

SPARC Assembler Language Programming in a Nutshell

Registers

General purpose

- 32 registers %r0..%r31 - 32 bits each
- Used as four groups of 8 registers each:

Reg#	Group	Assem. Syntax
0..7	Globals	%g0..%g7
8..15	Outs	%o0..%o7 (letter "oh")
16..23	Locals	%l0..%l7 (letter "ell")
24..31	Ins	%i0..%i7

- Register %g0 is a constant 0 value. It cannot be altered.
- There is only one set of global registers, shared by all.
- There are 2..32 other sets of 16 registers (32..512 regs).
- At any time, you have a "window" into these sets.
- %l0..%l7 and %i0..%i7 (%r16..%r31) are the current set.
- %o0..%o7 (%r8..%r15) are the In regs from the next set.
- SAVE/RESTORE instructions switch to the next/previous set.
- %i7 and %o7 are used for return addresses. DO NOT USE!
- %i6 and %o6 are used for stack/frame pointers.
- Aliases: %sp = %o6 %fp = %i6.

Special purpose

PC - Program Counter - address of executing instruction
nPC - Next Program Counter - address of instruction being fetched
%y - Holds high-order half of product/dividend for multiply/divide
PSR - Processor State Register - includes many fields, including
iCC - integer condition code (N Z V C bits)
CWP - current window pointer - selects the register set

Assembly Language Syntax

- Comments: ! to end of line, or between /* and */
- Labels are case-sensitive, always terminated by a colon (:)
- Labels may contain _ \$ and . characters.
- Labels that begin with _ \$ and . are special. DO NOT USE!
- Use 0x to prefix hexadecimal values (e.g. 0xF123ABC).
- Pseudo-operations (assembler directives) always begin with ".".
- Either 'chars' or "chars" may be used for ascii characters.
- Special symbols (such as register names) begin with "%".

Addressing Modes

There are no orthogonal "modes". There are only a few combinations of operands allowed in the basic instructions, which fall (with very few exceptions) into the following categories:

1) Load/Store (memory) instructions:

```
opcode [%reg+const],%reg
opcode [%reg+%reg],%reg
(reverse the operand order for Store instructions)
```

2) Arithmetic/Logical/Shift instructions:

```
opcode %reg,%reg,%reg
opcode %reg,const,%reg
```

3) Branch instructions:

```
opcode address
```

- In the above, "const" is a 13-bit signed integer (-4096..4095),
- "address" is a 22-bit signed integer (but branch instructions add two low-order 0's since all instructions are word-aligned, giving, in effect, 24 bits (+/- 8 Meg)).
- The assembler accepts [%reg] which it turns into [%reg+%g0].
- Examples:

```
LD [%i3+%L1],%L3 ! Loads reg L3
ST %L2,[%sp - 48] ! Stores L2 in stack
LD [%L1],%L2 ! L1 contains an address
ADD %L1,%L2,%L3 ! L1+L2 placed in L3
ADD %g1,48,%g1 ! add 48 to reg G1
```

Basic Instruction Set

Load/Store (memory) instructions

- Only these instructions reference data stored in memory.
- | | |
|------|---|
| LDUB | Load Unsigned Byte (padded with 0's) |
| LDSB | Load Signed Byte (sign-extended) |
| LDUH | Load Unsigned Halfword (padded with 0's) |
| LDSH | Load Signed Halfword (sign-extended) |
| LD | Load (a word - 32 bits - 4 bytes) |
| LDD | Load Doubleword (register # must be even) |

STB	Store Byte
STH	Store Halfword
ST	Store (a word)
STD	Store Doubleword (register # must be even)

- After a load, the data always occupies the entire register(s).
- All addresses must be aligned (by 2/4/8 for half/word/double).

Arithmetic/Logical

ADD	addition (Op1+Op2->Op3)
SUB	subtraction (Op1-Op2->Op3)
AND	bitwise AND (Op1^Op2->Op3)
ANDN	bitwise AND, Op2 inverted (Op1^Op2'->Op3)
OR	bitwise OR (Op1vOp2->Op3)
ORN	bitwise OR, Op2 inverted (Op1vOp2'->Op3)
XOR	bitwise exclusive OR (Op1 exor Op2->Op3)
XNOR	bitwise exclusive NOR (Op1 exor Op2'->Op3)

- To set the condition code (iCC), add "cc" to any of the above.
- Example: SUBcc %L1,%L2,%G0 !compare L1 to L2
- All operations always use the full 32 bits of the register(s).

SETHI/NOP

- Has a special instruction format and syntax:
SETHI const,%reg
- Places the 22-bit constant in the *high-order* 22 bits of %reg.
- The low-order 10 bits of the register are cleared to 0's.
- The special assembler functions %hi(x) and %lo(x) will give the high-order 22 bits and low-order 10 bits of any constant x.
- To place a 32-bit constant X (such as a fixed address) in a reg:
SETHI %hi(X),%reg ! Set the high 22 bits
OR %reg,%lo(X),%reg! Insert the low 10 bits
- The assembler will accept SET X,%reg instead of the above.
- NOP (no operation) (takes no parameters) is really SETHI 0,%G0

SAVE/RESTORE

- Identical to ADD except that SAVE/RESTORE switch to the next/previous set of registers.
- The addition is typically used to modify the stack pointer.
- Example:
SAVE %sp,-96,%sp
! switch register windows and allocate 96 bytes
- RESTORE with no operands is really RESTORE %g0,%g0,%g0

Branch

BA	Branch Always
BE	Branch Equal
BNE	Branch Not Equal
BL	Branch Less
BLE	Branch Less or Equal
BG	Branch Greater
BGE	Branch Greater or Equal

- The above (except BA) should be used following operations on *signed* arithmetic only. There are others for *unsigned* arithmetic.

SPARC Assembler Language Programming in a Nutshell

Delayed Branching

- All branches (including the one caused by CALL, below) take place *after* execution of the following instruction.
- The position immediately after a branch is the “delay slot” and the instruction found there is the “delay instruction”.
- If possible, place a useful instruction in the delay slot (one which can safely be done whether or not a conditional branch is taken).
- If not, place a NOP in the delay slot.
- *Never* place any other branch instruction in a delay slot.
- Do not use SET in a delay slot (only half of it is really there).

Calling Library Routines

- Use

```
CALL    address
```
- Place the parameters (up to 6) in %o0..%o5 first.
- A scalar result, if any, will be returned in %o0.
- Any standard C library routine may be used.
- Example of using “printf” for output:

```
SET    outstr,%o0
CALL   printf
ADD    %L1,%G0,%o1 !done before CALL
....
outstr: .asciz    "Register L1 contains %d\n"
```
- Example of using “scanf” for input:

```
SET    instr,%o0
SET    data,%o1    !ADDRESS of data
CALL   scanf
NOP                    ! Don't put SET here!
...
instr:  .asciz    "%d"
data:   .word
```

Pseudo-operations (Assembler directives)

- Forcing proper boundary alignment:

```
.align 2/4/8
```
- Selecting the code segment or data segment:

```
.section ".text" ! for the code
.section ".data" ! for the data
```
- Allocating blocks of memory

```
label: .skip num-of-bytes
```
- Allocating initialized memory

```
.byte value[,value,value...]
.half value[,value,value...]
.word value[,value,value...]
(Don't use .double - it gives real values!)
.ascii "characters"
.asciz "characters" !adds a NULL (\0) at the end
```
- Declaring symbols to be used externally

```
.global symbol
```

Minimal Linkage

- The environment expects at least 92 bytes (23 words) of temporary storage on the stack. (Register window overflow/underflow, other interrupts, and library routines may need and use this memory.) The stack pointer must be doubleword aligned (divisible by 8), so 96 is the minimum allocation.
- The entry point into the program is the global symbol “main”.
- The minimal linkage for a program is therefore

```
.global main
main: save    %sp,-96,%sp
```
- To return to the operating system, use

```
ret    ! standard subprogram return instruction
restore ! restore the old set of registers (delay slot)
```

Synthetic Instructions

As a RISC architecture, SPARC lacks many common instructions found on other processors. For convenience, the assembler provides many *synthetic instructions* which it translates for you.

Synthetic Instruction	Assembled As
clr %reg	or %g0,%g0,%reg
clr [address]	st %g0,[address]
clrh [address]	sth %g0,[address]
clrb [address]	stb %g0,[address]
cmp %reg,%reg	subcc %reg,%reg,%g0
cmp %reg,const	subcc %reg,%const,%g0
dec %reg	sub %reg,1,%reg
deccc %reg	subcc %reg,1,%reg
inc %reg	add %reg,1,%reg
inccc %reg	addcc %reg,1,%reg
mov %reg,%reg	or %g0,%reg,%reg
mov const,%reg	or %g0,const,%reg
not %reg	xnor %reg,%g0,%reg
neg %reg	sub %g0,%reg,%reg
restore	restore %g0,%g0,%g0
ret	jmp! %i7+8,%g0
set const22,%reg	sethi %hi(const22),%reg
	or %reg,%lo(const22),%reg
tst %reg	orcc %reg,%g0,%g0

Sample Program

```
!==== Minimal prologue ====
.section ".text"
.global main
.align 4
main: save    %sp,-96,%sp !minimum!
!=====

! Call a routine (printf) to print a message.
set    string,%o0
call   printf
nop    !Always fill the "delay slot"!

!====Standard epilogue====
ret    !Return to the OS.
restore !Delay slot - done first
!=====

! The data section
.section ".data"
string: .asciz "Hello, world!\n"
```