

Procedures and Stacks in Assembly Language

Procedures (functions) are very important for writing modular, reusable, maintainable code, in assembly language just like in high-level languages.

PowerPC features for procedure/function calls:

- link register (LR)
- bl: branch and link
- blr: branch to link register

ABIs

- To implement a procedure call, the calling procedure (the *caller*) and the called procedure (the *callee*) must agree on:
 - how to pass in parameters
 - how to return the return value (if any)
 - where the call stack is (for local variables etc.)
 - etc.
- The PowerPC *architecture* defines none of this
- Instead, these are defined by convention
- An *application binary interface* (ABI) defines everything needed for a binary object file to run on a system (CPU plus operating system), so that it can:
 - call system library routines
 - call (and be called by) code generated by other people & other compilers
- The ABI includes:
 - file format
 - rules for linker operation
 - procedure calling conventions
 - register usage conventions
- PowerPC has different (but very similar) ABIs for MacOS, AIX, embedded systems (EABI), SVR4 Unix, Windows NT

A Very Simple Calling Convention

- Arguments are passed in GPRs r3 to r10
 - first argument in r3, second in r4, etc.
 - more than eight arguments: use stack (in memory)
- Return value is passed back in r3

Example:

```
int func(int a, int b)
{
    return (a+b);
}
```

```
func(3,4);
```

A problem

```
int func2(int a, int b)
{
    return func(a, b);
}
```

```
func2(3,4);
```

The Stack

- Need to save per-function-invocation information (e.g. link register value) on the *call stack*
- Each function invocation has its own *stack frame* (a.k.a. *activation record*)
- Conventional PowerPC stack pointer is r1
- Other stack conventions:
 - stack pointer is always doubleword-aligned (8 byte)
 - stack grows down (toward address 0)
 - r1 points to lowest stack address (bottom of current stack frame)
 - first item in stack frame (offset 0 from r1) is address of previous stack frame
 - second item (offset 4) is saved link register
 - minimum stack frame is 8 bytes
 - stack frame optional for “leaf” functions
 - always use update addressing mode to allocate stack frame atomically

Func2 revisited

```
int func2(int a, int b)
{
    return func(a, b);
}
```

Local variables

```
int func3(int a, int b, int c)
{
    int temp;

    temp = func2(b, c);

    return func2(temp, a);
}
```

- Where can we keep a?

Register Usage Convention

Need some rules about who gets to use which registers

- Callee-save
- Caller-save
- Reserved
- PowerPC ABI conventions:
 - Caller-save: r0, r3-r12, LR, CTR, CR0, CR1, CR5-7
 - Callee-save: r14-r31, CR2-CR4
 - Reserved: r1, r2, r13