Complex Instruction Set Computers

CISC

Successors to the 6809 have continued to follow CISC philosophy.

- More registers
- More instructions
- More powerful instructions
- More addressing modes

Each improvement reduces the number of slow memory accesses made by the processor.

Reduced Instruction Set Computers

RISC

New Philosophy

- Invest effort and chip area in accelerating commonly used instructions.
- We can build a chip with slimmed down instruction set which clocks faster.
  - Common instructions execute faster.
  - Efficient compilers should be easier to write for a simpler instruction set.

Most implementations aim for 1 cycle per instruction although now we see machines with multiple execution units aiming for even greater throughput.

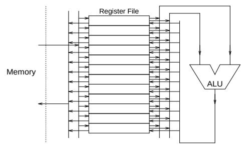
Reduced Instruction Set Computers

SPARC

- Scalable Processor ARChitecture
- Developed by Sun Microsystems
- Open architecture
- Owned by SPARC international
- Used in
  - Sun SPARCstations
  - other computers including CM-5
  - embedded microcontroller systems
  - ECS microprocessor lab

SPARC Architecture

- Register / Register architecture



- Most instructions will either reference three registers or two registers and one immediate.
  - `ADD %20,%17,%5`   `reg5 ← reg20 + reg17`
  - `LD [%11+%12],%5`   `reg5 ← [reg11 + reg12]`
  - `SLL %2,13,%5`   `reg5 ← reg2 >> 13`
- All instructions are 32 bits wide.

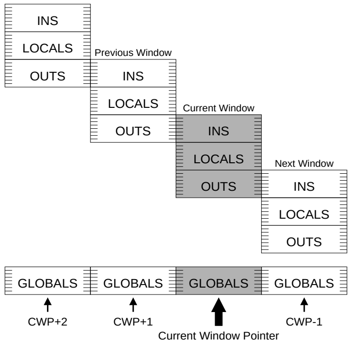
SPARC Architecture

	r15	(o7)	temp		r31	(i7)	return address
	r14	(SP)	stack pointer		r30	(FP)	frame pointer
	r13	(o5)	outgoing param reg 5		r29	(i0)	incoming param reg 5
out	r12	(o4)	outgoing param reg 4	in	r28	(i0)	incoming param reg 4
	r11	(o3)	outgoing param reg 3		r27	(i0)	incoming param reg 3
	r10	(o2)	outgoing param reg 2		r26	(i0)	incoming param reg 2
	r9	(o1)	outgoing param reg 1		r25	(i0)	incoming param reg 1
	r8	(o0)	outgoing param reg 0		r24	(i0)	incoming param reg 0
	r7	(g7)	global 7		r23	(l7)	local 7
	r6	(g6)	global 6		r22	(l6)	local 6
	r5	(g5)	global 5		r21	(l5)	local 5
global	r4	(g4)	global 4	local	r20	(l4)	local 4
	r3	(g3)	global 3		r19	(l3)	local 3
	r2	(g2)	global 2		r18	(l2)	local 2
	r1	(g1)	global 1		r17	(l1)	local 1
	r0	(g0)	0		r16	(l0)	local 0

note that `g0` is a dummy register; it is always zero.

SPARC Architecture

- The 32 registers visible to the program are merely a window on those available.



- The current window pointer determines which register sets appear in this window.

SPARC Architecture

- A subroutine call is normally combined with a context change, giving access to a new window.[1]

```
CALL 18348
                    %o7 ← PC
                    PC ← nPC
                    nPC ← PC + 18348
SAVE
                    CWP ← CWP - 1
                    PC ← nPC
```

- Parameters to be passed to a subroutine are set up using the **OUT** registers which become the **IN** registers of the subroutine.

[1] Due to the wonders of pipelining the `SAVE` is placed after the `CALL` and is executed anyway

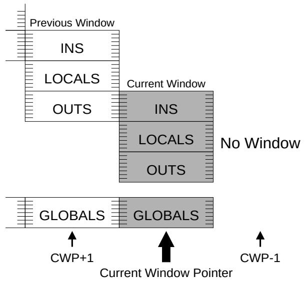
SPARC Architecture

- A return from subroutine uses `JMPL` to indirect via the stored return address and `RESTORE` to recover the old register window.

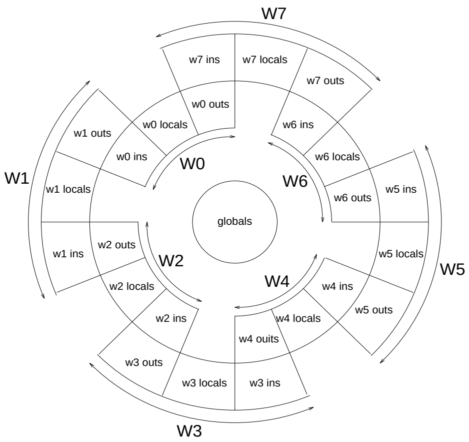
```
JMPL %i7 + 8, %0
                    %0 ← PC
                    PC ← nPC
                    nPC ← %i7 + 8
RESTORE
                    CWP ← CWP + 1
                    PC ← nPC
```

Window Overflow

The Scalable architecture of the SPARC allows the manufacturer to choose how many windows exist on the chip. Since this number is going to be finite, we must allow for the possibility of window overflow.

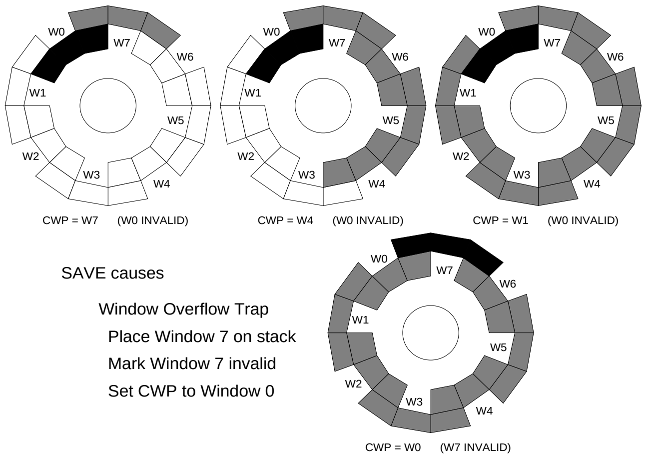


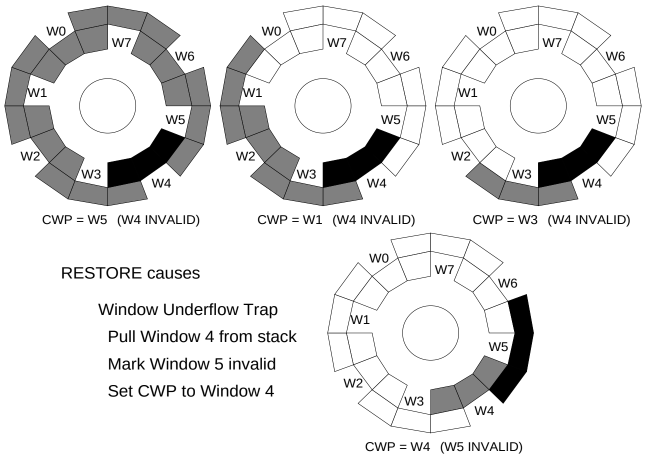
- Windows are arranged in a circle.



Up to seven full windows (including ins and outs) are supported by the eight window circular stack. Initially the Current Window Pointer may be set to W7 and the W0 window marked as invalid. The program may use all windows W7-W1 without any problems.

- A `SAVE` instruction which attempts to allocate the invalid window results in a *window_overflow* trap.
  - The trap routine is responsible for saving the contents of the oldest window to an *old window stack* in memory.
  - The trap routine will also change the Window Invalid Mask to indicate a new invalid window.
- A `RESTORE` instruction which attempts to allocate the invalid window results in a *window_underflow* trap.
  - The trap routine will pull a window from the *old window stack* into the correct register positions.




W0
W7
W6
W1
W5
W2
W3
W4
CWP = W5 (W4 INVALID)
CWP = W1 (W4 INVALID)
CWP = W3 (W4 INVALID)
RESTORE causes
Window Underflow Trap
Pull Window 4 from stack
Mark Window 5 invalid
Set CWP to Window 4
CWP = W4 (W5 INVALID)