Program Linkage
A Visible/Z Lesson

PROGRAM LINKAGE

The Idea:

The term “program linkage” or simply “linkage” encompasses all the techniques required to make separately assembled control sections work cooperatively. This includes transfers of execution between control sections as well as the communication of data between control sections or programs. To facilitate program linkage, IBM developed a series of rules called “Linkage Conventions” which specify the responsibilities of the cooperating programs. We will call the main program the “calling program” and the subroutine it calls the “called program”. Each type of program has it own conventions.

The Calling Program’s Conventions

1) Register 13 should contain the address of a “Save Area”. The save area is a contiguous collection of 18 fullwords that is used as a storage area for the registers. Since there are only 16 registers which must be shared by all programs, the calling program must provide the address of its save area in register 13. Subsequently, the called program stores the contents of the registers in this area before continuing execution.

2) Register 15 should contain the entry point address of the called program. Since the entry point is not in the calling program, we must use a “virtual” address constant in order to reference the entry point. For example, suppose the entry point in the called program is the CSECT name “SUBPROG”. Then the following instruction would load register 15 properly.

```
         L      R15,=V(SUBPROG)
```

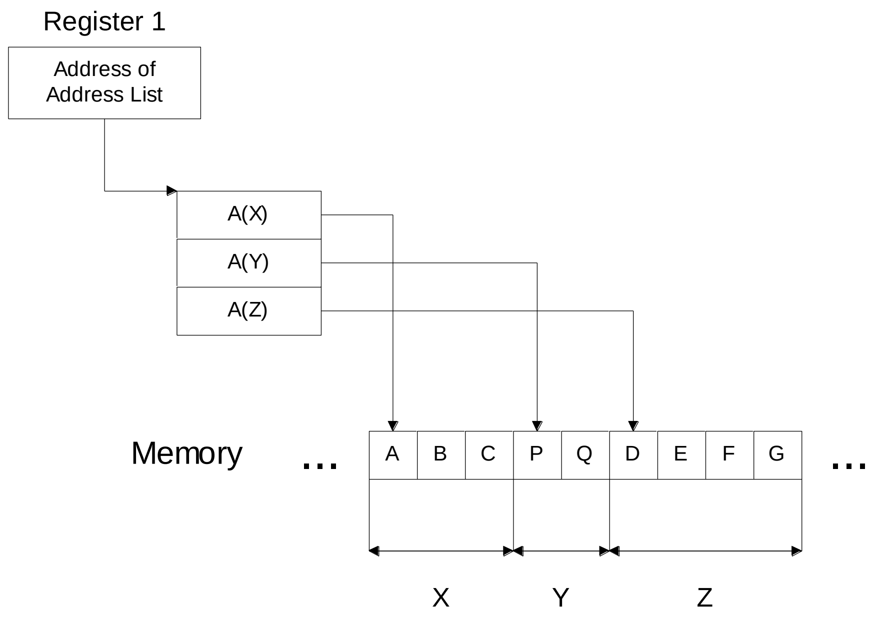
When the assembler processes the virtual address constant, it recognizes that the symbol SUBPROG is not part of the current control section. As a result it assembles a fullword containing zeros for the virtual address. When the linkage editor combines the calling and called subprograms, the fullword is filled in with the correct runtime address of SUBPROG. Only “external” symbols can be used for entry points into a control section. Control section names are automatically external. Other symbols can be declared external using the **EXTRN** directive.

3) When passing variables from the main program to the subprogram, the address of each variable we wish to pass becomes an entry in a list of consecutive fullword addresses. The address of the list is placed in register 1. Assume we want to pass variables X, Y, and Z. The following code will create the address list and initialize register 1.

```
         LA     R1,=A(X,Y,Z)
```

The following diagram illustrates the structure which is created by the above command. We assume the following declarations.

```
X       DC     CL3'ABC'
Y       DC     CL2'PQ'
Z       DC     CL4'DEFG'
```



Notice that only one item of data is passed - the address of the list of addresses. You must use register 1 to find all the variables which are passed.

4) Register 14 should be initialized with the “return” address. This is the address to which the called program will transfer control when it has completed execution. The following instruction will accomplish this.

```
BASR   R14,R15
```

BASR stores the address of the instruction following the **BASR** in register 14, and branches to the address in register 15. In this way, control is transferred to the subprogram.

The Called Program’s Conventions

1) The called program must save the current values of the registers in the caller’s save area. Since the caller has placed the address of the save area in register 13, the following code will save the registers.

```
STM    R14,R12,12(R13)
```

Notice that the explicit address 12(R13) means that R14 is stored in the fourth fullword in the save area (12 bytes off R13). You should also note that all the registers are stored except R13. Register 13 contains the address of the save area and must be stored in an area that belongs in the called program.

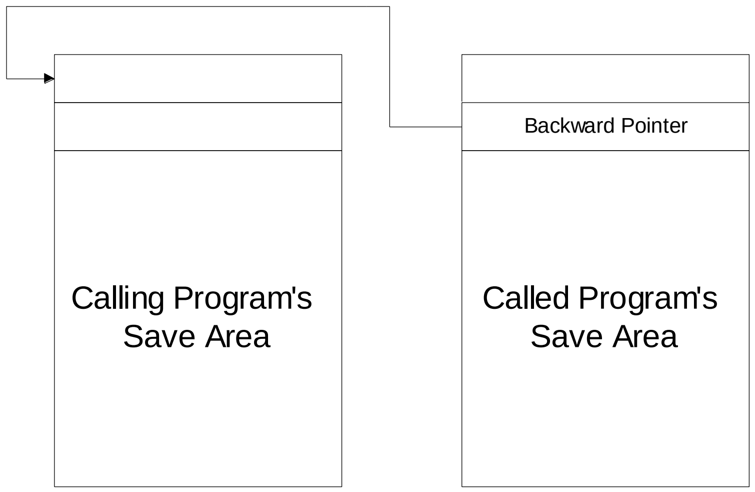
The format of the save area is listed below. We assume a sequence of program calls: Program “A” calls Program “B”, and Program “B” calls Program “C”. The save area in the diagram is for Program “B”.

Used by Pl/I
Program A's Save Area Address
Program C's Save Area Address
Register 14
Register 15
Register 0
Register 1
Register 2
Register 3
Register 4
Register 5
Register 6
Register 7
Register 8
Register 9
Register 10
Register 11
Register 12

2) The contents of register 13 (the address of the caller's save area) must be stored in the second fullword of the called program's save area. This can be done using the following instruction.

```
ST     R13,SAVE+4
```

Notice that "SAVE" refers to the save area in the called program. "SAVE+4" becomes a "backwards pointer" to the previous save area. This is illustrated below.



Additionally, the called program can store the address of its own save area in the calling program's save area in the third word (SAVE+8). This last technique is usually omitted and will not be used in this topic.

3) Like the calling program, R13 should contain the address of the program's save area. This is accomplished with the same instruction used in the main program.

```
LA     R13,SAVE
```

4) Register 1 can be used to access any variables that are passed from the main program. For example, suppose X, Y, and Z are passed as parameters as described in step 3 of the “The Calling Program’s Conventions”. Assume that Y is defined as CL2 in the main program, and YSUB is defined as CL2 in the subprogram. The data in Y can be copied to YSUB as follows.

```
L    R8,4(R0,R1)     LOAD THE 2ND ADDRESS CONSTANT
MVC  YSUB,0(R8)      R8 POINTS AT Y IN THE MAIN
```

The technique of copying a variable to the subprogram and working with the copy is called “Pass by Value”. An alternative would be to work directly with Y in the main program. This technique is called “Pass by Reference”. Either technique is acceptable according to the linkage conventions. “Pass by Value” provides protection for the main program from “side effects” generated by the subprogram.

5) At the end of the subprogram the called program must restore the registers to the values that were stored in the save area. The following code will restore the registers.

```
L      R13,SAVE+4
LM     R14,R12,12(R13)
```

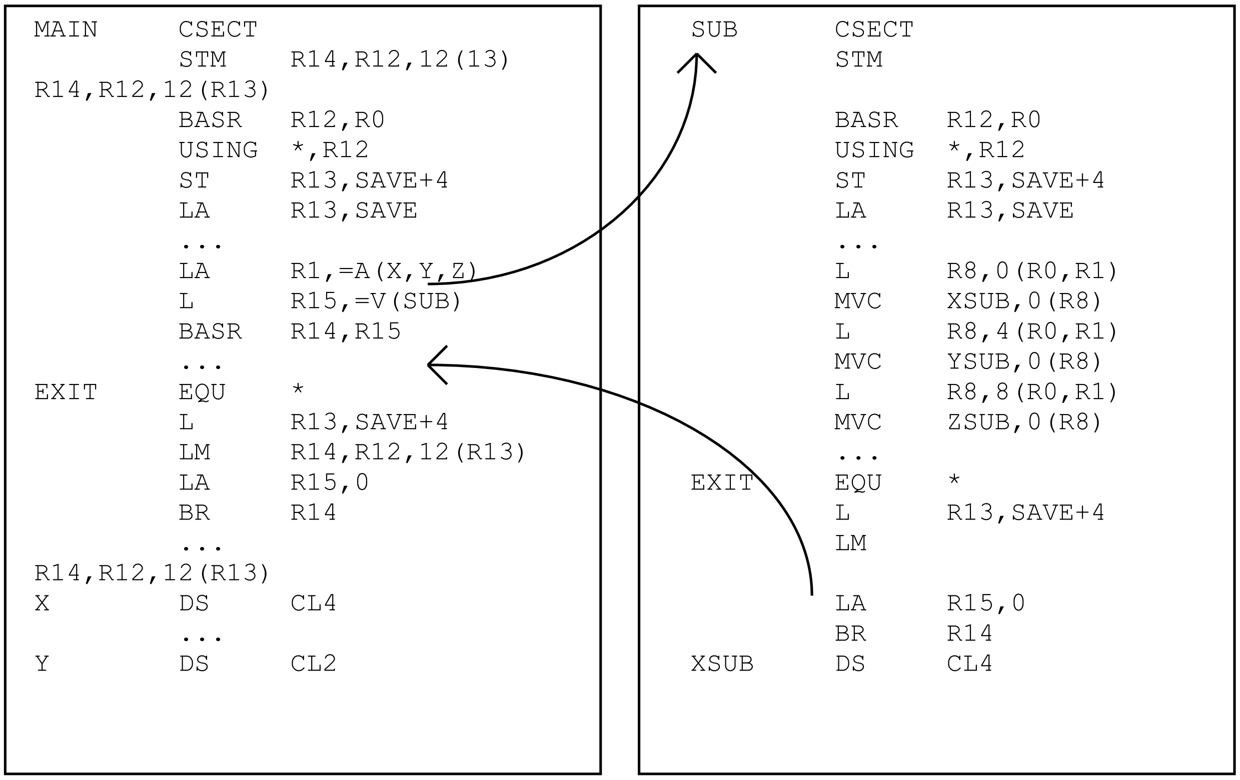
6) The called program should initialize register 15 with a return code value to indicate the results of the subprogram. Generally, return codes are multiples of four - 0, 4, 8, ... where 0 indicates success, 4 is a warning, and all the other codes indicate that errors have occurred. The following command will initialize register 15 with a small (< 4096) return code.

```
LA     R15,8       SET RETURN CODE TO 8
```

7) The subprogram can return to the caller by branching to the return address that was stored in register 14 by the caller.

```
BR      R14
```

The following code illustrates a calling and called program side by side.



```
MAIN     CSECT
         STM    R14,R12,12(13)
R14,R12,12(R13)
         BASR   R12,R0
         USING  *,R12
         ST     R13,SAVE+4
         LA     R13,SAVE
         ...
         LA     R1,=A(X,Y,Z)
         L      R15,=V(SUB)
         BASR   R14,R15
         ...
EXIT     EQU    *
         L      R13,SAVE+4
         LM     R14,R12,12(R13)
         LA     R15,0
         BR     R14
         ...
R14,R12,12(R13)
X        DS     CL4
         ...
Y        DS     CL2
```

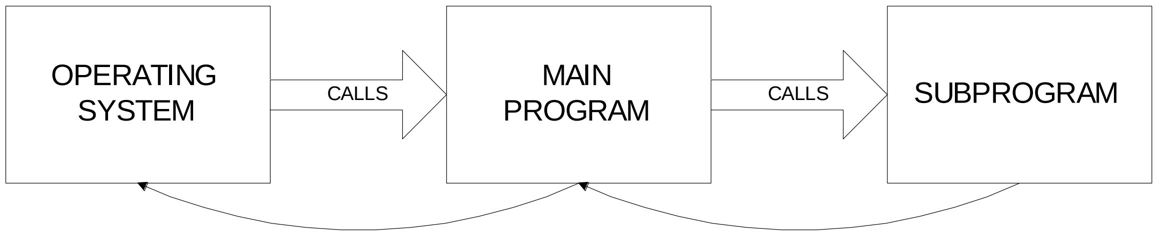
```
SUB      CSECT
         STM

         BASR   R12,R0
         USING  *,R12
         ST     R13,SAVE+4
         LA     R13,SAVE
         ...
         L      R8,0(R0,R1)
         MVC    XSUB,0(R8)
         L      R8,4(R0,R1)
         MVC    YSUB,0(R8)
         L      R8,8(R0,R1)
         MVC    ZSUB,0(R8)
         ...
EXIT     EQU    *
         L      R13,SAVE+4
         LM

         LA     R15,0
         BR     R14
XSUB     DS     CL4
```

```
Z          DS      F                          YSUB      DS       CL2
SAVE       DS     18F                         ZSUB      DS       F
           END     MAIN                       SAVE      DS       18F
                                                        END      SUB
```

Tips

1. Each program you write should follow the calling program's conventions as well as the called program's conventions. The operating system treats each main program you write as a subprogram and so even a main program behaves as a subprogram. On the other hand, each subprogram you write may call other programs either explicitly or by requesting services from the operating system.

Write each program as if it is one link in a chain of program calls.



Trying It Out in VisibleZ:

1) Load the program **linkage.obj** from the \Codes directory and single step through each instruction. At what address are the original contents of the registers stored upon entry to the program?

2) What address is in Register 13 upon entry to the program? Which program set that value?

3) Our save area is filled with X'ff's at program entry. Where are the contents of R13 stored? Why is R13 important?

4) The main body of the program (where work would begin) is represented by two LR instructions: 18 45 18 87. Where is R13 pointing during the time the main body is executing? Why?

5) After executing the main body, R13 is reloaded. What value is put in R13? Why? After the LM instruction, do the registers have the same values they did upon entry to this program?

6) Register 15 is the return code register. What return code is signaled by this program?

7) The BR R14 is the last instruction this program executes. Where does that branch lead us?

8) Load the program **linkage1.obj** from the \Codes directory. This program is used to demonstrate parameter passing, so let's create a couple of parameters. Click on the Parameters tab in VisibleZ and clear the parameter table. Click into the table and type 11 11 11 11 in the first parameter and 22 22 22 22 in the second. Click the enterprise tab.

9) When passing parameters, the convention is to use R1 as a pointer to the table of parameters. What is the address of the table of parameters? Since we passed two parameters, we should

see two addresses in the table. What is the address of the field containing 11 11 11 11? What is the address of the field containing 22 22 22 22?

10) Single step through each instruction and watch as the program loads the two fields into registers, swaps the values, and stores the values in the original fields. Give the parameter table, how many statements are required to get 11 11 11 11 stored in a register?

11) Load the program **linkage2.obj** from the \Codes directory. This program demonstrates how one program can call another program and return. Single step through each instruction. What is the beginning address where the subprograms registers are saved?