%**r0** is always **zero**

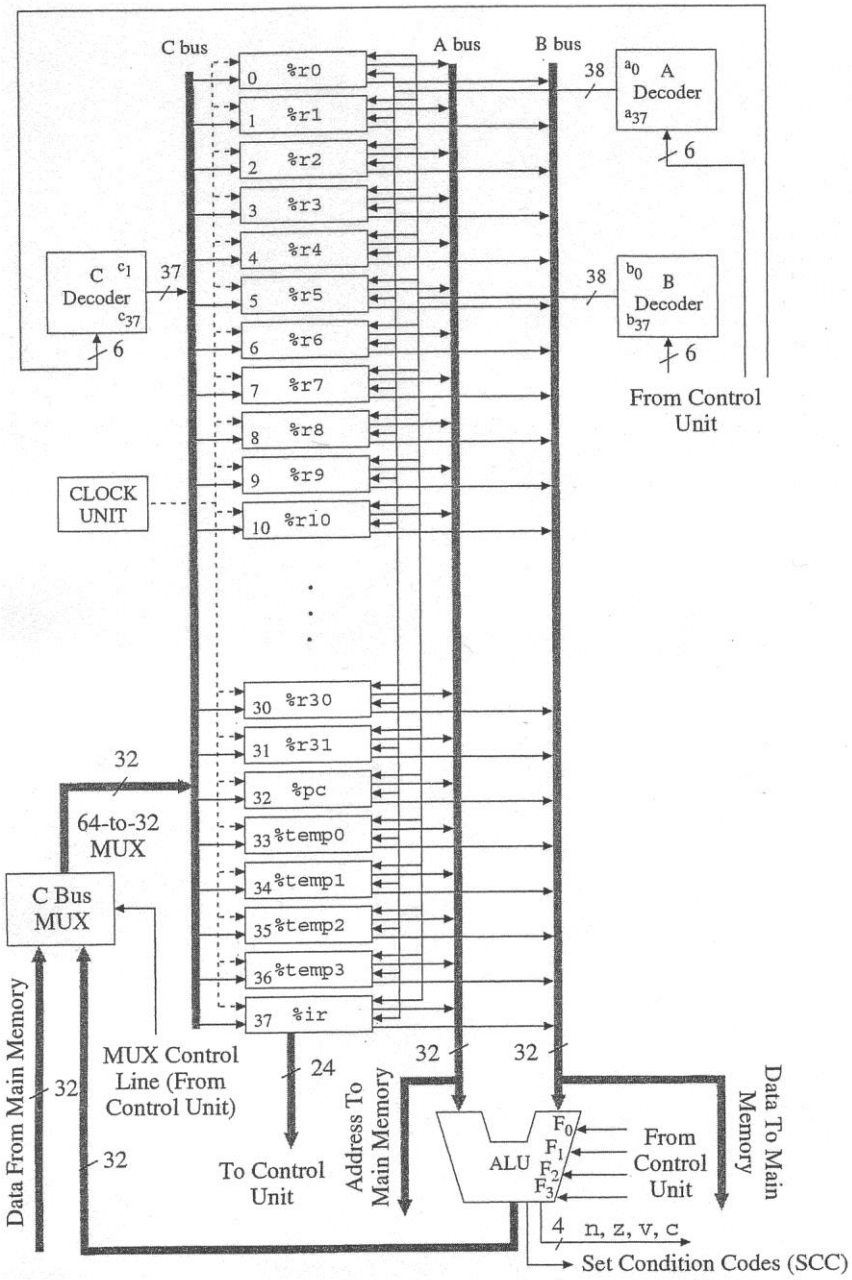
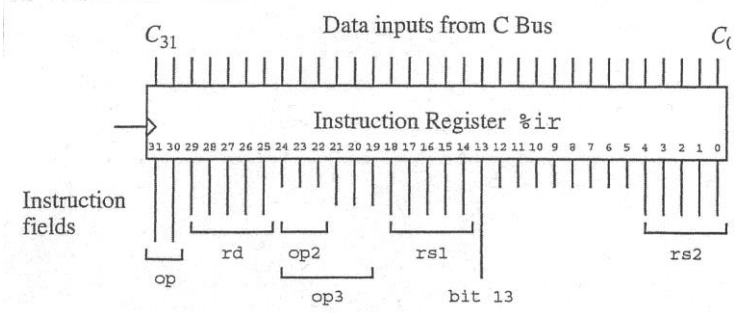


Figure 5-3
The datapath of the ARC.



Mnemonic	Meaning
ld	Load a register from memory
st	Store a register into memory
sethi	Load the 22 most significant bits of a register
andcc	Bitwise logical AND
orcc	Bitwise logical OR
orncc	Bitwise logical OR of rs1 and the inverse of rs2
srl	Shift right (logical)
addcc	Add
call	Call subroutine
jmpl	Jump and link (return from subroutine call)
be	Branch if equal
bneg	Branch if negative
bcs	Branch on carry
bvs	Branch on overflow
ba	Branch always

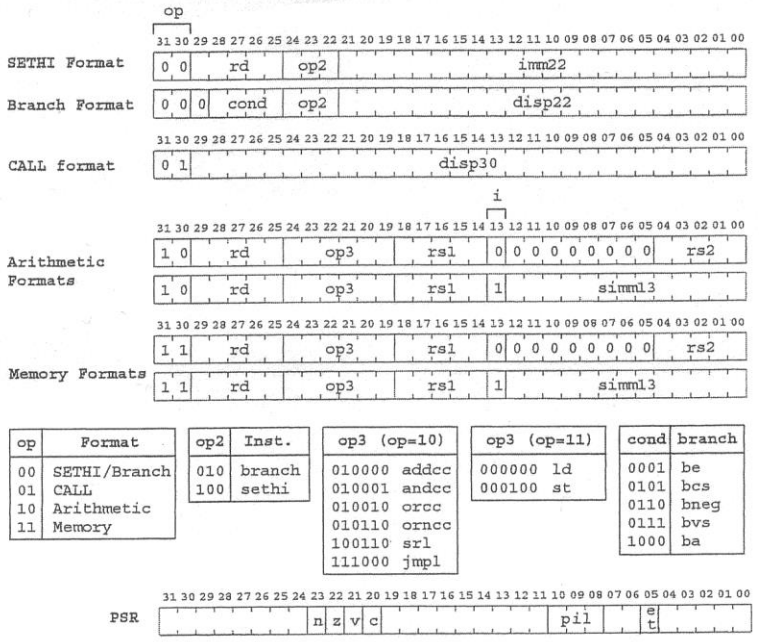


Figure 5-2
Instruction subset and instruction formats for the ARC.

F_3 F_2 F_1 F_0	Operation	Change condition codes	comment
0 0 0 0	ANDCC (A, B)	Yes	
0 0 0 1	ORCC (A, B)	Yes	
0 0 1 0	ORNCC (A, B)	Yes	$A+\overline{B}$ note: + is bitwise logical OR
0 0 1 1	ADDCC (A, B)	Yes	
0 1 0 0	SRL (A. B)	No	
0 1 0 1	AND (A, B)	No	
0 1 1 0	OR (A, B)	No	
0 1 1 1	ORN (A, B)	No	$A+\overline{B}$ note: + is bitwise logical OR
1 0 0 0	ADD (A, B)	No	
1 0 0 1	LSHIFT2 (A)	No	Shift left 2 positions; zeros on the right
1 0 1 0	LSHIFT10 (A)	No	Shift left 10 positions; zeros on the right
1 0 1 1	SIMM13 (A)	No	Lower 13 bits A; zero extended on the left
1 1 0 0	SEXT13 (A)	No	Lower 13 bits A; sign extended on the left
1 1 0 1	INC (A)	No	$A+1$
1 1 1 0	INCPC (A)	No	$A+4$
1 1 1 1	RSHIFT5 (A)	No	Shift right 5 positions with sign extension

Figure 5-4 ARC ALU operations

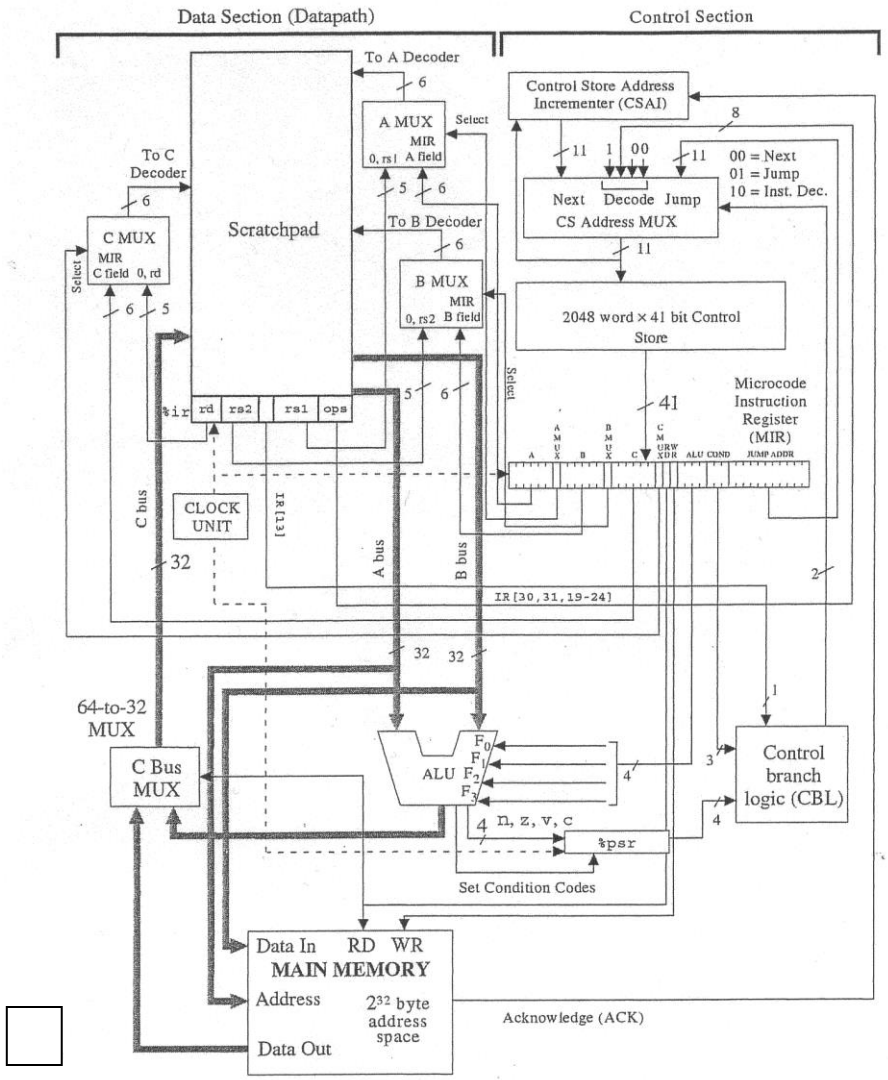


Figure 5-10
The microarchitecture of the ARC.

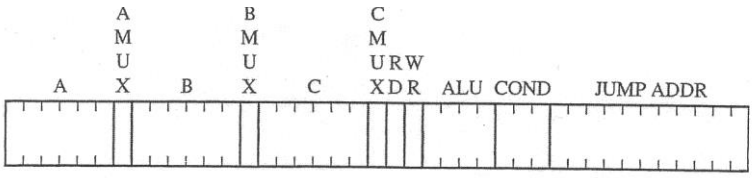


Figure 5-11
The microword format.

C_2	C_1	C_0	Operation
0	0	0	Use NEXT ADDR
0	0	1	Use JUMP ADDR if n = 1
0	1	0	Use JUMP ADDR if z = 1
0	1	1	Use JUMP ADDR if v = 1
1	0	0	Use JUMP ADDR if c = 1
1	0	1	Use JUMP ADDR if IR[13] = 1
1	1	0	Use JUMP ADDR
1	1	1	DECODE

Figure 5-12
Settings for the COND field of the microword.

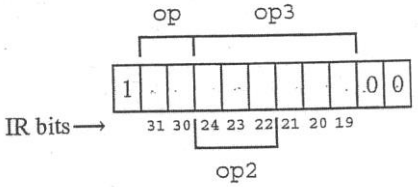


Figure 5-13
DECODE format for a microinstruction address.

Select A Mux
0 → A field MIR
1 → rs1 field %ir

Select B Mux
0 → B field MIR
1 → rs2 field %ir

Select C Mux
0 → C field MIR
1 → rd field %ir

Instruction set examples

	RTL
ld [%r5 + x], %r1	%r1←M[%r5+x]
ld [%r2 + %r4], %r1	%r1←M[%r2+%r4]
ld [x], %r1	%r1←M[x]
ld [%r3], %r1	%r1←M[%r3]
st %r1, [%r5 + x]	M[%r5+x]←%r1 Similar formats as for instruction ld Note: the mapping on the 32 bits instruction format is a little bit different: %r1 is stored in the *rd field* op instruction and %r5 in the *rs1field*.
sethi 0x304f15, %r1	set high 22 bits to $304f15_{16}$ and lower 10 bits to zero
andcc %r1, %r2, %r3	%r3←%r1 AND %r2 + status
andcc %r1, 0xAAB, %r3	%r3←%r1 AND *sign extension lower 13 bits* + update status Instructions **add**, **addcc**, **and**, **andn**, **andncc**, **or**, **orn**, **orncc**, **sub**, **subcc**, **xnor**, **xnorcc**, **xor**, **xorcc** have similar format as andcc Note: only instructions that end with **cc** will update the status bits.
srl %r1, 4, %r3	Shift %r1 right by 4 bits and store the result in %r3. Zeros are copied into the four most significant bits of %r3
srl %r1, %r2, %r3	The shift amount is determined by the lowest 5 bits of %r2 (unsigned representation). Instructions **sll** and **sra** are similar as srl.
call *address*	%r15←PC and PC←address Note: the assembler translates the *address* in the number (*disp30*) of instructions back/forward to the current instruction (relative). %r15←PC and PC←PC+4×*disp30*
jmpl %r1+4,%r4	%r4←PC and PC←%r1+4
be *address*	if equal (Z=1) then PC← PC+4×disp22 else PC←PC+4 The other branch instructions are similar. The assembler translates the *address* in the number (*disp22*) of instructions back/forward to the current instruction

Pseudo-operations recognized by ARCTools

.equ value	Equate a symbol to a value
.begin	begin assembly
.end	end of assembly
.org value	move location counter to value
.dwb value	reserve space for *value* words