OK, so the main thing you first want to distinguish is a "reference (&) parameter" versus a "regular/value parameter". In your function declaration, you would have the following:

```
function : int blah (int a, int &b) {
     int c;
     int d;
     return 0;
}
```

In the above case, "a" is a regular parameter, and the **value** will be in the corresponding i-register (%i0). "a" is still be treated like a variable (VarSTO). On the other hand, "b" is a reference parameter, and the **address** will be in the corresponding i-register (%i1). "b" will also be treated like a variable (VarSTO), and is even addressable, but the only difference is that when you want to access what is in "b", you need to load the value (or to set a value, you have to store into that address).

Now, the above described the parameters. What about local variables? Local variables will be declared in the function body, so in this case, "c" is a local variable. The local variables will be the ones at a negative offset from the frame pointer (%fp). So, the location for "c" would be at %fp-4. The next local variable, "d" would be at %fp-8. And so on. The space for these local variables is allocated by the **save** instruction. Remember the:

```
-(92 + localvars) & -8
```

In the above example, since we have two local variables, a total of 8 bytes are needed, so the equation would read:

```
-(92 + 8) & -8
```

OK, now that we know where everything is, let's look at a larger example:

```
typedef int MYI;  /* Type Alias */

function : MYI fubar(MYI x, MYI &y, MYI p, MYI &q)
{
    x = x + 1;
    y = y + 1;
    p = p + 1;
    q = q + 1;
    return 0;
}

function : int blah (int a, int &b)
{
    int c;
    int d;
    c = a - 1;
    d = b - 1;
    fubar(a, a, b, b);
    return  0;
}

int z; /* global */
```

```
function : int main()
{
    z = 12;
    blah(17, z);
    return 0;
}
```

So, the assembly will be something like:

```
    ! function fubar
    .section    ".text"
    .align      4
    .global     fubar
fubar:
    set     fubar.SIZE, %g1
    save    %sp, %g1, %sp

    ! x = x + 1
    add     %i0, 1, %i0     ! x = x + 1

    ! y = y + 1
    ld      [%i1], %l0      ! load value of y
    add     %l0, 1, %l0     ! y = y + 1
    st      %l0, [%i1]      ! store value back into y

    ! p = p + 1
    add     %i2, 1, %i2     ! p = p + 1

    ! q = q + 1
    ld      [%i3], %l0      ! load value of q
    add     %l0, 1, %l0     ! q = q + 1
    st      %l0, [%i3]      ! store value back into q

    ! return 0;
    mov     0, %i0

    ret
    restore
fubar.SIZE = -(92 + 0) & -8


    ! function blah
    .section    ".text"
    .align      4
    .global     blah
blah:
    set     blah.SIZE, %g1
    save    %sp, %g1, %sp

    ! c = a - 1
    sub     %i0, 1, %l0     ! a - 1
    st      %l0, [%fp-4]    ! store into c

    ! d = b - 1
    ld      [%i1], %l0      ! load value of b
```

```
    sub     %l0, 1, %l0     ! b - 1
    st      %l0, [%fp-8]    ! store into d

    ! fubar(a, a, b, b)
    mov     %i0, %o0        ! arg0 will be value of a
    st      %i0, [%fp+68]   ! store a into first param memory
    add     %fp, 68, %o1    ! arg1 will be address of a
    ld      [%i1], %o2      ! arg2 will be loaded value of b
    mov     %i1, %o3        ! arg3 will be addr of b (which is already an addr)
    call    fubar           ! Call the function
    nop

    ! return 0;
    mov     0, %i0

    ret
    restore
blah.SIZE = -(92 + 8) & -8


    ! Global variable
    .section    ".bss"
    .align      4
    .skip       4           ! z is at %l7-4
globals:


    ! function main
    .section    ".text"
    .align      4
    .global     main
main:
    set     main.SIZE, %g1
    save    %sp, %g1, %sp

    set     globals, %l7    ! Set global var base ptr

    ! z = 12
    set     12, %l0         ! Value of 12
    st      %l0, [%l7-4]    ! store into z

    ! blah(17, z)
    set     17, %o0         ! value for arg0
    add     %l7, -4, %o1    ! arg1 is address of z
    call    blah
    nop

    ! return 0;
    mov     0, %i0

    ret
    restore
main.SIZE = -(92 + 0) & -8
```

So, as you can see, when you pass something as a "reference parameter", you need to have the actual address in the out register. When you are passing by value, then the value is in the out register. Particularly pay attention to the code

in function "blah".

You can have 8 combinations:
1) a value parameter being sent to a subsequent function as a value argument.
2) a value parameter being sent to a subsequent function as a reference argument (you need to store the value into a param location (e.g. %fp+68, %fp+72, ...) and pass the address)
3) a reference parameter being sent to a subsequent function as a value argument (you have to load from the address to get the value you want to pass)
4) a reference parameter being sent to a subsequent function as a reference argument (you just copy the address to the out register).
5) a local variable being sent to a subsequent function as a value argument (you load from %fp-4 and send the value).
6) a local variable being sent to a subsequent function as a reference argument (you send the actual address of %fp - 4 to the function).
7) a global variable being sent to a subsequent function as a value argument (you load from %l7-4 and send the value).
8) a global variable being sent to a subsequent function as a reference argument (you send the actual address of %l7 - 4 to the function).

Disclaimer: If you implemented global variables using another method (ie, without the %l7), then make the according changes to the description above.