MIPS o32 Calling Convention Examples

Jeffrey Knockel <jeffk at cs dot unm dot edu>

This is intended to teach the MIPS o32 Calling Convention by example but should not be considered a complete substitute for reading the official MIPS ABI.

A simple non-leaf function that takes two arguments:

Memory Addresses ↑

Stack Grows ↓

Old sp →	Slot for incoming a3	N/A	16 bytes	36(sp)
	Slot for incoming a2	N/A		32(sp)
	Slot for incoming a1	a1		28(sp)
	Slot for incoming a0	a0		24(sp)
	Register Save Area	ra ($31)	8 bytes	20(sp)
		Padding		16(sp)
	Slot for outgoing a3	Reserved	16 bytes	Do not use
	Slot for outgoing a2			
	Slot for outgoing a1			
New sp →	Slot for outgoing a0			

```
addiu   sp, sp, -24
sw      ra, 20(sp)
sw      a1, 28(sp)
sw      a0, 24(sp)
// storing arguments e.g. a0, a1, etc. is optional--do it only if
// convenient.  here, let's say we want/have to.
. . .
lw      ra, 20(sp)
addiu   sp, sp, 24
jr      ra
```

Note the following things:

- A caller reserves four words (16 bytes) at the end of its stack frame for the callee to store its arguments, even if the callee takes fewer than four arguments, even if the callee does not actually use this space. In other words, if you are a non-leaf function, then you must never address 0(sp), 4(sp), 8(sp), or 12(sp)! However, supposing your frame is 32 bytes, then you may use 32(sp), 36(sp), 40(sp), and 44(sp) for storing a0, a1, a2, and a3, respectively, even though this is in the frame of your caller!
- All of the different sections on the stack must be double-word (8 byte) aligned. This is just so that we can store double words onto the stack without them being unaligned. In the case above, we have to pad the register save area to maintain this alignment.
- As a corollary to the above, the minimum size for the stack frame of a non-leaf function is 24 bytes. This is 16 bytes for the outgoing argument slots and a minimum of 8 bytes for the register save area (we always have to save the return address, and the sections on the stack must be 8 byte aligned, so the register save area is a minimum of 8 bytes). The minimum size for the stack frame of a leaf function is 0 bytes, but more on that later.

A more complicated non-leaf function that takes two arguments:

Memory Addresses ↑

Stack Grows ↓

	Slot for incoming a3	N/A	16 bytes	84(sp)
	Slot for incoming a2	N/A		80(sp)
	Slot for incoming a1	Unused		76(sp)
	Slot for incoming a0	Unused		72(sp)
Old sp →	Local Variables	int ary[10]	40 bytes	68(sp) … 32(sp)
	Register Save Area	ra ($31)	16 bytes	28(sp)
		s1 ($17)		24(sp)
		s0 ($16)		20(sp)
		Padding		16(sp)
	Slot for outgoing a3	Reserved	16 bytes	Do not use
	Slot for outgoing a2			
	Slot for outgoing a1			
New sp →	Slot for outgoing a0			

```
addiu   sp, sp, -72
sw      ra, 28(sp)
sw      s1, 24(sp)
sw      s0, 20(sp)
// sw      a1, 76(sp)
// sw      a0, 72(sp)
// storing arguments e.g. a0, a1, etc. is optional--do it only if
// convenient.  here, let's say we do not want/have to.
. . .
lw      s0, 20(sp)
lw      s1, 24(sp)
lw      ra, 28(sp)
addiu   sp, sp, 72
jr      ra
```

Now note:

- Registers in register save area are stored in numerical order (higher registers in higher addresses).
- An array is allocated above the register save area in the local variable space.

For simple leaf functions, we do not have to allocate any stack frame. Why? We do not need the outgoing arguments section, because, by definition, we do not call any functions, and functions that we call is the only thing that could use this space. We do not need to save the return address to the register save area, because, again, by definition, we do not call any functions, so nothing will clobber this register. For temporary registers, in a leaf function, we can use the t# registers instead of the s# registers, so we won't have to store or load them to the register save area either, assuming we do not run out of registers. So, for simple leaf functions we do not need a register save area either. Since, for simple leaf functions, we need neither the outgoing arguments section nor the register save area, we do not need a stack frame.