ABI Register and Stack Usage Convention

Register Usage

Register	Type	Use
R0	Volatile	Link register
R1	Dedicated	Stack pointer
R2	Dedicated	Read-only small data area anchor
R3 – R4	Volatile	Parameter Passing, return values
R5 – R10	Volatile	Parameter passing
R11 – R12	Volatile	General use
R13	Dedicated	Read-write small data area anchor
R14 – R31	Non volatile	
F0 – F31		Floating point is not available on the MPC823
CR2 – CR4	Nonvolatile	Condition register
CR fields except CR2-CR4	Volatile	Condition register

- Use dedicated registers for their intended purpose only.
- Use R3 – R10 for passing values and R3 and R4 for returning values across function calls beginning with R3.
- If you need a temporary register, use one of the volatile registers. Begin with R12 and work your way down.
- Volatile registers **are not** preserved across function calls. If your function uses volatile registers, their contents are not necessarily preserved if you call an ABI compliant function.
- If you need a variable that is preserved across function calls, use one of the nonvolatile registers. Begin with R31 and work your way down.
- Nonvolatile registers **are** preserved across function calls. You can call an ABI compliant function with the confidence that nonvolatile register contents will be preserved on return.
- If you use a nonvolatile register in a function, you are obligated to preserve its contents.
- Most of the condition register fields are volatile. Be careful to preserve the condition register if a conditional test and corresponding conditional branch are separated by a function call.

Stack Usage

- Use R0 for the link register.
- Use mflr to read the link register.
- Use mtlr to write the link register.
- The link register is always stored in the second to the last location of the **previous** stack frame.
- Use R1 for the stack pointer.
- The stack pointer to the previous frame (back chain) is stored in the last location of the **current** stack frame.
- Use stwu to store the stack pointer on the stack, update the stack pointer and allocate stack space. For example, consider stwu r1, -12 (r1) assuming r1 contains the current stack pointer.
    1. The value of r1 is stored in the location specified by r1 –12 bytes. Thus, the stack pointer is stored on the bottom of the stack.
    2. Because of the – 12 offset, 12 bytes or 3 words are allocated in the frame.
    3. The location r1 – 12 bytes is stored in r1. Thus, r1 now points to the bottom of the current stack frame.

Stack allocation for stwu r1, -12(r1) where r1 = 0x0011000.

Address in stack	contents
0x001100c	
0x0011008	Available for general use
0x0011004	Reserved for link register
0x0011000	00x001100c

Example

The following C program is expressed in assembly illustrating ABI compliant stack usage and function calls.

C code

```
int dox();
int doy();


void main() {
int x;
        x = 0;
        while(1) x = do1(x) + 1;
}
int do1(int x) {
        x = x + 1;
        x = do2()+ x;
        return(x);
}
int do2() {
        abi compliant function that returns a number between 0 and 100;
        return(int x);
}
```

Assembler

#the following is the main()

```
        .data                       #define data section (location 0x11000)
        .align 2
        .skip 4*100                 #allocate 100 words of space for stack
stack:  .skip 4                     #set stack label to top of stack

        .text                       #define text or program
                                    #section (location 0x10000)
        .align 2
        .global _start

_start: lis     r1,stack@h          # load stack pointer to r1
        ori     r1,r1,stack@l
        stwu    r1, -0x8(r1)        #store stack pointer to bottom of frame
                                    # allocate stack space
                                    # set stack pointer r1
                                    # to point to bottom of frame

        addi    r31,r0,0            # set r31 to zero   <---- 1
loop:   ori     r3,r31,0            # put r31 into r3
        bl      do1                 # branch to do1
                                    # argument in r3
                                    # return address in link register
        addi    r31,r3,1            # increment returned value
        b       loop                # loop and continue
```

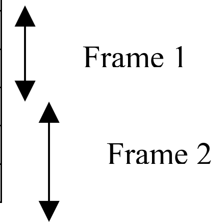
```
                # the following is the d01 function

do1:    mflr    r0                  # get LR
        stw     r0,4(r1)            # save LR in previous frame
        stwu    r1,-0xC(r1)         # stack management as above
                                    # allocate space for: stack pointer,
                                    #link register and one non volatile reg
        stw     r31,8(r1)           # save nonvolatile register r31

        addi    r3,r3,1             # increment  <------------------- [2]
        ori     r31,r3,0            # save argument r3 into r31
        bl      do2                 # branch to do2
                                    # do2 is a abi compliant func
        add     r3,r31,r3           # add return value with argument

        lwz     r5,8(r1)            # restore nonvolatile register r31
        lwz     r0,0x10(r1)         # get LR from previous frame
        mtlr    r0                  # restore LR
        addi    r1,r1,0xC           # restore SP
        blr                         # return to call point with argument
                                    # in r3

                # the following is the do2 function

 do2:
      abi compliant function that returns a number between 0 and 100.
      The details of register usage are not known, but usage is ABI
      compliant.

      Position 3 is after stack allocation.  <------------------- [3]

      return(r3)
```

Stack contents at position 1

address	contents
0x0011068	unknown
0x0011064 contents of r1	0x0011068

Stack contents at position 2

address	contents
0x0011068	LR to main just after do1 call
0x0011064	0x0011068
0x0011060	r31
0x001105c	unknown
0x0011058 contents of r1	0x0011064



Stack contents at position 3

address	contents
0x0011068	LR to main just after do2 call
0x0011064	0x0011068
0x0011060	r31
0x001105c	unknown
0x0011058	0x0011064
0x0011054	
0x0011050	
0x001104c	
0x0011048	

Depends on frame size	0x0011058