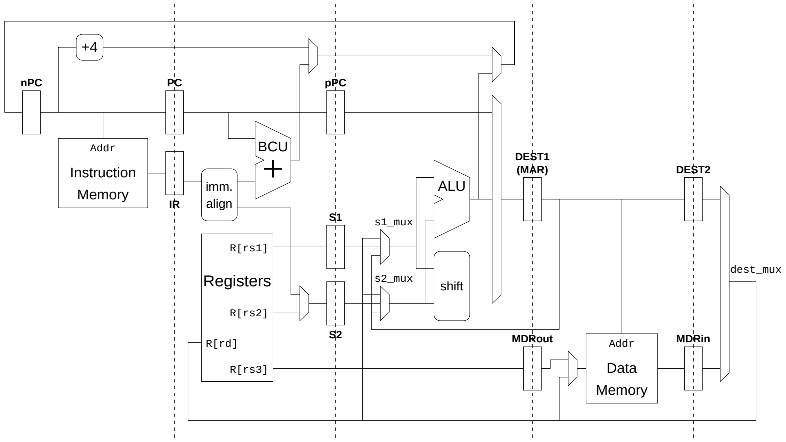


Typical RISC

- Few instructions
- Very few addressing modes
- Architecture
  - Three Address
  - Register-Register
  - Load-Store

The processor described here is based on a subset of the SPARC specification[1].
This implementation employs a Harvard memory architecture.

[1]note that SPARC is not an architecture only a specification based around a complete instruction set.

The ALU supports a small number of functions based on only five control signals.

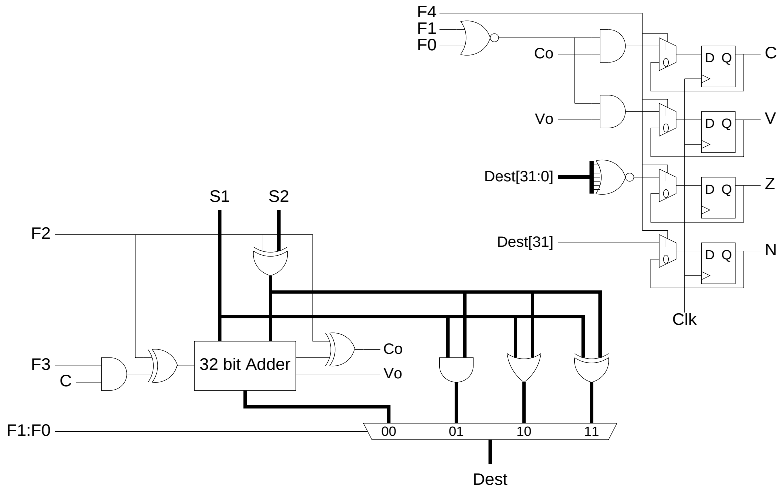
A separate barrel shifter supports three simple shift instructions.

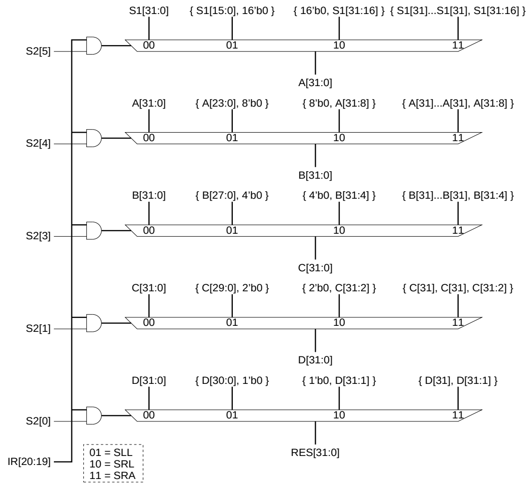
ALU instructions

F4:F3:F2 \ F1:F0	0 0	0 1	1 0	1 1
0 0 0	ADD	AND	OR	XOR
0 0 1	SUB	ANDN	ORN	XNOR
0 1 0	ADDX			
0 1 1	SUBX			
1 0 0	ADDcc	ANDcc	ORcc	XORcc
1 0 1	SUBcc	ANDNcc	ORNcc	XNORcc
1 1 0	ADDXcc			
1 1 1	SUBXcc			

Shift instructions

SLL SRL SRA





- All ALU and shift instructions take two operands (either two registers or one register and one immediate) and return a single register result.

  `SUBX R3,R7,R5` $R5' \leftarrow R3 - R7 - C$

  `ADD R4,5,R1` $R1' \leftarrow R4 + 5$

  – for special cases we have a pseudo register, $R0$, which always contains zero.

  `SUB R0,R2,R2` $R2' \leftarrow -R2$

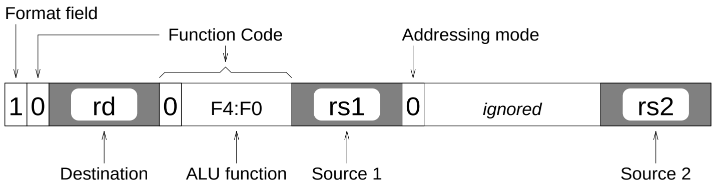
  – we may even execute an instruction merely for its side effects

  `SUBcc R5,201,R0`

Instruction Format

- Arithmetic and logic instructions contain all the relevant fields for register selection and ALU control.

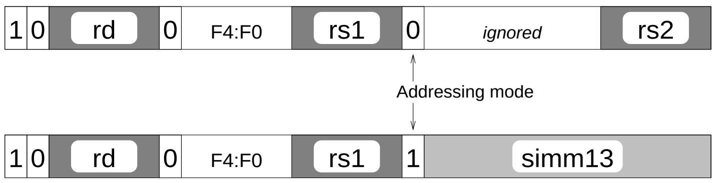
  The decoder is responsible for enabling these fields during the correct cycle.



Variable Instruction Format

The use of variable fields allows us to specify more information within a limited instruction width[2].

For simplicity the signed immediate may only be used on the **s2** bus.

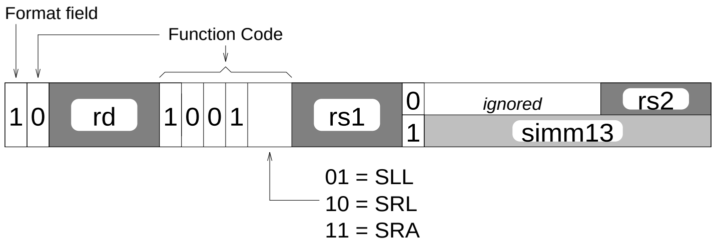


[2] note that the **rs2** field is not always present but when present it is always in the same place.

SPARCjnr

- Shift instructions are indicated by a different function code.

  The decoder will use a part of the function code in order to select which functional unit drives the destination bus.



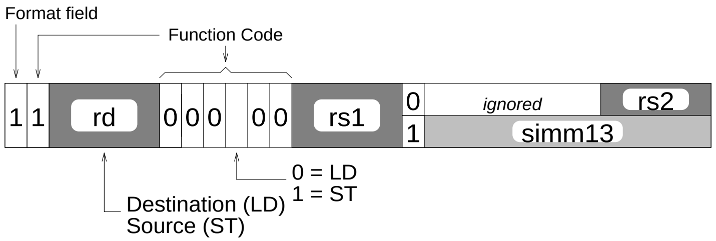
Note that only the least significant 5 bits of the simm13 or R[rs2] are used to determine the amount of shift.

SPARCjnr

- Load and store instructions use similar instruction formats.

```
LD [R3+R7],R5
```
$R5' \leftarrow mem(R3 + R7)$

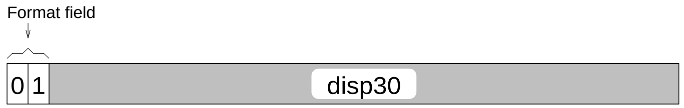
```
ST R1,[R4+5]
```
$mem(R4 + 5)' \leftarrow R1$



SPARCjnr

- Control Transfer instructions.

CALL



Since instructions are word aligned, any location within the 4 GByte address space may be accessed via the `CALL` instruction[3].

`CALL` *label*

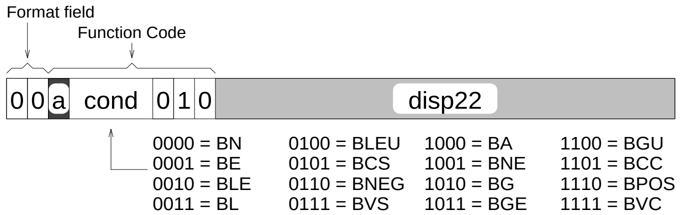
$R15' \leftarrow PC$
$PC' \leftarrow nPC$
$nPC' \leftarrow PC + (disp30 * 4)$

[3] A price is paid for this facility – R15 is fixed as the register storing the return address.

SPARCjnr

- Control Transfer instructions.

Bicc



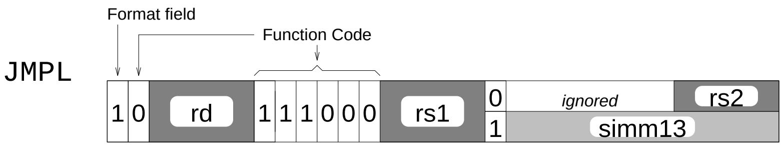
The conditional branch instructions have a shorter range since they need eight bits to specify the branch condition and the annul[4] status.

`BNE` *label*

If $(Z = 0)$ then $nPC' \leftarrow PC + (disp22 * 4)$
$PC' \leftarrow nPC$

[4] $a = 1$ indicates that we have a branch with annul.

- Control Transfer instructions.



This instruction allows a register indirect jump. Since it saves the old program counter it may be used to jump to or return from a subroutine[5].

`JMPL rs1+rs2,rd` $rd' \leftarrow PC$
$PC' \leftarrow nPC$
$nPC' \leftarrow rs1 + rs2$

[5]`JMPL R15+8,R0` performs a subroutine return via the address stored in R15.

- Using an `ADD` instruction we can set the value of a register in the range -4096 to +4095[6].

`ADD R0,simm13,rd` $rd' \leftarrow simm13$

- The `SETHI` instruction enables the setting of the more significant register bits.

`SETHI imm22,rd` $rd < 31 : 10 >' \leftarrow imm22$
$rd < 9 : 0 >' \leftarrow 0$

[6]note that the signed immediate is sign extended before placement on the **s2** bus.

- Manipulation of immediate and displacement values: