S2 Instruction Set

S2 is a simple 32-bit processor for teaching purpose. The choice of the 3-address instruction format is suitable to map to a typical "a = b op c" in a high level language. The instruction set is modelled after modern processors. It includes full integer arithmetic and logic to enable compiling a realistic program into this ISA. Three addressing modes are introduced: displacement, index and immediate. These mode enables a realistic access to data structure such as a local variable in an activation record and array indexing. S2 ISA represents a realistic ISA for a typical 32-bit processor without excess details. This is not "optimal" design. There are plenty of room for improvement.

Instruction format

```
L-format    op:5 rd1:5 ads:22
D-format    op:5 rd1:5 rs2:5 disp:17
X-format    op:5 rd1:5 rs2:5 rs3:5 xop:12
```

(rd dest, rs source, disp sign extended)
Instructions are fixed length at 32-bit. Register set is 32, with R[0] always return zero. The address space is 32-bit, addressing is word. Flags are: Z zero, S sign, C carry, O overflow/underflow.

ISA and opcode encoding

mode: a - absolute, d -displacement, x - index, i - immediate, r - register, r2 - register 2 operands, s - special 1 operand
jump conditional coding in r1: 0 always, 1 eq, 2 neq, 3 lt, 4 le, 5 ge, 6 gt

opcode	op	mode	format	
0	ld	a	L	
1	ld	d	D	
2	ld	i	L	
3	st	a	L	
4	st	d	D	
5	jmp	a	L	r1 as cond
6	jal	a	L	
7	add	i	D	
8	sub	i	D	
9	mul	i	D	
10	div	i	D	
11	and	i	D	
12	or	i	D	
13	xor	i	D	
14..30	undefined			
31	xop	-	X	

xop			
0	add	r	X
1	sub	r	X
2	mul	r	X
3	div	r	X
4	and	r	X
5	or	r	X
6	xor	r	X
7	shl	r2	X
8	shr	r2	X
9	ld	x	X
10	st	x	X
11	jr	s	X use r1
12	trap	s	X use r1 as trap number
13..4095	undefined		

Meaning

format: op dest, source1, source2, r0 always returns zero

```
ld r1 ads          R[r1] = M[ads]
ld r1 #n           R[r1] = n
ld r1 @d r2        R[r1] = M[ d + R[r2] ]
ld r1 +r2 r3       R[r1] = M[ R[r2] + R[r3] ]
st ads r1          M[ads] = R[r1]
st @d r2 r1        M[ d + R[r2] ] = R[r1]
st +r2 r3 r1       M[ R[r2] + R[r3] ] = R[r1]

jmp cond ads       if cond true PC = ads
jal r1 ads         R[r1] = PC; PC = ads
jr  r1             PC = R[r1]
```

jal r1 ads is jump and link, or call to subroutine, saving next pc in r1
jr r1 is jump with register, or return from subroutine

```
add r1 r2 r3     R[r1] = R[r2] + R[r3]
add r1 r2 #n     R[r1] = R[r2] + sign extended n
```

arithmetic: two-complement integer arithmetic
add, sub affect Z,C -- C indicates carry (add) or borrow (sub)
mul, div affect Z,O -- O indicates overflow (mul) or underflow (div) and divide by zero
logic (bitwise) affect Z,S bits

```
and r1 r2 r3     R[r1] = R[r2] bitand R[r3]
and r1 r2 #n     R[r1] = R[r2] bitand sign extended n
or xor  . . .
shl r1 r2        R[r1] = R[r2] shift left one bit
shr r1 r2        R[r1] = R[r2] shift right one bit

trap n r2
```

special instruction, n is in r1-field. Trap is a call to OS functions or to simulator supported functions. The opcode format and assembly language format for S2 follow the tradition dest = source1 op source2 from PDP, VAX and IBM S360. As r0 always is zero, many instructions can be synthesis using r0.

```
or r1 r2 r0          move r1 <- r2
or r1 r0 r0          clear r1
sub r0 r1 r2         compare r1 r2  affects flags
```

To complement a register, xor with 0xFFFFFFFF (-1) can be used.

```
xor r1 r2 #-1        r1 = complement r2
```

7 Feb 2005
Prabhas Chongstitvatana
end