

Published: 05/19/67

Identification

Do Statement

B. P. Goldberg

Introduction

The basic pattern of the do statement coding is the same for all types of expressions. The expressions used in the examples below were chosen for simplicity. (See Section BN.6.03 for a description of the evaluation of other types of expressions.) A simple assignment statement ($a = a;$) is used as the program element in each example. Assume that a is a single-precision floating-point number and is defined as the first variable in the program; i.e., $xx0026$.

Use of a Variable or a Signed Constant in the "by expression"Source Code: do $i = j$ to k by 1 : $a = a;$

end;

Pass 1 generates the following code:

	<u>Macro</u>	<u>Comment</u>
ldfx	<u>alias1</u> , <u>bits1</u> ,0,xxx,int,auto,0, <u>level</u> ,0	Loads j
stfx	<u>alias2</u> , <u>bits2</u> ,0,xxx,int,auto,0, <u>level</u> ,0	Stores j in i
golb	<u>alias3</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Transfers to test
dclb	, <u>alias4</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	
ldfx	<u>alias2</u> , <u>bits2</u> ,0,xxx,int,auto,0, <u>level</u> ,0	Loads i
adfx	<u>bits2</u> ,0, <u>alias5</u> , <u>bits5</u> ,0,xxx,int,auto,0, <u>level</u> ,0	Adds 1 to i
fxfx	<u>bitsn</u> ,0,17,0	Changes precision of result
stfx	<u>alias2</u> , <u>bits2</u> ,0,xxx,int,auto,0, <u>level</u> ,0	Stores result in i
dclb	, <u>alias3</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	
ldfx	<u>alias2</u> , <u>bits2</u> ,0,xxx,int,auto,0, <u>level</u> ,0	Loads i

sbfx	<u>bits2,0,alias6,bits6,0,xxx,int,auto,0,level,0</u>	} Subtracts k Negates difference if $l \geq 0$ Transfers if difference < 0 ; i.e., satisfied do Program element
dofx	<u>alias5,bits5,0,xxx,int,auto,0,level,0</u>	
outlb	<u>alias7,144,0,xxx,con,xxxx,0,level,0</u>	
ldfl	xx0026,27,0,xxx,int,auto,0, <u>level,0</u>	} Program element
stfl	xx0026,27,0,xxx,int,auto,0, <u>level,0</u>	
gogolb	<u>alias4,144,0,xxx,con,xxxx,0,level,0</u>	} Transfers to incrementation coding
dclb	, <u>alias7,144,0,xxx,con,xxxx,0,level,0</u>	

Pass 2 translates the macros shown above as follows:

	lda	sp <u>alias1</u>	} Initialization of i
	sta	sp <u>alias2</u>	
	tra	<u>alias3</u>	
<u>alias4:</u>	null	"	
	lda	sp <u>alias2</u>	} Incrementing of i
	ada	sp <u>alias5</u>	
	sta	sp <u>alias2</u>	
<u>alias3:</u>	null	"	
	lda	sp <u>alias2</u>	} Testing of i
	sba	sp <u>alias6</u>	
	szn	sp <u>alias5</u>	
	tmi	*+2	
	neg		
	als	0	
	tmi	<u>alias7</u>	
"			
	fld	sp xx0026	} Program element
	fst	sp xx0026	
"			
	tra	<u>alias4</u>	Next iteration
<u>alias7:</u>	null	"	

Use of an Unsigned Constant in the "by expression"

Source Code: do i = j to k;
 a = a;
 end;

In this example, the expression "by 1" is understood. The initialization and iteration coding is similar to that for Case 1. However, the testing

is different. Here Pass 1 produces the following macros to determine whether the do is satisfied and to transfer out of the loop:

<u>Macros</u>	<u>Comment</u>
ldfx <u>alias1</u> , <u>bits1</u> ,0,xxx,int,auto,0, <u>level</u> ,0	Loads i
iflfix <u>bits1</u> ,0, <u>alias2</u> , <u>bits2</u> ,0,xxx,int,auto,0, <u>level</u> ,0	If $i \leq$ limit, then continues;
golb <u>alias3</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	otherwise, do is satisfied.

Pass 2 then produces the following code:

<u>Instruction</u>	<u>Comment</u>
lda sp <u>alias1</u>	
cmpa sp <u>alias2</u>	
tze *+2	Transfers to program element
tpl <u>alias3</u>	Transfers out of loop

While Clause

Source Code: do while (i = j);

Here Pass 1 produces the following macros to determine whether the do is satisfied and to transfer out of the loop:

<u>Macros</u>	<u>Comment</u>
ldfx <u>alias1</u> , <u>bits1</u> ,0,xxx,int,auto,0, <u>level</u> ,0	Loads i
ifeqfx <u>bits1</u> ,0, <u>alias2</u> , <u>bits2</u> ,0,xxx,int,auto,0, <u>level</u> ,0	If $i = j$, then continues;
golb <u>alias3</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	otherwise do is satisfied.

Pass 2 translates these macros as follows:

```
lda sp|alias1
cmpa sp|alias2
tnz sp|alias3
```

Do Statement with a Simple Specification

```
Source Code :   do i = 1;
                a = a;
                end;
```

Pass 1 first produces macros to set i equal to 1. It then generates the following code:

	<u>Macro</u>	<u>Comment</u>
golb	<u>alias1</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Goes to program element
dclb	, <u>alias2</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	
golb	<u>alias3</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Dummy test
dclb	, <u>alias1</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	
ldf1	xx0026,27,0,xxx,int,auto,0, <u>level</u> ,0	
stf1	xx0026,27;0,xxx,int,auto,0, <u>level</u> ,0	Program element
gogolb	<u>alias2</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Goes to dummy test
dclb	, <u>alias3</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	

Here the do is satisfied immediately after the computation is performed. Therefore, no testing is necessary.

Pass 2 then produces the following code:

	<u>Instruction</u>	<u>Macro</u>
	tra <u>alias1</u>	golb
<u>alias2</u> : null	"	dclb
	tra <u>alias3</u>	golb
<u>alias1</u> : null	"	dclb
"	fld sp xx0026	
"	fst sp xx0026	program element
	tra <u>alias2</u>	gogolb
<u>alias3</u> : null	"	dclb

Multiple Specifications

```
Source Code:  do i = 4,9,16,25;
               a = a;
               end;
```

Pass 1 first creates macros to set $i = 4$. It then generates the following code:

<u>Macro</u>	<u>Comment</u>
dflb , <u>alias1</u> ,144,0,xxx,0,int,auto,0, <u>level</u> ,0	Defines temporary location
ldlb <u>alias2</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	} Sets up transfer from program element
stlb <u>alias1</u> ,144,0,xxx,int,auto,0, <u>level</u> ,0	
golb <u>alias3</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Transfers to program element
dclb , <u>alias2</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Program element transfers here:
golb <u>alias4</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Transfers to code for next specification
dclb , <u>alias3</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Goes to program element
golb <u>alias5</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	
dclb , <u>alias4</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	
coding for remaining specifications	
dclb , <u>alias5</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	
ldfl xx0026,27,0,xxx,int,auto,0, <u>level</u> ,0	Program element
stfl xx0026,27,0,xxx,int,auto,0, <u>level</u> ,0	
gogolb <u>alias1</u> ,144,0,xxx,int,auto,0, <u>level</u> ,0	Transfers to code for next specification
dclb , <u>alias2</u> ,144,0,xxx,con,xxxx,0, <u>level</u> ,0	Declares label for transfer to next statement

The gogolb macro transfers to the "contents of " alias1; i.e., alias2. Again, since there is no test involved, an immediate transfer is made to the coding for the next specification. This has the same pattern as the coding shown above. The temporary defined by alias1 is used for each specification, but it is only defined once.

This process is shown more graphically by the Pass 2 coding:

	<u>Instruction</u>	<u>Macro</u>
	equ <u>alias1,location</u>	df1b
	tra s.x	
s.x:	eapbp <u>alias2</u>	} ldlb
	stpbb sp .u1+4	
	stpsp sp .u1+6	
	eapbp sp <u>alias1</u>	
	ldaq sp .u1+4	} stlb
	staq bp 0	
	ldaq sp .u1+6	
	staq bp 2	
	tra <u>alias3</u>	golb
<u>alias2:</u>	null #	dclb
	tra <u>alias4</u>	golb
<u>alias3:</u>	null "	dclb
	tra <u>alias5</u>	golb
<u>alias4:</u>	null "	dclb
	⋮	
<u>alias5:</u>	null "	
"	fld sp xx0026	program element
	fst sp xx0026	
"		
	tra sp <u>alias1,*</u>	gogolb
<u>alias9:</u>	null "	dclb