MN10300 FUNCTION CALL ABI

GENERAL

The MN10300/AM33 kernel runs in little-endian mode; big-endian mode is not supported.

The stack grows downwards, and should always be 32-bit aligned. There are separate stack pointer registers for userspace and the kernel.

ARGUMENT PASSING

The first two arguments (assuming up to 32-bits per argument) to a function are passed in the D0 and D1 registers respectively; all other arguments are passed on the stack.

If 64-bit arguments are being passed, then they are never split between registers and the stack. If the first argument is a 64-bit value, it will be passed in D0:D1. If the first argument is not a 64-bit value, but the second is, the second will be passed entirely on the stack and D1 will be unused.

Arguments smaller than 32-bits are not coalesced within a register or a stack word. For example, two byte-sized arguments will always be passed in separate registers or word-sized stack slots.

CALLING FUNCTIONS

The caller must allocate twelve bytes on the stack for the callee's use before it inserts a CALL instruction. The CALL instruction will write into the TOS word, but won't actually modify the stack pointer; similarly, the RET instruction reads from the TOS word of the stack, but doesn't move the stack pointer beyond it.

```
        Stack:
        |               |
        |               |
        |---------------| SP+20
        | 4th Arg       |
        |---------------| SP+16
        | 3rd Arg       |
        |---------------| SP+12
        | D1 Save Slot  |
        |---------------| SP+8
        | D0 Save Slot  |
        |---------------| SP+4
        | Return Addr    |
        |----------------|  SP
        |                |
        |                |
```

The caller must leave space on the stack (hence an allocation of twelve bytes) in which the callee may store the first two arguments.

RETURN VALUE

The return value is passed in D0 for an integer (or D0:D1 for a 64-bit value), or A0 for a pointer.

If the return value is a value larger than 64-bits, or is a structure or an array, then a hidden first argument will be passed to the callee by the caller: this will point to a piece of memory large enough to hold the result of the function. In this case, the callee will return the value in that piece of memory, and no value will be returned in D0 or A0.

REGISTER CLOBBERING

The values in certain registers may be clobbered by the callee, and other values must be saved:

```
	Clobber:	D0-D1, A0-A1, E0-E3
	Save:		D2-D3, A2-A3, E4-E7, SP
```

All other non-supervisor-only registers are clobberable (such as MDR, MCRL, MCRH).

SPECIAL REGISTERS

Certain ordinary registers may carry special usage for the compiler:

```
	A3:	Frame pointer
	E2:	TLS pointer
```

KERNEL ABI

The kernel may use a slightly different ABI internally.

 (*) E2

If CONFIG_MN10300_CURRENT_IN_E2 is defined, then the current task pointer will be kept in the E2 register, and that register will be marked unavailable for the compiler to use as a scratch register.

Normally the kernel uses something like:

```
	MOV	SP,An
	AND	0xFFFFE000,An
	MOV	(An),Rm		// Rm holds current
	MOV	(yyy,Rm)	// Access current->yyy
```

To find the address of current; but since this option permits current to be carried globally in an register, it can use:

```
	MOV	(yyy,E2)	// Access current->yyy
```

instead.

SYSTEM CALL ABI

System calls are called with the following convention:

REGISTER	ENTRY	EXIT
D0	Syscall number	Return value
A0	1st syscall argument	Saved
D1	2nd syscall argument	Saved
A3	3rd syscall argument	Saved
A2	4th syscall argument	Saved
D3	5th syscall argument	Saved
D2	6th syscall argument	Saved

All other registers are saved. The layout is a consequence of the way the MOVM instruction stores registers onto the stack.