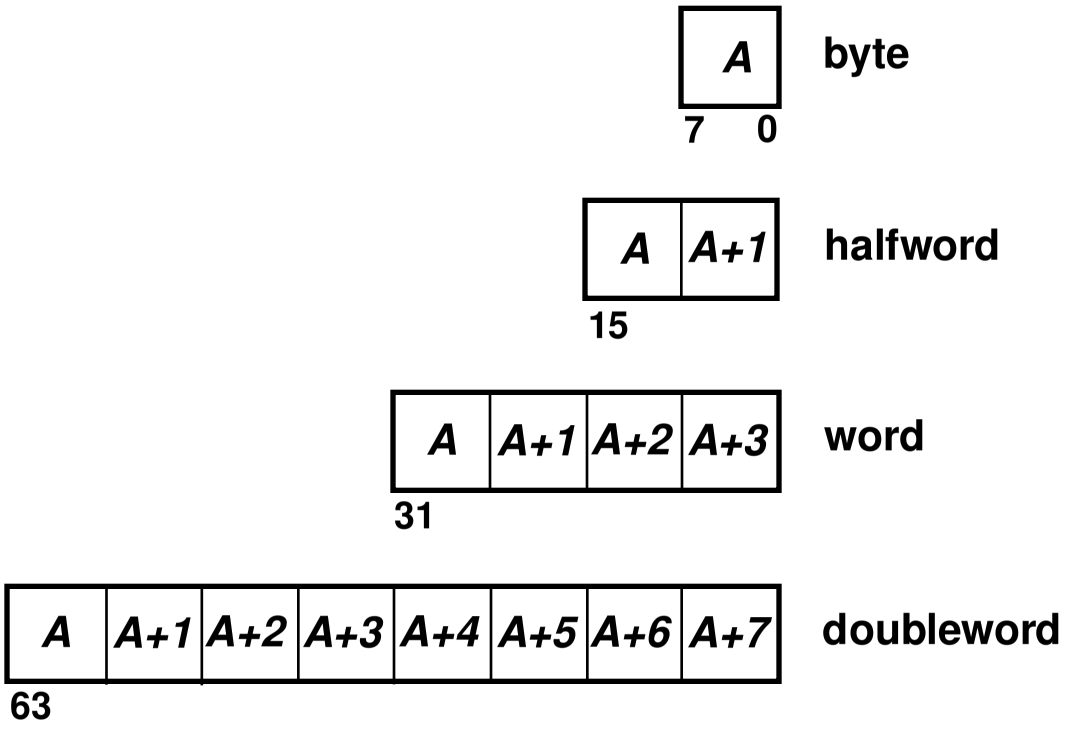
SPARC Architecture

- 8-bit cell (byte) is smallest addressable unit
- 32-bit addresses, i.e., 32-bit virtual address space
- Larger sizes: at address ***A***



- SPARC is a ***big-endian*** (big end, or most-significant end, first) machine

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

SPARC Registers

- 32, 32-bit wide general-purpose registers

			`%r0`	... `%r31`	
`%g0`	... `%g7`		`%r0`	... `%r7`	global
`%o0`	... `%o7`		`%r8`	... `%r15`	output
`%l0`	... `%l7`		`%r16`	... `%r23`	local
`%i0`	... `%i7`		`%r24`	... `%r31`	input

- The groups relate to procedure calling conventions
- Some registers have ***dedicated uses***

`%sp` (`%r14`)	stack pointer
`%fp` (`%r30`)	frame pointer
`%r15`	temporary
`%r31`	return address

- Register `%g0` always has the value 0 when read; writing it has no effect
- Other special registers (manipulated by special instructions):

  floating point registers (`%f0`...`%f31`)
  program counter (`pc`)
  next program counter (`npc`)
  `PSR`, `TBR`, `WIM`, `Y`

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

SPARC Register Map

- see page 193 in the SPARC Architecture Manual, §2.2 in Paul

	`%i7`	`%r31`	return address - 8
	`%i6, %fp`	`%r30`	frame pointer
in	`%i5`	`%r29`	incoming parameter 6
	...	...	...
	`%i0`	`%r24`	incoming parameter 1/return value *to* caller
	`%l7`	`%r23`	local 7
local	...	...	...
	`%l0`	`%r16`	local 0
	`%o7`	`%r15`	temporary value/address of `call` instruction
	`%o6, %sp`	`%r14`	stack pointer
out	`%o5`	`%r13`	outgoing parameter 6
	...	...	...
	`%o0`	`%r8`	outgoing parameter 1/return value *from* caller
	`%g7`	`%r7`	global 7
global	...	...	...
	`%g1`	`%r1`	temporary value
	`%g0`	`%r0`	0

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

SPARC Register Map, cont'd

- Other registers

state	`%y`	**Y register**
	`%psr`	**integer condition codes**
	`%fsr`	**floating point condition codes**
	`%csr`	**coprocessor condition codes**
floating point	`%f31`	**floating point value**
	... **...**	**...**
	`%f0`	**floating point value**

- Register save conventions: what happens across calls?

`%g2 ... %g7`	saved?
`%g1`	destroyed
`%o0 ... %o5, %o7`	destroyed
`%o6`	saved
`%l0 ... %l7`	saved
`%i0 ... %i7`	saved
`%f0 ... %f31`	saved

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

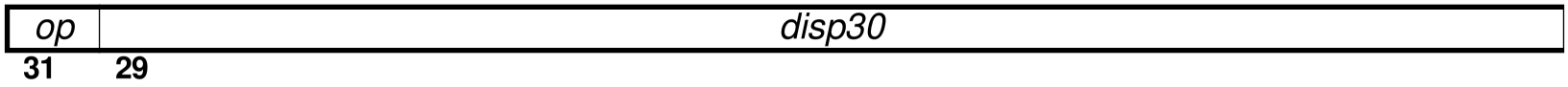
SPARC Instruction Set

- Instruction groups

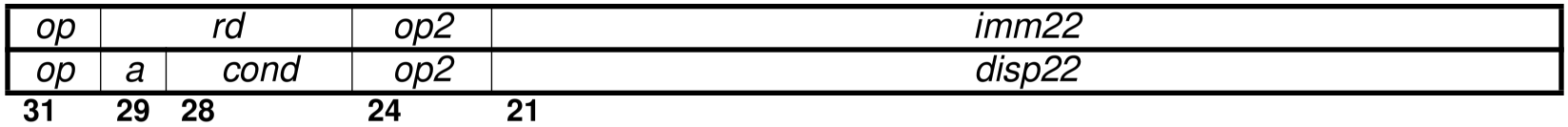
  load/store instructions
  integer arithmetic and bitwise logical instructions
  control instructions (branches, calls)
  special instructions (operating system)
  floating point arithmetic and conversion

- Instruction formats (see page 44, Ch. 8 in Paul)

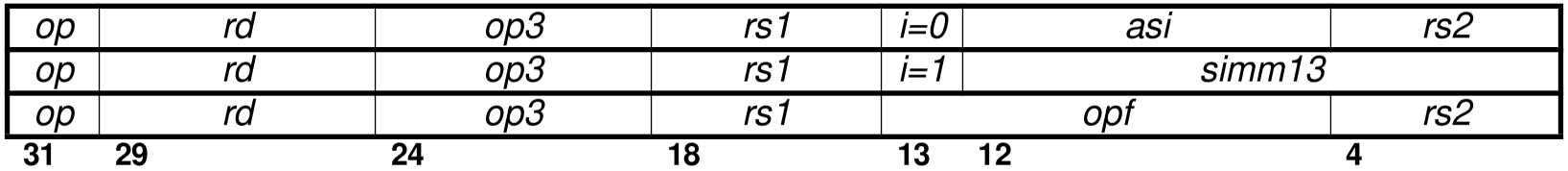
  format 1 (*op* = 1): **`call`**



  format 2 (*op* = 0): **`sethi`** and branches (**`bi`*`cc`***, **`fb`*`fcc`***, **`cb`*`ccc`***)



  format 3 (*op* = 2 or 3): remaining instructions



Copyright ©1995 D. Hanson, K. Li & J.P. Singh

Assembly vs. Machine Language

- ***Machine language*** is the ***bit patterns*** that represent instructions
- ***Assembly language*** is a ***symbolic representation*** of machine language
- ***Assemblers*** translate from assembly language to machine language

  `add %i1,360,%o2` is a format 3 instruction: `022401460550`

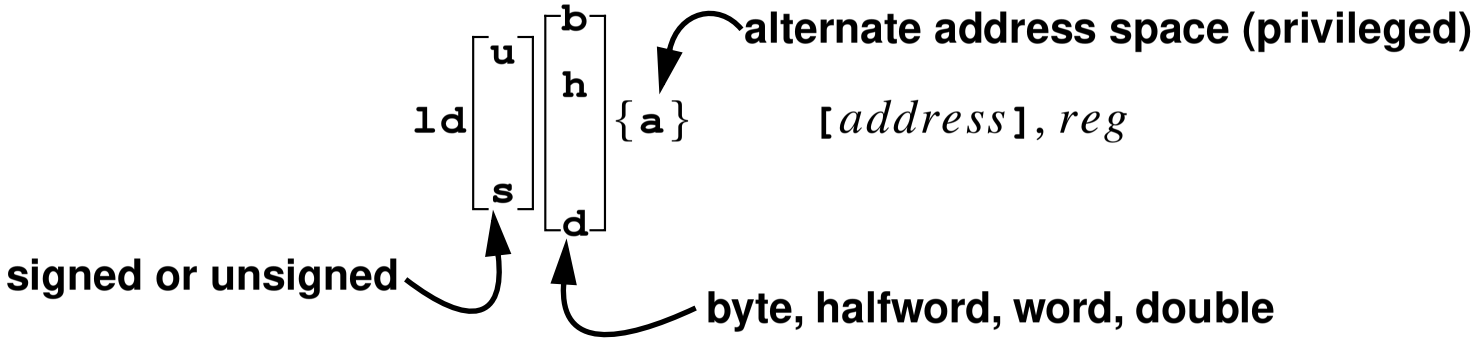
2	10	0	25	1	360
31	**29**	**24**	**18**	**13**	**12**
2	12	0	31	1	550 octal

- Assemblers: mapping an assembly instruction to a machine instruction (1-to-1)
- Compilers: mapping a statement to 1 or many assembly instructions
- ***Disassemblers*** translate from machine language to ***an*** assembly language

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

Load Instructions

- Load: move data from memory to a register



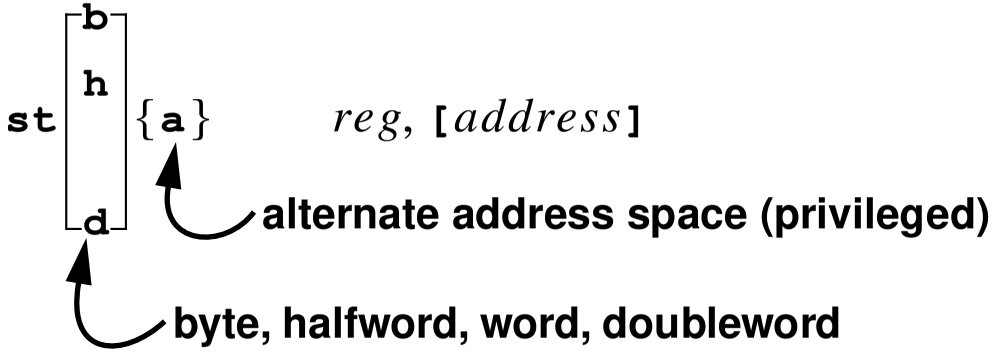
- Fetched byte or halfword appears right-justified in the 32-bit register
- Leftmost bits are zero-filled or sign-extended
- A double word is loaded into a register ***pair*** **and** reg must be even
  - the most-significant word lands in reg
  - the least-significant word in $reg + 1$
- Addresses must be ***aligned***: for address A

halfword	$A \bmod 2 = 0$
word	$A \bmod 4 = 0$
double word	$A \bmod 8 = 0$

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

Store Instructions

- Move data from a register to memory



- Storing bytes and halfwords

  the rightmost bits are stored

  the leftmost bits are ignored

- Storing double words

  reg must be even

Copyright ©1995 D. Hanson, K. Li & J.P. Singh

Addressing Modes

- SPARC has two ***addressing modes*** to yield an ***effective address***
  1. add the contents of two registers
  2. add the contents of a register and a signed, 13-bit number
- Common names
  1. register indirect or deferred

     ```
     ld [%o1],%o2
     ```

  1. register indexed (above is a special case that uses `%g0`)

     ```
     st %o1,[%o2+%o3]
     ```

  2. register displacement or based

     ```
     ld [%o1+10],%o2
     ```

- Assembly-language syntax: *N* is a 13-bit integer constant

address	*synonym*
reg	*reg* + `%g0`
reg + *reg*	
reg + *N*	
N + *reg*	*reg* + *N*
N	`%g0` + *N*

Copyright ©1995 D. Hanson, K. Li & J.P. Singh