREA+4,W2     piece 2
        WA,W0     S
        RS,W0     AREA+4,W2
LINK    RL,W2     AREA+0,W2     piece 3
        SN,W2     0
        JL,0      LOOP
```

The extra branch in piece one occurs because no inverse exists for the instruction Skip if register bits Zero (comparative AND). The only way we could get the Instruction count down to seven was by increasing the average execution time:

```
LOOP    RL,W0     AREA+4,W2     piece 1
        WA,W0     S
        SZ,S1     *AREA+2,W2
        RS,W0     AREA+4,W2     piece 2
        RL,W2     AREA+0,W2     piece 3
        SN,W2     0
        JL,0      LOOP
```

Here the addition is performed unconditionally, but the result is stored under control of the SZ Instruction.

Univac 1106

(Includes Univac 1108) Here we have a handful of "upper" Instructions which perform an operation between an A register and memory, then deliver the result to A+1. So if we load T into the mask register (R2), zero into AZ and S into AS:

```
LOOP    MLU,AZ      AREA+1,E        piece 1
        JZ,AZ+1     LINK
        AU,AS       AREA+2,E        piece 2
        SA,AS+1     AREA+2,E
LINK    LA,E-12     AREA+0,E        piece 3
        JNZ,E-12    LOOP
```

The Masked Loader Upper Instruction delivers T AND memory to AZ+1 for the subsequent test jump. Likewise Add Upper performs S plus memory to AS+1. It is assumed for convenience that the X register used for E is one of the top four that overlap the bottom four A registers. Otherwise a Jump Greater and Decrement (JGD) is required to test E, and the Indexed addresses must be adjusted accordingly. In either case we save one instruction over the manufacturer's version.

Conclusions

We have succeeded in reducing the number of instructions required for the inner loop given above by an average of one per machine. With the possible exception of the IBM 360/370, our shorter code should also execute faster. It is important to note, however, that speed improvements are sometimes obtained by adding instructions - certainly in the case of the RC 4000 and possibly with the IBM 360/370.

Comparing machines is difficult enough when it is clear what is being measured. Asking for minimum execution time, then complaining about the variation in instruction count obtained, is worse than adding apples and oranges. Nor is there any merit in comparing apple/orange ratios (though this appears much more scholarly), for apples come in different sizes and oranges are often artificially colored.