Explanation:

Before going into caller and callee saved register first we see, What is need of saving registers process.

Saving Registers during calls

-> A procedure often needs many registers for local variables, copies of parameters, and temporary calculations. In a load/store architecture, the procedure needs registers in order to access values stored on the stack. A newly invoked procedure therefore must immediately have some registers made available to it. Therefore any register that is part of the environment of the calling program cannot be used by a newly called procedure unless its value can first be preserved and then be restored upon return to the calling program.

1. Registers are typically partitioned into:

**Callee saved registers**

whose values the called method must save and restore (if it uses them).

**Caller saved registers**

whose value the calling method must save and restore (if it depends on them after the call).

It is possible (and not uncommon) to put all registers in one group or the other (callee-saved is the favorite).

1. Callee saving has the advantage of keeping the total size of your code small (each method only contains one set of instructions to save registers).

Note: You have to generate the register saving instructions before you know what registers need to be saved. The fact that you are generating assembly code lets you leave this problem to the assembler by using a symbolic name for the mask that determines what registers need to be saved.

1. Caller saving has the advantage that you only save the registers in use at a particular call rather than all registers ever used in the procedure (although you may end up saving registers that aren't altered by the called procedure).

====================================================================

Why all register all register are not taing as same type like all caller saved or callee saved ?

1. A register allocator can be designed to take advantage of these two classes of registers:
   * Use caller saved registers to hold values whose usefulness begins and ends in a section of code including no calls (entire body of any "leaf" procedure.
   * Use callee saved registers for values produced before and needed after a call.
2. Where are register values saved?
   * In the caller's frame -- for caller saved registers.
   * In the called method's frame -- for callee saved.

Caller and callee contain

“Caller‐saved” registers ♣ It is the caller’s responsibility to save any important data in  these registers before calling another procedure (i.e. the  callee can freely change data in these registers)  ♣ Caller saves values in its stack frame before calling Callee,  then restores values after the call ϖ

“Callee‐saved” registers ♣ It is the callee’s responsibility to save any data in these  registers before using the registers (i.e. the caller assumes  the data will be the same across the callee procedure call) ♣ Callee saves values in its stack frame before using, then  restores them before returning to caller

int fn(int a,int b,int c,int d,int e,int f,int g,int h,int i,int j,int k)

{

return a+b+c+d+e+f+g+h+i+j+k;

}

int main()

{

int a=10,b=20,c=30,d=40,e=50,f=60,g=70,h=80,i=90,j=a,k=b;

printf("%d %d %d %d\n",a,b,j,k);

int r=fn(a,b,c,d,e,f,g,h,i,j,k);

printf("%d %d %d %d\n",a,b,j,k);

}

extern foo(int,int,int,int,int,int,int,int,int,int,int);

//extern fun(int,int,int,int,int,int,int,int,int,int,int);

int fn(int a,int b,int c,int d,int e,int f,int g,int h,int i,int j,int k)

{

int r=a+b+c+d+e+f+g+h+i+j+k;

foo(a,b,c,d,e,f,g,h,i,j,k);

fun(g,h,i,j,k,a,b,c,d,e,f);

return r;

}

Reference:

-<http://pages.cs.wisc.edu/~goadl/cs354/handouts/ch9.pdf>

-<http://www.cs.williams.edu/~tom/courses/434/outlines/lect18_2.html>

-<https://courses.cs.washington.edu/courses/cse410/17wi/lectures/CSE410-L13-procedures-II_17wi.pdf>

**Registers are typically partitioned into:into**

**1.Examples**

**1)**

* int fn(int a,int b,int c,int d,int e,int f,int g,int h,int i,int j,int k)
* {
* return a+b+c+d+e+f+g+h+i+j+k;
* }
* int main()
* {
* int a=10,b=20,c=30,d=40,e=50,f=60,g=70,h=80,i=90,j=a,k=b;
* printf("%d %d %d %d\n",a,b,j,k);
* int r=fn(a,b,c,d,e,f,g,h,i,j,k);
* printf("%d %d %d %d\n",a,b,j,k);
* }

2)

* extern foo(int,int,int,int,int,int,int,int,int,int,int);
* //extern fun(int,int,int,int,int,int,int,int,int,int,int);
* int fn(int a,int b,int c,int d,int e,int f,int g,int h,int i,int j,int k)
* {
* int r=a+b+c+d+e+f+g+h+i+j+k;
* foo(a,b,c,d,e,f,g,h,i,j,k);
* fun(g,h,i,j,k,a,b,c,d,e,f);
* return r;
* }

3)

* extern foo(int,int,int,int,int,int,int,int,int,int,int);
* extern fun(int,int,int,int,int,int,int,int,int,int,int);
* void fn(int a,int b,int c,int d,int e,int f,int g,int h,int i,int j,int k,int l,int m,int n)
* {
* int a1, b1, c1, d1, e1, f1, g1, h1, i1, j1, k1, l1,m1, n1;
* foo(a,b,c,d,e,f,g,h,i,j,k);
* fun(a1=a,b1=b,c1=c,d1=d,e1=e,f1=f,g1=g,h1=h,i1=i,j1=j,k1=k);
* }

4)

void fn(int a,int b,int c,int d,int e,int f,int g)

{

display(a+b+c+d+e+f+g);

try(a,b,c,d,e,f,g);

}