516-49
RFG
5/19/71

Subject: 516 Segment Assembler.

This document explains some of the details which come into play when assembling segmented programs, using the DDP segment assembler described in Doc. #516-41.

The following mnemonics are available to effect a change of control from one area of code to another.

JMP - Hardware interpreted intrasegment jump instruction.

GOTO - Software interpreted macro instruction for intersegment jumps (it may be used for an intrasegment jump, but it is inefficient).

Example: To continue execution at the fifth location of segment "AB".

Coding Example	Equivalent Expansion
.	.
.	.
.	.
GOTO AB,4	JST .GOTO.,*
	VADDR AB,4

Note in passing that the macro XGOTO available in the assembler documented by 516-14, assembles into the same code in the present assembler.

JST - Hardware interpreted intrasegment jump and store instruction (also used to access system's programs).

Caveats: 1) Pseudo-op DAC is illegal in the segment assembler, so use OCT 0 or equivalent to prefix the referenced code.

ii) Be aware that since the loc +1 of the JST instruction is saved in the first loc. of the referenced code, the code so referenced is not reentrant; so be careful that no I/O is called for from within that code, i.e., do not do anything which will cause a roadblock.

JCALL, JSUBR, JRET - Software interpreted macro instructions for reentrant intrasegment calling (reentrant equivalent of JST type).

JCALL is defined as OPSYN JST, and as such, JST may be used in its place.

JSUBR is the entry point to the code referenced by JCALL (JST) and assembles into:

```
        AB : JSUBR ≡   AB:OCT 0
                              JST   .JSUBR,*
```

The contents of AB are deposited on a push down list. The last entry on the list may be accessed by system defined location .JSTAD; this location may be IRS'ed and LDA'ed indirectly in order to pick arguments.

JRET ≡ JRETRN will return to the location pointed to by .JSTAD.

JRET2, JRET3 will return to the contents of .JSTAD +1 or +2 respectively.

Coding Example			Equivalent Expansion	
JCALL	AB		JST	AB
ARG1			ARG1	
ARG2			ARG2	
1RET			1RET	
2RET			2RET	
.			.	
.			.	
.			.	
AB: JSUBR			AB: OCT	0
			JST	.JSUBR,*
LDA	.JSTAD,*	(get ARG1)	LDA	.JSTAD,*
.			.	
IRS	.JSTAD		IRS	.JSTAD
.			.	
LDA	.JSTAD,*	(get ARG2)	LDA	.JSTAD,*
IRS	.JSTAD		IRS	.JSTAD,*
JRET2		(take 2RET)	JST	.JRET2,*

CALL, RCALL, RET1, RRET - Software interpreted macro instructions for reentrant intersegment calling. CALL and RCALL are essentially the same, CALL has the effect of locking the calling segment in core, while RCALL releases the calling segment from core, thus making room available for other segments. So if you are passing arguments from within the calling sequence or if the call instruction is within a loop, use CALL; otherwise RCALL is the preferred way. Note in passing that the macro GETARG will not handle virtual addresses, only absolute or relocatable. RET1 and RRET are the corresponding return macros, also implemented are: RET2, RET3, RRET2 and RRET3; these are self-explanatory.

```
     Coding Example                  Equivalent Expansion

      CALL  AB                          JST    .CALL.,*
      ADDR  C                          VADDR   AB
      1RET                              ADDR   C
      2RET                              1RET
                                        2RET
        .
        .                                 .
        .                                 .

    C:OCT. 7                          C:OCT    7
      END   SEG                         END    SEG

    - - - - -                         - - - - -

    GETARG  1 (get argument)            JST    .GETA.,*
     STA    OCTAL                       OCT    1
                                        STA    OCTAL
       .
       .                                  .
       .                                  .

     RET2       (take 1RET)             JST    .RET1,*
     END   AB                           END    AB
```