Call by Value

/ File main.c /

```
void main() {
  int a = 3;
  int b = 7;
  int c;
  c = sum(a, b);
}
```

gcc -S -m486 main.c

/ File main.s /

```
.file "main.c"
.version "01.01"
gcc2_compiled.:
.text
.align 16
.globl main
.type  main,@function
main:
pushl %ebp
movl %esp,%ebp
subl $12,%esp
movl $3,-4(%ebp)
movl $7,-8(%ebp)
movl -8(%ebp),%eax
pushl %eax
movl -4(%ebp),%eax
pushl %eax
call sum
addl $8,%esp
movl %eax,%eax
movl %eax,-12(%ebp)
.L1:
movl %ebp,%esp
popl %ebp
ret
.Lfe1:
.size  main,.Lfe1-main
.ident "GCC: (GNU) 2.7.2.1"
```

Call by Value Continued

/ File sum.c /

```
int sum(int x, int y) {
  return (x+y);
}
```

gcc -S -m486 sum.c

/ File sum.s /

```
.file "sum.c"
.version "01.01"
gcc2_compiled.:
.text
.align 16
.globl sum
.type  sum,@function
sum:
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%edx
addl 12(%ebp),%edx
movl %edx,%eax
jmp .L1
.align 16
.L1:
movl %ebp,%esp
popl %ebp
ret
.Lfe1:
.size  sum,.Lfe1-sum
.ident "GCC: (GNU) 2.7.2.1"
```

Call by Reference

/ File main.c /

```
void  main() {
  int a = 3;
  int b = 7;
  int c;
  sum(&a, &b, &c);
}
```

gcc -S -m486 main.c

/ File main.s /

```
.file "main.c"
.version "01.01"
gcc2_compiled.:
.text
.align 16
.globl main
.type  main,@function
main:
pushl %ebp
movl %esp,%ebp
subl $12,%esp
movl $3,-4(%ebp)
movl $7,-8(%ebp)
leal -12(%ebp),%eax
pushl %eax
leal -8(%ebp),%eax
pushl %eax
leal -4(%ebp),%eax
pushl %eax
call sum
addl $12,%esp
.L1:
movl %ebp,%esp
popl %ebp
ret
.Lfe1:
.size  main,.Lfe1-main
.ident "GCC: (GNU) 2.7.2.1"
```

Call by Reference Continued

/ File sum.c /

```
void sum(int *x, int *y, int *z) {
  *z = *x + *y;
}
```

gcc -S -m486 sum.c

/ File sum.s /

```
.file "sum.c"
.version "01.01"
gcc2_compiled.:
.text
.align 16
.globl sum
.type  sum,@function
sum:
pushl %ebp
movl %esp,%ebp
pushl %ebx
movl 16(%ebp),%eax
movl 8(%ebp),%edx
movl 12(%ebp),%ecx
movl (%edx),%ebx
addl (%ecx),%ebx
movl %ebx,(%eax)
.L1:
movl -4(%ebp),%ebx
movl %ebp,%esp
popl %ebp
ret
.Lfe1:
.size  sum,.Lfe1-sum
.ident "GCC: (GNU) 2.7.2.1"
```

Call by Value for the MAC-1

/ main.c /
```
void main() { int a = 3;  int b = 7;  int c;  c = sum(a, b); }
```

The assumed conventions for the MAC-1 are that values are passed on the stack to the called function "sum" and that the function returns the value it computes in the accumulator (ac) register. Also it is assumed that the main program has a starting address specified on the assembler's END directive and also that it must initialize the stack pointer (sp) register.

```
  /Main Program illustrating call by value (version 1)
        EXTRN  sum      /Declare label sum to be externally defined
a       3               /Define variables (symbolic addresses) and
b       7               /  their contents
c       RES     1       /Reserve a memory location for the answer
start   loco    4020    /Initialize stack pointer register contents
        swap            /  to 4020 (base 10)
begin   loco    b       /Put address b into ac
        pshi            /Push value in address b ([b] = 7) onto stack
        loco    a       /Put address a into ac
        pshi            /Push value in address a ([a] = 3) onto stack
        call    sum     /Call addition function; result returns in ac
        stod    c       /Put away result in ac into location c
        insp    2       /Clear out stack frame; i.e., reset sp to 4020
        halt            /Stop execution
        END     start   /This main program has starting address start
```

An alternate way of placing the values on the stack directly is shown below:

```
  /Main Program illustrating call by value (version 2)
        EXTRN  sum      /Declare label sum to be externally defined
a       3               /Define variables (symbolic addresses) and
b       7               /  their contents
c       RES     1       /Reserve a memory location for the answer
start   loco    4020    /Initialize stack pointer register contents
        swap            /  to 4020 (base 10)
begin   lodd    b       /Put value in address b ([b] = 7) into ac
        push            /Push it onto the stack
        lodd    a       /Put value in address a ([a] = 3) into ac
        push            /Push it onto the stack
        call    sum     /Call addition function; result returns in ac
        stod    c       /Put away result in ac into location c
        insp    2       /Clear out stack frame; i.e., reset sp to 4020
        halt            /Stop execution
        END     start   /This main program has starting address start
```

Call by Value for the MAC-1 Continued

/ sum.c /

```
int sum(int x, int y) {return (x+y);}
```

Note that the variables x and y are sometimes called “dummy variables” because they refer only to the parameters passed to the function subprogram and not to actual memory locations. x refers to the first parameter and y refers to the second parameter. Parameters are sometimes called arguments. The assumed convention is that the parameters are passed on the stack in the order placed there by the calling program. Upon entry the stack pointer register (sp) points to the memory location containing the return address (i.e., the program counter register contents when the call instruction in the calling program was executed). Thus, the return address is currently in the top of stack location. The second parameter pushed on the stack is underneath it at location sp+1 and the first parameter pushed on the stack is underneath the first parameter at location sp+2. (Recall that the sp register is decremented by a push or call and is incremented by a pop or return). Upon entry to the function subprogram the stack appearss as follows:

```
                  -----------------
sp ---> 4017 |Return address |
                  -----------------
        4018 |      3        |
                  -----------------
        4019 |      7        |
                  -----------------
        4020 |      ?        |
                  -----------------
```

```
  /Function subprogram sum
        ENTRY   sum     /Declare label sum as entry point
x       EQU     1       /Define absolute constant values for
y       EQU     2       /  symbols x and y (not really necessary)
sum     lodl    x       /Pick up first parameter value; ac:=m[sp+1]
        addl    y       /Add to ac second parameter value ac:=ac+m[sp+2]
        retn            /Leave result in ac register and return
        END
```

Assembly, linking, and execution of the above main program and sum function would be carried out by the following sequence of statements (assuming that the main program is in file “prog” and that the function is in file “sub”:

```
tap ee350
assem prog
assem sub
load prog sub
sim prog.abs $EE350/halt $EE350/halt.pascal
tsim prog.abs $EE350/halt $EE350/halt.pascal
```

Call by Reference for the MAC-1

/ main.c /

```
void  main() {int a = 3;  int b = 7;  int c;  sum(&a, &b, &c);}
```

Call by reference passes addresses as parameters to the called procedure. This called procedure is sometimes called a subroutine. FORTRAN distinguishes two types of subprograms: (1) subroutines, and (2) functions. Furthermore, FORTRAN compilers typically use call by reference for parameter passing in both types of subprograms. In C everything is a function; however, those declared to be void do not return a value in some register such as the accumulator register and essentially act like FORTRAN subroutines if call by reference is used.

```
  /Main Program illustrating call by reference
        EXTRN  sum      /Declare label sum to be externally defined
a       3               /Define variables (symbolic addresses) and
b       7               /  their contents
c       RES     1       /Reserve a memory location for the answer
start   loco    4020    /Initialize stack pointer register contents
        swap            /  to 4020 (base 10)
begin   loco    c       /Put address value c into ac
        push            /Push it onto the stack
        loco    b       /Put address value b into ac
        push            /Push it onto the stack
        loco    a       /Put address value a into ac
        push            /Push it onto the stack
        call    sum     /Call addition function; result returns in ac
        insp    3       /Clear out stack frame; i.e., reset sp to 4020
        halt            /Stop execution
        END     start   /This main program has starting address start
```

Call by Reference for the MAC-1 Continued

/ sum.c /

```
void sum(int *x, int *y, int *z) { *z = *x + *y;}
```

Upon entry to subroutine sum the stack (area of memory) appears as follows:

```
                        -----------------
                   4014 |               |
                        -----------------
                   4015 |               |
                        -----------------
           sp ---> 4016 |Return address |
                        -----------------
                   4017 |  Address a    |
                        -----------------
                   4018 |  Address b    |
                        -----------------
                   4019 |  Address c    |
                        -----------------
                   4020 |      ?        |
                        -----------------
```

```
  /Subroutine sum(*x, *y, *z)
        ENTRY   sum     /Declare label sum as entry point
x       EQU     2       /Define absolute constant offset values for
y       EQU     4       /  symbols x and y (not really necessary)
sum     lodl    x       /Pick up first parameter value; address b
        pshi            /Push content of address b onto stack; [b]=7
        lodl    x       /Pick up second parameter value; address a
        pshi            /Push content of address a onto stack; [a]=3
        pop             /Get content of address a into ac
        addl    0       /Add to ac content of address b; note 0 = zero
        stol    0       /Put sum onto stack immediately above ret. addr.
        lodl    y       /Get address c into ac
        popi            /Pop the resulting sum off the stack into location c
        retn            /Return
        END
```