PowerPC EABI (Embedded Applications Binary Interface) study report

Document id : **PowerPC-EABI-000**
Document Name : PowerPC EABI

Contents

1. Introduction

This section briefs the purpose of this document, section 2 briefs the functions accounted in EABI, section 3 details stack frame organization in M68000 using MRI tool suite, section 4 states stack frame organization in any PowerPC using GCC tool suite, section 5 signifies the stack pointer initialization in PowerPC for any board/card, section 6 shall cover about the limitations of this document, and the rest of the sections contain appendix and document history.

1.1) Scope

This document will be used by the system architects and system software developers(especially board initialization, POST, device driver, interrupts/exceptions, linux internal system software modules) to conceptualize, specify, design and implement various elements in the embedded operating system architecture.

1.2) References

1. PowerPC Embedded Processors Application Note:
   Developing PowerPC Embedded Application Binary Interface(EABI) compliant programs, Version: 1.0
   IBM Microelectronics
2. PowerPC EABI: 32-bit implementation, Version: 1.0
   Microcontroller Technologies Group, Motorola
3. IBM PowerPC 405, Embedded Processor Core
   User's manual
4. Motorola PowerPC Microprocessor Family:
   The Programming Environments for 32-bit microprocessors
5. Microtec Research Inc.: MCC68k User's Guide

6. a sample C program for explaining MRI M68k ABI and PPC EABI

```
struct s {
int     as;
int     bs;
}s1, *s2, *s3;

enter_here() {

int a = 100, b = 50, *pa;

int func(int, int);
int func1(int, int);
int func2(struct s, int);
int* func3(struct s, int);
int func4(struct s*, int);
struct s* func5(struct s*, int);

               /* observe assemlby listing of pushing a and b here ...  */
a = func(a,b);  /* func returns fixed-point data, uses D0               */

s1.as = a;
s1.bs = b;

a = func2(s1, b); /* passing structure by value */

pa = func3(s1, b); /* func returning int*, uses D0 */

s2 = &s1;
a = func4(s2, b); /* passing structure by reference/pointer */

s3 = func5(s2, b); /* func returns struct pointer/reference uses D0 */

}

int func(int a, int b) {

int c, d, e;

c=a; d=b;

e = func1(c, d);

return(c+d);

}

int func1(int a, int b) {

int c, d;

c=a; d=b;

return(c+d);

}

int func2(struct s a, int b) {

int c, d;

c=a.as; d=b;

return(c+d);
}

int* func3(struct s a, int b) {

int c, d;

c=a.as; d=b;

return(&a);
}

int func4(struct s *a, int b) {

int c, d;
```

```
c=a->as; d=a->bs;

return(c+d);

}


struct s* func5(struct s *a, int b) {

int c, d;

c=a->as; d=a->bs;

return(a);
}
```

2. EABI - overview

2.0) Introduction

ABI serves as an interface for the compiled application programs to the system software. Generally, any processor architecture comes with the ABI which would be used by tools vendor for the best understood methodologies for the kinds of object file format support, register usage conventions, data types which map the high level basic data types to processor specific access, data types alignments(any byte align(char), 2-bytes align(integer/short), etc) for the best use of processor performance, stack frame generation, its use and destruction, function parameters passing (i.e.) the order by which the actual arguments to be passed into stack by calling functions and function return values.
And some ABI defines data access(which may belong to .data/.bss sections) using smaller displacement fields which can better fit in assembly instructions itself, by a specific register which would be loaded with the base address of the section desired.
To name a few ABIs, Unix System V ABI, IBM PowerPC EABI, Microtec M68000 ABI, GCC M68000 ABI, GCC 80x86 ABI and so on.
The more the same definition of ABI from multiple vendors, the more the portability and mixing up of code between different tool sets.
IBM PowerPC EABI is derived from Unix System V ABI with the goal of reducing memory usage and optimizing execution speed, as these are prime requirements of embedded system software.

3. Understanding M68000 ABI using MRI tool suite

Motorola M68000 is a CISC GPR architecture and it has got 8 Data registers, referred as D0 to D7, and 8 Address registers, referred as A0 to A7 in fixed point arithmetic architecture. Each register is of 32-bit wide, despite its address bus width of 24-bit and data bus width of 16-bit. So care is to be applied in visualizing contents of address register. In fact, it is called to be pseudo 32-bit processor, signifying 32-bit fixed point ALU.
MRI defines ABI for M68000 assembly programming as follows:

Register/function desired	Usage
A7 register	Stack pointer, user and superuser
A6 register	Stack frame pointer
D0 register	1. function returning 32-bit fixed point arithmetic data, 2. function returning pointer/reference 32-bit(actually 24-bit address))
function parameters passing	push the actual values/references from right to left

3.0) Stack frame

In function calling, the calling function should push the actual arguments into stack before making actual call to the function. Called function has to generate stack frame which would be based on number and data types of the local variables used in that function. So every

function has to create a stack frame and destruct before the exit of function. The portion of code which is used for creating stack frame would be called as prologue and the portion of code used for destructing stack frame is called as epilogue. The prologue and epilogue would be appearing in assembly listing. Normally a stack appears like this once PC set into called the function after stack frame creation,

(low address in memory) .....
non-volatile registers(which used in function)
local variables(local to the function invoked)
stack frame pointer's previous data(used for destruction of stack)
machine status register(used in exceptions redirection and normal function invoke need not have this info)
(high address in memory....) return address(to where to continue exection after exiting from this function)

In M68000, 2 mnemonics such as LINK and UNLK used to facilitate the stack frame creation and destruction for prologue and epilogue code portion respectively.

the syntax of LINK is **LINK A6, #-N** which does

1. SP <- SP-4
2. (SP) <- A6
3. A6 <- SP
4. SP <- SP-N

the syntax of UNLK is **UNLK A6** which does

1. SP <- A6
2. A6 <- (SP)
3. SP <- SP+4

3.1) M68000 assembly listing

To better understand the implementation, follow the M68000 assembly listing mixed with commented C listing,

(to have a see on the C listing only)

M68000 assembly listing with commented C listing

To Generate assembly listing with comment c lines using MRI tool suite

```
root@zest~/zest/PowerPC:-) mcc68k -Fsm -S file.c
```

```
;       Microtec(R) Research Inc. 68K C Compiler 4.5ZC
;       Host Operating System - SUN4 UNIX
;       Command Line Options Specified:
;               -S -Ri/tmp/111110274.1 -o 1111.s -Fsm 1111.c

        TTL     1111.c
;       MCC68K 4.5T/4.5ZC s S4U051795 s408-May-97.14:47:20
        OPT     NOABSPCADD,E,NOPCR,P=68000,CASE
        NAME    1111
        XCOM _s1,8
        XCOM _s2,4
        XCOM _s3,4
        SECTION code,,C
        XDEF    _enter_here
_enter_here:
;*** allocation of local variables ***
;       a = d3
;       b = d2
;       pa = a6-4
;
;struct s {
;int    as;
;int    bs;
;}s1, *s2, *s3;
;
;enter_here() {
        link    a6,#-4
        movem.l d2/d3,-(sp)
;
```

```
;int a = 100, b = 50, *pa;
        moveq   #100,d3         // a is stored in d3 register
        moveq   #50,d2          // b is stored in d2 register
;
;int func(int, int);
;int func1(int, int);
;int func2(struct s, int);
;int* func3(struct s, int);
;int func4(struct s*, int);
;struct s* func5(struct s*, int);
;
;               /* observe assemlby listing of pushing a and b here ...  */
;a = func(a,b);          /* func returns fixed-point data, uses D0              */
        move.l  d2,-(sp)   // push b value first as per C calling convention
        move.l  d3,-(sp)   // push a value as per C calling convention
        jsr     _func
        move.l  d0,d3      // return value in d0, restored to d3 register
;
;s1.as = a;
        move.l  d3,_s1
;s1.bs = b;
        move.l  d2,_s1+4
;
;a = func2(s1, b); /* passing structure by value */
        move.l  d2,-(sp)
        movea.l #_s1+8,a0
        move.l  -(a0),-(sp)
        move.l  -(a0),-(sp)
        jsr     _func2
        move.l  d0,d3
;
;pa = func3(s1, b); /* func returning int*, uses D0 */
        move.l  d2,-(sp)
        movea.l #_s1+8,a0
        move.l  -(a0),-(sp)
        move.l  -(a0),-(sp)
        jsr     _func3
        move.l  d0,-4(a6)
;
;s2 = &s1;
        move.l  #_s1,_s2
;a = func4(s2, b); /* passing structure by reference/pointer */
        move.l  d2,-(sp)
        move.l  _s2,-(sp)
        jsr     _func4
        move.l  d0,d3
;
;s3 = func5(s2, b); /* func returns struct pointer/reference uses D0 */
        move.l  d2,-(sp)
        move.l  _s2,-(sp)
        jsr     _func5
        move.l  d0,_s3
;
;}
        movem.l -12(a6),d2/d3
        unlk    a6
        rts
;        code: 134 bytes   stack: 4 bytes
        XDEF    _func
_func:
;*** allocation of local variables ***
;       a = a6+8
;       b = a6+12
;       c = d3
;       d = d2
;       e = a6-4
;
;int func(int a, int b) {
        link    a6,#-4          // stack frame(4 bytes) creation
        movem.l d2/d3,-(sp)
;
;int c, d, e;
;
;c=a; d=b;
        move.l  8(a6),d3
        move.l  12(a6),d2
;
;e = func1(c, d);
        move.l  d2,-(sp)
        move.l  d3,-(sp)
```

```
        jsr     _func1
        move.l  d0,-4(a6)
;
;return(c+d);
        move.l  d3,d0
        add.l   d2,d0           // fixed-point return value in d0
;
;}
        movem.l -12(a6),d2/d3
        unlk    a6              // stack frame(4bytes) destruction
        rts
;        code: 44 bytes   stack: 4 bytes
        XDEF    _func1
_func1:
;*** allocation of local variables ***
;       a = sp+8
;       b = sp+12
;       c = d1
;       d = d2
;
;int func1(int a, int b) {
        move.l  d2,-(sp)
;
;int c, d;
;
;c=a; d=b;
        move.l  8(sp),d1
        move.l  12(sp),d2
;
;return(c+d);
        move.l  d1,d0
        add.l   d2,d0
;
;}
        move.l  (sp)+,d2
        rts
;        code: 18 bytes   stack: 0 bytes
        XDEF    _func2
_func2:
;*** allocation of local variables ***
;       a = a6+8
;       b = a6+16
;       c = d1
;       d = d2
;
;int func2(struct s a, int b) {
        link    a6,#0
        move.l  d2,-(sp)
;
;int c, d;
;
;c=a.as; d=b;
        move.l  8(a6),d1
        move.l  16(a6),d2
;
;return(c+d);
        move.l  d1,d0
        add.l   d2,d0
;}
        move.l  (sp)+,d2
        unlk    a6
        rts
;        code: 24 bytes   stack: 0 bytes
        XDEF    _func3
_func3:
;*** allocation of local variables ***
;       a = a6+8
;       b = a6+16
;       c = a6-4
;       d = a6-8
;
;int* func3(struct s a, int b) {
        link    a6,#-8
;
;int c, d;
;
;c=a.as; d=b;
        move.l  8(a6),-4(a6)
        move.l  16(a6),-8(a6)
;
```

```
;return(&a);
        lea.l   8(a6),a0
        move.l  a0,d0   // struct pointer/reference getting passed thro d0 reg

;}
        unlk    a6
        rts
;        code: 26 bytes   stack: 8 bytes
        XDEF    _func4
_func4:
;*** allocation of local variables ***
;       a = a0
;       b =
;       c = d1
;       d = d2
;
;int func4(struct s *a, int b) {
        move.l  d2,-(sp)
        movea.l 8(sp),a0
;
;int c, d;
;
;c=a->as; d=a->bs;
        move.l  (a0),d1
        move.l  4(a0),d2
;
;return(c+d);
        move.l  d1,d0
        add.l   d2,d0
;
;}
        move.l  (sp)+,d2
        rts
;        code: 20 bytes   stack: 0 bytes
        XDEF    _func5
_func5:
;*** allocation of local variables ***
;       a = a0
;       b =
;       c = a6-4
;       d = a6-8
;
;
;struct s* func5(struct s *a, int b) {
        link    a6,#-8
        movea.l 8(a6),a0
;
;int c, d;
;
;c=a->as; d=a->bs;
        move.l  (a0),-4(a6)
        move.l  4(a0),-8(a6)
;
;return(a);
        move.l  a0,d0   // struct pointer/reference getting passed thro d0 reg
;}
        unlk    a6
        rts
;        code: 24 bytes   stack: 8 bytes
        END
; code: 290 bytes
```

4. PowerPC EABI

POWER Architecture, a second generation reduced instruction set computer (RISC) Load-Store architecture. Here the POWER apparently gets unshrunk to Performance Optimized with Enhanced RISC. Refer the documents related to PowerPC listed under reference section for the complete ABI details. PowerPC architecture of 32-bit is referred here for the listing.
More importantly, there are no stack operation codes defined in PowerPC architecture, (i.e.) no **PUSH and POP** opcodes provisioned. Complete stacking and unstacking to be done manually by dedicating one of the GPRs(defined according to the EABI).
User and superuser stack frame handling should be done using single stack pointer. It may appear that superuser program space(exceptions handler) would run on user space using single stack pointer but the machine status has been kept by the processor intact, user or superuser

indicated in MSR.
And stack frame pointer(dedicated register) is Register-31(Fixed point register, r31) under GCC.2.95.2 cross tools for powerpc-eabi. Stack frame in powerPC comes with **Link Register Save word and Back Chain Word** , in which Back chain word holds the key of destructing stack frame by having the top of stack contains address of the previous stack frame.
So prologue listing part of function should store the current SP at the top of the stack after calculating the desired stack size with updation to the SP. Therefore, the highest(?) address of stack frame would contain Link Register Save Word and the lowest address of stack frame would contain Back Chain Word for this stack frame. Trivially, the space between LR Save and Back Chain Word contains data space for local variables, return EA of the function called by function related to the particular stack frame we talk of (!), and stack frame pointer's previous content. Epilogue listing part of function needs to just copy the contents top of the stack to SP and restore LR Save Word from stack at the offset of current stack pointer+4(important, in case of 32-bit addr) to LR prior calling return(in PowerPC, it is one of the special branches using LR as target address).

In PowerPC architecture, **STWU(Store Word with Update) and LWZ(Load Word and Zero the unupdated Most Significant Bits)** (normal load and store opcodes) used to facilitate the stack frame creation and destruction for prologue and epilogue code portion respectively.

the syntax of STWU is **STWU rS, d(rA)** which does
rS - Source register(data/address), d - displacement, rA - Target register(address)

1. EA <- (rA) + EXTS(d); EXTS(d) - sign extended d field
2. MEM(EA, 4) <- rS
3. rA <- EA

4.0) Stack frame

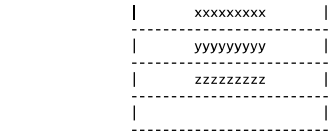
Follow the flow diagram for each of the function calls,

```
(instruction
pointer)

-------------------------------------------------------------------------
(assume stack points to 200 000)
_entry:
2000:           .
                .
                .
                .
                .
2300:           bl _f1 (branch to f1 and store 2304 in LR)
2304:           .
                .
                .
                blr(return)
-------------------------------------------------------------------
_f1:
4000:           (stack frame formation)
                (store return addr @SP(updated)+d+4, link reg save word)
                .
4020:           bl _f2 (branch to f2 and store 4024 in LR)
4024:
                .
                restore LR
                blr(return)
-------------------------------------------------------------------
_f2:
5000:           (stack frame formation)
5004:           (store return addr @SP(updated)+d+4, link reg save word)
                .
                .
                restore LR
                blr(return)
-------------------------------------------------------------------------
```

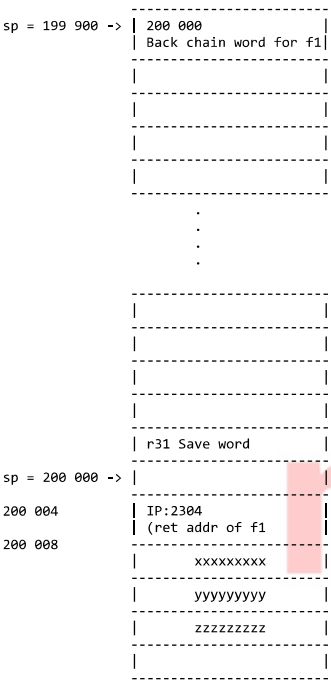
stack appears at IP: 2000

```
                -------------------------
sp = 200 000 -> |                       |
                -------------------------
200 004         |(very important,       | used by next func call
                |leave it unused)       | for LR save word
200 008         -------------------------
```



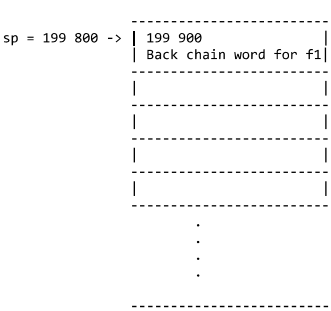
stack appears after IP: 4000 but prior returning from _f1

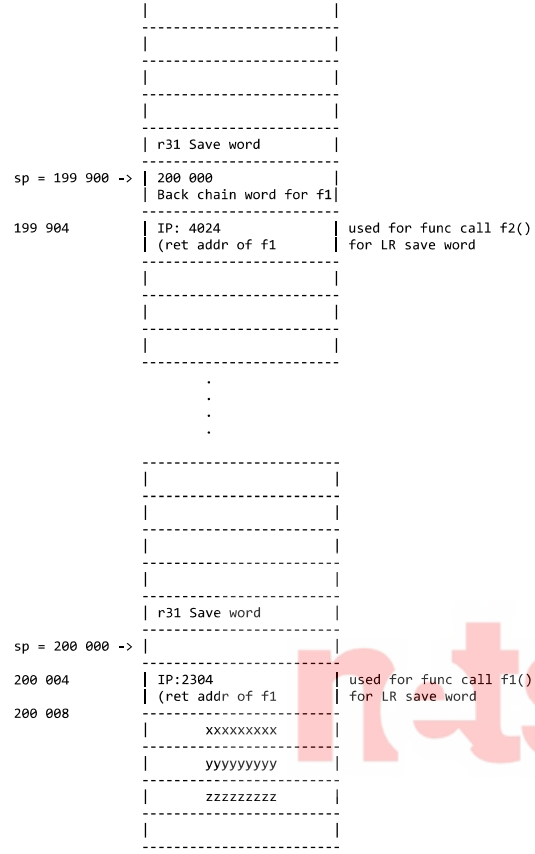
for better understanding, assume, stack frame is of size 100 bytes for the function f1()



stack appears after IP: 5000 but prior returning from _f2

for better understanding, assume, stack frame is of size 100 bytes for the function f2()





4.1) PowerPC assembly listing

To better understand the implementation, follow the C and assembly listing mixed with commented C listing,

(to have a see on the C listing only)

PowerPC EABI assembly listing with commented C listing

To Generate assembly listing with comment c lines using powerpc-eabi-gcc

```
root@zest~/zest/PowerPC:-) powerpc-eabi-gcc -v
        Reading specs from /usr/local/powerpc-eabi/lib/gcc-lib/powerpc-eabi/2.95.2/specsgcc version 2.95.2 19991024 (release)
root@zest~/zest/PowerPC:-)  powerpc-eabi-gcc -g -Wa,-a -c file.c > file.s
```

stripped version of actual assembly file

```
"GCC: (GNU) 2.95.2 19991024 (release)"


  29                            .globl enter_here
  30                            .type    enter_here,@function
  31                    enter_here:
   1:1111.c        ****
```

```
  2:1111.c        **** struct s {
  3:1111.c        **** int     as;
  4:1111.c        **** int     bs;
  5:1111.c        **** }s1, *s2, *s3;
  6:1111.c        ****
  7:1111.c        **** enter_here() {
32                      .stabn 68,0,7,.LM1-enter_here
33                      .LM1:
34 0000 9421FFD8                stwu 1,-40(1)
35 0004 7C0802A6                mflr 0
36 0008 93E10024                stw 31,36(1)
37 000c 9001002C                stw 0,44(1)
38 0010 7C3F0B78                mr 31,1
  8:1111.c        ****
  9:1111.c        **** int a = 100, b = 50, *pa;
39                      .stabn 68,0,9,.LM2-enter_here
40                      .LM2:
41                      .LBB2:
42 0014 38000064                li 0,100
43 0018 901F0008                stw 0,8(31)
44 001c 38000032                li 0,50
45 0020 901F000C                stw 0,12(31)
10:1111.c        ****
11:1111.c        **** int func(int, int);
12:1111.c        **** int func1(int, int);
13:1111.c        **** int func2(struct s, int);
14:1111.c        **** int* func3(struct s, int);
15:1111.c        **** int func4(struct s*, int);
16:1111.c        **** struct s* func5(struct s*, int);
17:1111.c        ****
18:1111.c        ****                 /* observe assemlby listing of pushing a and b here ...  */
19:1111.c        **** a = func(a,b);  /* func returns fixed-point data, uses D0               */
46                      .stabn 68,0,19,.LM3-enter_here
47                      .LM3:
48 0024 807F0008                lwz 3,8(31)
49 0028 809F000C                lwz 4,12(31)
50 002c 48000001                bl func
51 0030 7C601B78                mr 0,3
52 0034 901F0008                stw 0,8(31)
20:1111.c        ****
21:1111.c        **** s1.as = a;
53                      .stabn 68,0,21,.LM4-enter_here
54                      .LM4:
55 0038 3D600000                lis 11,s1@ha
56 003c 392B0000                la 9,s1@l(11)
57 0040 801F0008                lwz 0,8(31)
58 0044 90090000                stw 0,0(9)
22:1111.c        **** s1.bs = b;
59                      .stabn 68,0,22,.LM5-enter_here
60                      .LM5:
61 0048 3D600000                lis 11,s1@ha
62 004c 392B0000                la 9,s1@l(11)
63 0050 801F000C                lwz 0,12(31)
64 0054 90090004                stw 0,4(9)
23:1111.c        ****
24:1111.c        **** a = func2(s1, b); /* passing structure by value */
65                      .stabn 68,0,24,.LM6-enter_here
66                      .LM6:
67 0058 3D600000                lis 11,s1@ha
68 005c 392B0000                la 9,s1@l(11)
69 0060 C8090000                lfd 0,0(9)
70 0064 D81F0018                stfd 0,24(31)
71 0068 381F0018                addi 0,31,24
72 006c 7C030378                mr 3,0
73 0070 809F000C                lwz 4,12(31)
74 0074 48000001                bl func2
75 0078 7C601B78                mr 0,3
76 007c 901F0008                stw 0,8(31)
25:1111.c        ****
26:1111.c        **** pa = func3(s1, b); /* func returning int*, uses D0 */
77                      .stabn 68,0,26,.LM7-enter_here
78                      .LM7:
79 0080 3D600000                lis 11,s1@ha
80 0084 392B0000                la 9,s1@l(11)
81 0088 C8090000                lfd 0,0(9)
82 008c D81F0018                stfd 0,24(31)
83 0090 381F0018                addi 0,31,24
84 0094 7C030378                mr 3,0
85 0098 809F000C                lwz 4,12(31)
86 009c 48000001                bl func3
```

```
 87 00a0 7C601B78              mr 0,3
 88 00a4 901F0010              stw 0,16(31)

 27:1111.c        ****
 28:1111.c        **** s2 = &s1;
 89                     .stabn 68,0,28,.LM8-enter_here
 90                     .LM8:
 91 00a8 3D200000              lis 9,s2@ha
 92 00ac 3D600000              lis 11,s1@ha
 93 00b0 380B0000              la 0,s1@l(11)
 94 00b4 90090000              stw 0,s2@l(9)
 29:1111.c        **** a = func4(s2, b); /* passing structure by reference/pointer */
 95                     .stabn 68,0,29,.LM9-enter_here
 96                     .LM9:
 97 00b8 3D200000              lis 9,s2@ha
 98 00bc 80690000              lwz 3,s2@l(9)
 99 00c0 809F000C              lwz 4,12(31)
100 00c4 48000001              bl func4
101 00c8 7C601B78              mr 0,3
102 00cc 901F0008              stw 0,8(31)
 30:1111.c        ****
 31:1111.c        **** s3 = func5(s2, b); /* func returns struct pointer/reference uses D0 */
103                    .stabn 68,0,31,.LM10-enter_here
104                    .LM10:
105 00d0 3D200000              lis 9,s2@ha
106 00d4 80690000              lwz 3,s2@l(9)
107 00d8 809F000C              lwz 4,12(31)
108 00dc 48000001              bl func5
109 00e0 7C601B78              mr 0,3
110 00e4 3D200000              lis 9,s3@ha
111 00e8 90090000              stw 0,s3@l(9)
 32:1111.c        ****
 33:1111.c        **** }
112                    .stabn 68,0,33,.LM11-enter_here
113                    .LM11:
114                    .LBE2:
115                    .stabn 68,0,33,.LM12-enter_here
116                    .LM12:
117                    .L2:
118 00ec 81610000              lwz 11,0(1)
119 00f0 800B0004              lwz 0,4(11)
120 00f4 7C0803A6              mtlr 0
121 00f8 83EBFFFC              lwz 31,-4(11)
122 00fc 7D615B78              mr 1,11
123 0100 4E800020              blr


137                            .globl func
138                            .type    func,@function

139                    func:
 34:1111.c        ****
 35:1111.c        **** int func(int a, int b) {
140                    .stabn 68,0,35,.LM13-func
141                    .LM13:
142 0104 9421FFD8              stwu 1,-40(1)
143 0108 7C0802A6              mflr 0
144 010c 93E10024              stw 31,36(1)
145 0110 9001002C              stw 0,44(1)
146 0114 7C3F0B78              mr 31,1
147 0118 907F0008              stw 3,8(31)
148 011c 909F000C              stw 4,12(31)
 36:1111.c        ****
 37:1111.c        **** int c, d, e;
149                    .stabn 68,0,37,.LM14-func
150                    .LM14:
151                    .LBB3:
 38:1111.c        ****
 39:1111.c        **** c=a; d=b;
152                    .stabn 68,0,39,.LM15-func
153                    .LM15:
154 0120 801F0008              lwz 0,8(31)
155 0124 901F0010              stw 0,16(31)
156 0128 801F000C              lwz 0,12(31)
157 012c 901F0014              stw 0,20(31)
 40:1111.c        ****
 41:1111.c        **** e = func1(c, d);
158                    .stabn 68,0,41,.LM16-func
159                    .LM16:
160 0130 807F0010              lwz 3,16(31)
```

```
161 0134 809F0014              lwz 4,20(31)
162 0138 4CC63182              crxor 6,6,6
163 013c 48000001              bl func1
164 0140 7C601B78              mr 0,3
165 0144 901F0018              stw 0,24(31)
 42:1111.c         ****
 43:1111.c         **** return(c+d);
166                            .stabn 68,0,43,.LM17-func
167                            .LM17:
168 0148 801F0010              lwz 0,16(31)
169 014c 813F0014              lwz 9,20(31)
170 0150 7C004A14              add 0,0,9
171 0154 7C030378              mr 3,0
172 0158 48000004              b .L3
 44:1111.c         ****
 45:1111.c         **** }
173                            .stabn 68,0,45,.LM18-func
174                            .LM18:
175                            .LBE3:
176                            .stabn 68,0,45,.LM19-func
177                            .LM19:
178                            .L3:
179 015c 81610000              lwz 11,0(1)
180 0160 800B0004              lwz 0,4(11)
181 0164 7C0803A6              mtlr 0
182 0168 83EBFFFC              lwz 31,-4(11)
183 016c 7D615B78              mr 1,11

184 0170 4E800020              blr


198                            .globl func1
199                            .type    func1,@function
200                        func1:
 46:1111.c         ****
 47:1111.c         **** int func1(int a, int b) {
201                            .stabn 68,0,47,.LM20-func1
202                            .LM20:
203 0174 9421FFE0              stwu 1,-32(1)
204 0178 93E1001C              stw 31,28(1)
205 017c 7C3F0B78              mr 31,1
206 0180 907F0008              stw 3,8(31)
207 0184 909F000C              stw 4,12(31)
 48:1111.c         ****
 49:1111.c         **** int c, d;
208                            .stabn 68,0,49,.LM21-func1
209                            .LM21:
210                            .LBB4:
 50:1111.c         ****
 51:1111.c         **** c=a; d=b;
211                            .stabn 68,0,51,.LM22-func1
212                            .LM22:
213 0188 801F0008              lwz 0,8(31)
214 018c 901F0010              stw 0,16(31)
215 0190 801F000C              lwz 0,12(31)
216 0194 901F0014              stw 0,20(31)
 52:1111.c         ****
 53:1111.c         **** return(c+d);
217                            .stabn 68,0,53,.LM23-func1
218                            .LM23:
219 0198 801F0010              lwz 0,16(31)
220 019c 813F0014              lwz 9,20(31)
221 01a0 7C004A14              add 0,0,9
222 01a4 7C030378              mr 3,0
223 01a8 48000004              b .L4
 54:1111.c         ****
 55:1111.c         **** }
224                            .stabn 68,0,55,.LM24-func1
225                            .LM24:
226                            .LBE4:
227                            .stabn 68,0,55,.LM25-func1
228                            .LM25:
229                            .L4:
230 01ac 81610000              lwz 11,0(1)
231 01b0 83EBFFFC              lwz 31,-4(11)
232 01b4 7D615B78              mr 1,11
233 01b8 4E800020              blr

246                            .globl func2
247                            .type    func2,@function
```

```
248                     func2:
 56:1111.c          **** 
 57:1111.c          **** int func2(struct s a, int b) {
249                     .stabn 68,0,57,.LM26-func2
250                     .LM26:
251 01bc 9421FFE0               stwu 1,-32(1)
252 01c0 93E1001C               stw 31,28(1)
253 01c4 7C3F0B78               mr 31,1
254 01c8 7C691B78               mr 9,3
255 01cc 909F0008               stw 4,8(31)
 58:1111.c          **** 
 59:1111.c          **** int c, d;
256                     .stabn 68,0,59,.LM27-func2
257                     .LM27:
258                     .LBB5:
 60:1111.c          **** 
 61:1111.c          **** c=a.as; d=b;
259                     .stabn 68,0,61,.LM28-func2
260                     .LM28:
261 01d0 80090000               lwz 0,0(9)
262 01d4 901F000C               stw 0,12(31)
263 01d8 801F0008               lwz 0,8(31)
264 01dc 901F0010               stw 0,16(31)
 62:1111.c          **** 
 63:1111.c          **** return(c+d);
265                     .stabn 68,0,63,.LM29-func2
266                     .LM29:
267 01e0 801F000C               lwz 0,12(31)
268 01e4 817F0010               lwz 11,16(31)
269 01e8 7C005A14               add 0,0,11
270 01ec 7C030378               mr 3,0
271 01f0 48000004               b .L5
 64:1111.c          **** }
272                     .stabn 68,0,64,.LM30-func2
273                     .LM30:
274                     .LBE5:
275                     .stabn 68,0,64,.LM31-func2
276                     .LM31:
277                     .L5:
278 01f4 81610000               lwz 11,0(1)
279 01f8 83EBFFFC               lwz 31,-4(11)
280 01fc 7D615B78               mr 1,11
281 0200 4E800020               blr

294                             .globl func3
295                             .type    func3,@function
296                     func3:
 65:1111.c          **** 
 66:1111.c          **** int* func3(struct s a, int b) {
297                     .stabn 68,0,66,.LM32-func3
298                     .LM32:
299 0204 9421FFE0               stwu 1,-32(1)
300 0208 93E1001C               stw 31,28(1)
301 020c 7C3F0B78               mr 31,1
302 0210 7C691B78               mr 9,3
303 0214 909F0008               stw 4,8(31)
 67:1111.c          **** 
 68:1111.c          **** int c, d;
304                     .stabn 68,0,68,.LM33-func3
305                     .LM33:
306                     .LBB6:
 69:1111.c          **** 
 70:1111.c          **** c=a.as; d=b;
307                     .stabn 68,0,70,.LM34-func3
308                     .LM34:
309 0218 80090000               lwz 0,0(9)
310 021c 901F000C               stw 0,12(31)
311 0220 801F0008               lwz 0,8(31)
312 0224 901F0010               stw 0,16(31)
 71:1111.c          **** 
 72:1111.c          **** return(&a);
313                     .stabn 68,0,72,.LM35-func3
314                     .LM35:
315 0228 7D234B78               mr 3,9
316 022c 48000004               b .L6
 73:1111.c          **** }
317                     .stabn 68,0,73,.LM36-func3
318                     .LM36:
319                     .LBE6:
320                     .stabn 68,0,73,.LM37-func3
```

```
321                     .LM37:
322                     .L6:
323 0230 81610000               lwz 11,0(1)
324 0234 83EBFFFC               lwz 31,-4(11)
325 0238 7D615B78               mr 1,11
326 023c 4E800020               blr

339                             .globl func4
340                             .type    func4,@function
341                     func4:
 74:1111.c          ****
 75:1111.c          **** int func4(struct s *a, int b) {
342                     .stabn 68,0,75,.LM38-func4
343                     .LM38:
344 0240 9421FFE0               stwu 1,-32(1)
345 0244 93E1001C               stw 31,28(1)
346 0248 7C3F0B78               mr 31,1
347 024c 907F0008               stw 3,8(31)
348 0250 909F000C               stw 4,12(31)
 76:1111.c          ****
 77:1111.c          **** int c, d;
349                     .stabn 68,0,77,.LM39-func4
350                     .LM39:
351                     .LBB7:
 78:1111.c          ****
 79:1111.c          **** c=a->as; d=a->bs;
352                     .stabn 68,0,79,.LM40-func4
353                     .LM40:
354 0254 813F0008               lwz 9,8(31)
355 0258 80090000               lwz 0,0(9)
356 025c 901F0010               stw 0,16(31)
357 0260 813F0008               lwz 9,8(31)
358 0264 80090004               lwz 0,4(9)
359 0268 901F0014               stw 0,20(31)
 80:1111.c          ****
 81:1111.c          **** return(c+d);
360                     .stabn 68,0,81,.LM41-func4
361                     .LM41:
362 026c 801F0010               lwz 0,16(31)
363 0270 813F0014               lwz 9,20(31)
364 0274 7C004A14               add 0,0,9
365 0278 7C030378               mr 3,0
366 027c 48000004               b .L7
 82:1111.c          ****
 83:1111.c          **** }
367                     .stabn 68,0,83,.LM42-func4
368                     .LM42:
369                     .LBE7:
370                     .stabn 68,0,83,.LM43-func4
371                     .LM43:
372                     .L7:
373 0280 81610000               lwz 11,0(1)
374 0284 83EBFFFC               lwz 31,-4(11)
375 0288 7D615B78               mr 1,11
376 028c 4E800020               blr

389                             .globl func5
390                             .type    func5,@function
391                     func5:
 84:1111.c          ****
 85:1111.c          ****
 86:1111.c          **** struct s* func5(struct s *a, int b) {
392                     .stabn 68,0,86,.LM44-func5
393                     .LM44:
394 0290 9421FFE0               stwu 1,-32(1)
395 0294 93E1001C               stw 31,28(1)
396 0298 7C3F0B78               mr 31,1
397 029c 907F0008               stw 3,8(31)
398 02a0 909F000C               stw 4,12(31)
 87:1111.c          ****
 88:1111.c          **** int c, d;
399                     .stabn 68,0,88,.LM45-func5
400                     .LM45:
401                     .LBB8:
 89:1111.c          ****
 90:1111.c          **** c=a->as; d=a->bs;
402                     .stabn 68,0,90,.LM46-func5
403                     .LM46:
404 02a4 813F0008               lwz 9,8(31)
405 02a8 80090000               lwz 0,0(9)
```

```
406 02ac 901F0010              stw 0,16(31)
407 02b0 813F0008              lwz 9,8(31)
408 02b4 80090004              lwz 0,4(9)
409 02b8 901F0014              stw 0,20(31)
 91:1111.c          ****
 92:1111.c          **** return(a);
410                       .stabn 68,0,92,.LM47-func5
411                       .LM47:
412 02bc 801F0008              lwz 0,8(31)
413 02c0 7C030378              mr 3,0
414 02c4 48000004              b .L8
 93:1111.c          **** }
 421 02c8 81610000             lwz 11,0(1)
422 02cc 83EBFFFC              lwz 31,-4(11)
423 02d0 7D615B78              mr 1,11
424 02d4 4E800020              blr
```

5. Frequently Asked Questions

why stack is so important ?

probably, it is not in the listing code, so serial flow visualizing may not give some memory operation clearly. may be memory with a difference.

any idea why function parameters pushing from right to left !

good Q. very much useful in case of variable no. of arguments passing to get knowing no. of arguments to be parsed if that info is kept in the stack at the fixed offset. this info(1st arg) would be followed by return address generally. this is C lang convention.

stack frame pointer is needed ?

good Q. i am still looking for answer. perhaps, you say the same functionalities can also be accomplished using current SP.