digital INTEROFFICE MEMORANDUM

SUBJECT: Extension of PDP-11 Instruction Set - Floating pt.

DATE: December 22, 1969

TO: Bruce Delagi

FROM: Grant Saviers

LOCATION:

LOCATION:

Assume that a floating pt processor (Multiply, Divide, Add, Subtract, Float, Entier) is interfaced to the unibus.

Associate 6 memory (I/O) locations with the floating pt functions. Use the Move instruction to extend the instruction set, as follows:

MOV address, function

where function is one of the six addresses associated with the FP processor. The output word is an address pointer to a 3 word table of operand addresses.

| Address | Location of operand 1 |
|---|---|
| " | " " 2 |
| " | " result |

The FP processor then obtains operands and stores the result.

For multiple word operands, this technique is faster and requires less core than distinct moves of addresses or operands.

Hand translating the expression.

$$F = (A + (B * C)/(A + B)) * F$$

| Results | Code | | Address Pointers |
|---|---|---|---|
| | Move z, mult | z | B |
| | | | C |
| | Move y, add | | $t_1$ |
| | Move x, div | y | A |
| | Move v, mult | | B |
| | | | $t_2$ |
| | | x | $t_1$ |
| | | | $t_2$ |
| | | w | $t_3$ |
| | | | A |
| | | v | $t_4$ |
| | | | F |
| | | | F |
| this would change example to | | y | A |
| | | | B |
| | | x | $t_2$ |
| | | | $t_1$ |
| | | w | $t_3$ |

Note that the code is automatically "optimized" to eliminate unnecessary saving of results ($t_n$'s) and that no Moves are required, even for the replacement (=) operation.

This technique has definite possibilities for common subexpression evaluation/optimization.

It is clear that the I/O devices should have access to the condition codes for this to really work.

For the anti-symmetric operators (-)/ it may not cost much (in terms of instruction codes) to have the backwards operators.

| | | address list | |
|---|---|---|---|
| e.g. | normal/ | A<br>B<br>C | C = A/B |
| | backwards/ | A<br>B<br>C | C = B/A |

G.E. apparently thought backwards operators useful enough to include in instruction sets, to avoid swapping registers/core. With 3 address instructions they may be superfluous.

Note that the end result looks suspiciously like a "stack" machine with operators in one stack and operands in the other.

I mentioned this technique to Ad van de Goor. He proposed putting operators on same stack, but note that you still must have an instruction to initiate the operation.

As a generalization, I believe one should always pass the address at a descriptor table, so that the table may be completely general. This also has the advantage that only one word is moved.

Ad suggests a micro-programmed arithmetic processor for things like matrix multiply or even implementation of the floating point. Note that for a 3 address pointer table, the instructions and table are independent of arithmetic precision. Only actual reserved operand locations are different.

Why mess around with other machines when such things are possible on the 11?

/bca