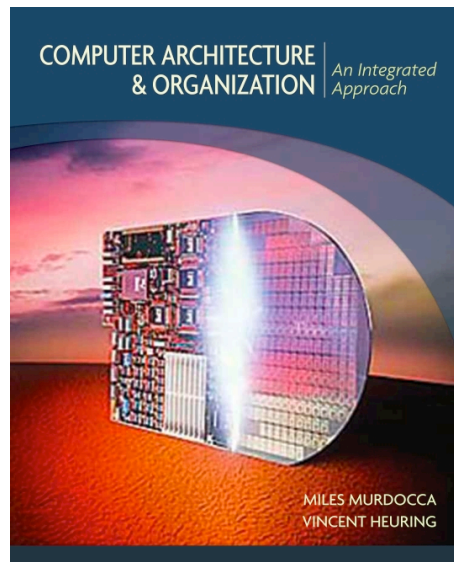


# Computer Architecture and Organization

*Miles Murdocca and Vincent Heuring*

---



## Chapter 5 – Datapath and Control

# Chapter Contents

**5.1 Basics of the Microarchitecture**

**5.2 The Datapath**

**5.3 The Control Section – Microprogrammed**

**5.4 The Control Section – Hardwired**

**5.5 Case Study: The VHDL Hardware Description Language**

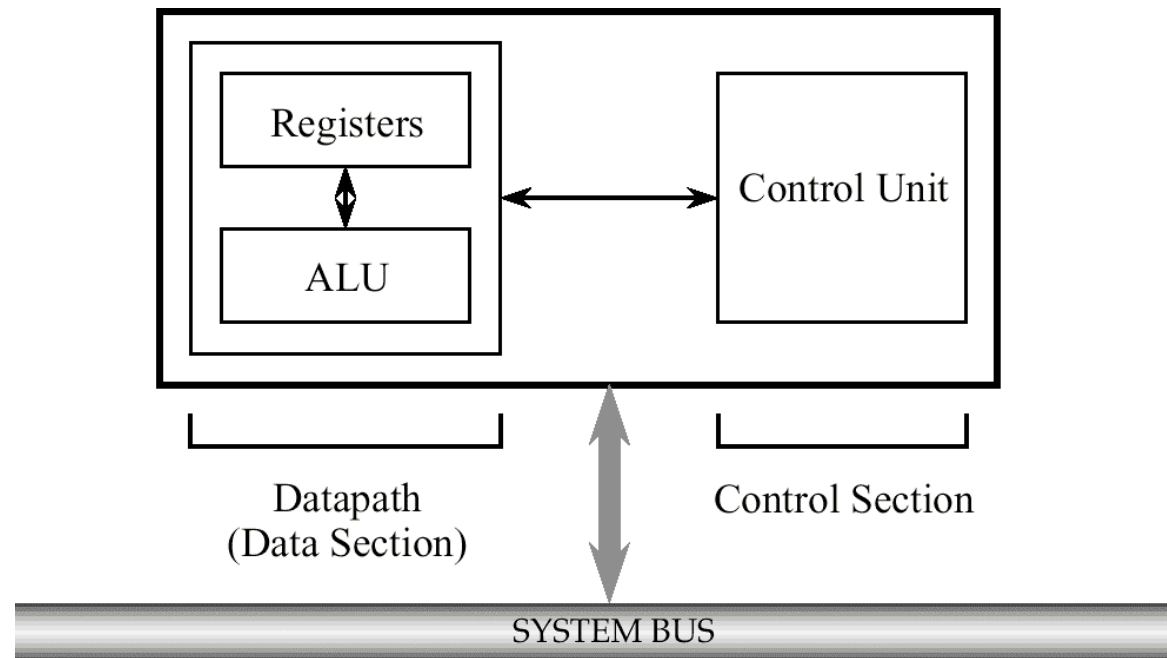
**5.6 Case Study: What Happens when a Computer Boots Up?**

# The Fetch-Execute Cycle

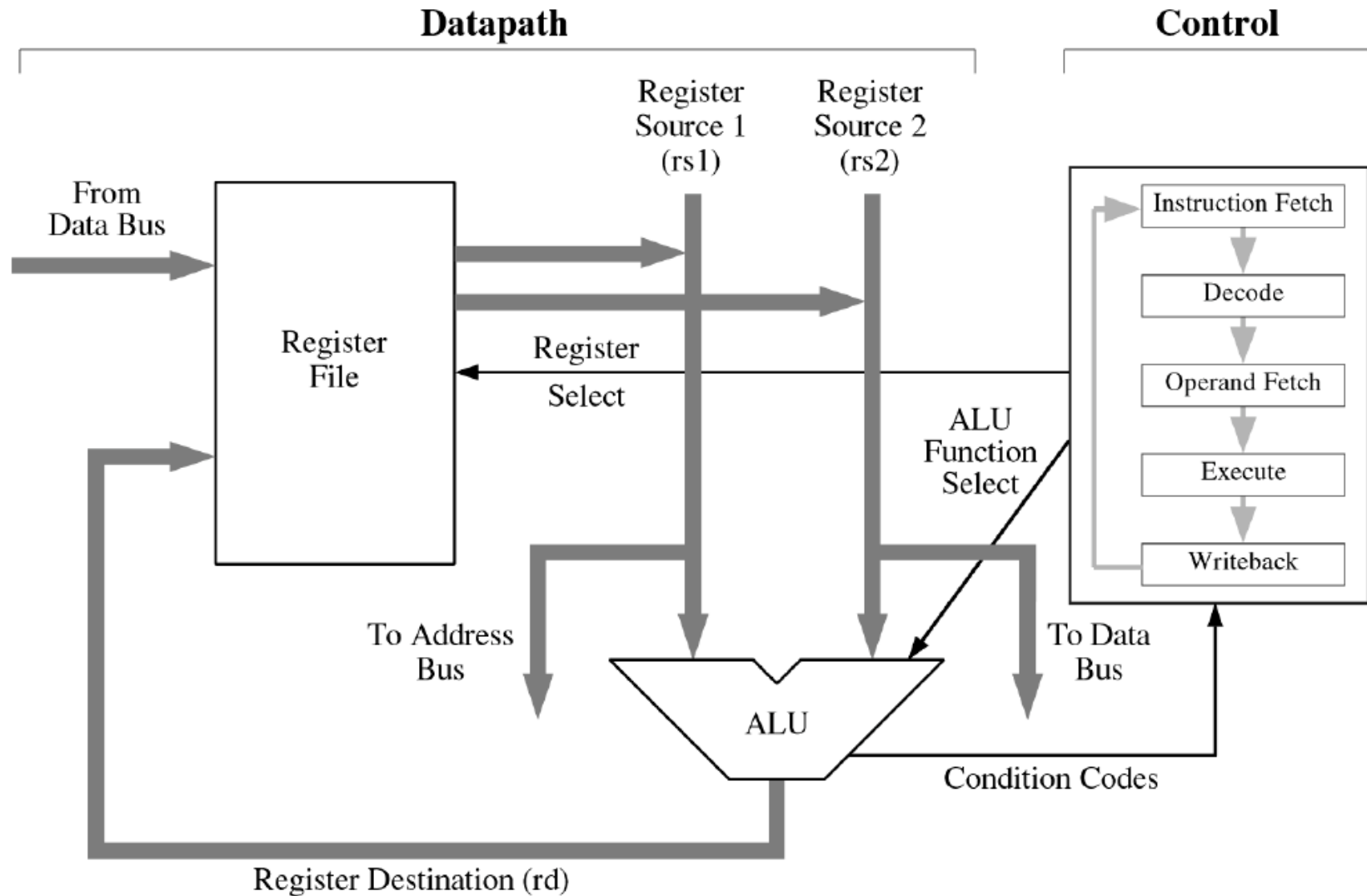
- The steps that the control unit carries out in executing a program are:
  - (1) Fetch the next instruction to be executed from memory.
  - (2) Decode the opcode.
  - (3) Read operand(s) from main memory, if any.
  - (4) Execute the instruction and store results, if any.
  - (5) Go to step 1.

# High Level View of Microarchitecture

- The microarchitecture consists of the control unit and the programmer-visible registers, functional units such as the ALU, and any additional registers that may be required by the control unit.



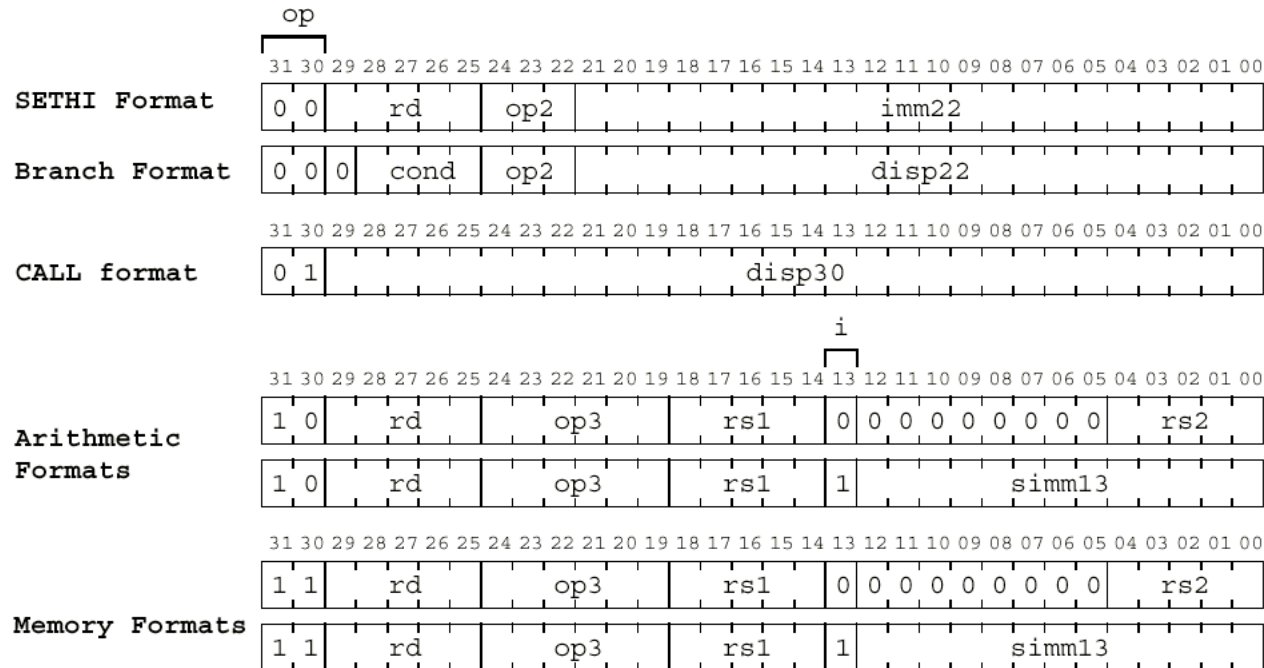
# A More Detailed View



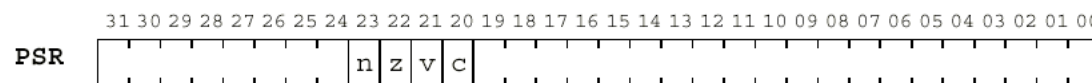
# ARC Instruction Subset

	Mnemonic	Meaning
Memory	<code>ld</code>	Load a register from memory
	<code>st</code>	Store a register into memory
Logic	<code>sethi</code>	Load the 22 most significant bits of a register
	<code>andcc</code>	Bitwise logical AND
	<code>orcc</code>	Bitwise logical OR
	<code>orncc</code>	Bitwise logical NOR
	<code>srl</code>	Shift right (logical)
Arithmetic	<code>addcc</code>	Add
Control	<code>call</code>	Call subroutine
	<code>jmp1</code>	Jump and link (return from subroutine call)
	<code>be</code>	Branch if equal
	<code>bneg</code>	Branch if negative
	<code>bcs</code>	Branch on carry
	<code>bvs</code>	Branch on overflow
	<code>ba</code>	Branch always

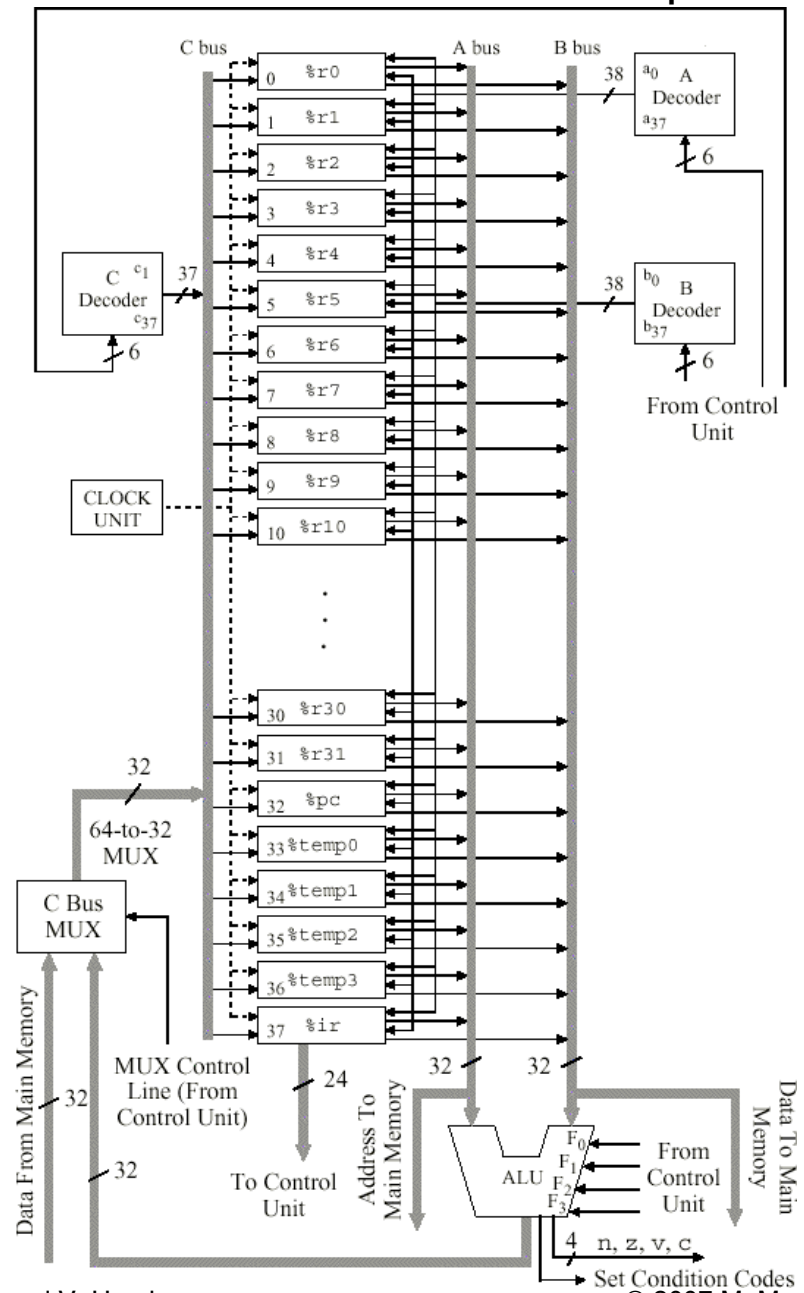
# ARC Instruction Formats



op	Format	op2	Inst.	op3 (op=10)	op3 (op=11)	cond	branch
00	SETHI/Branch	010	branch	010000 addcc	000000 ld	0001	be
01	CALL	100	sethi	010001 andcc	000100 st	0101	bcs
10	Arithmetic			010010 orcc		0110	bneg
11	Memory			010110 orncc		0111	bvs
				100110 srl		1000	ba
				111000 jmp1			



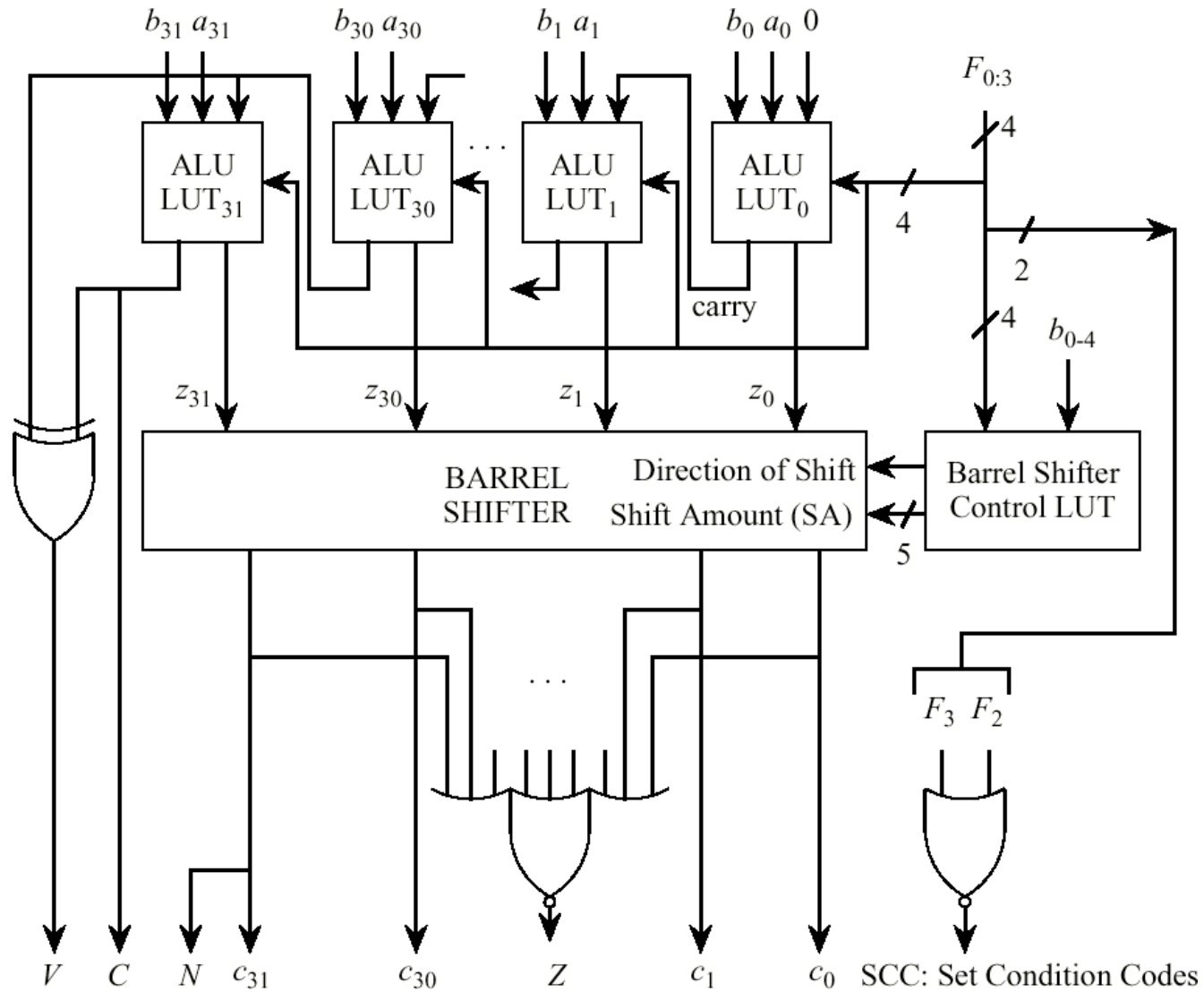
# ARC Datapath



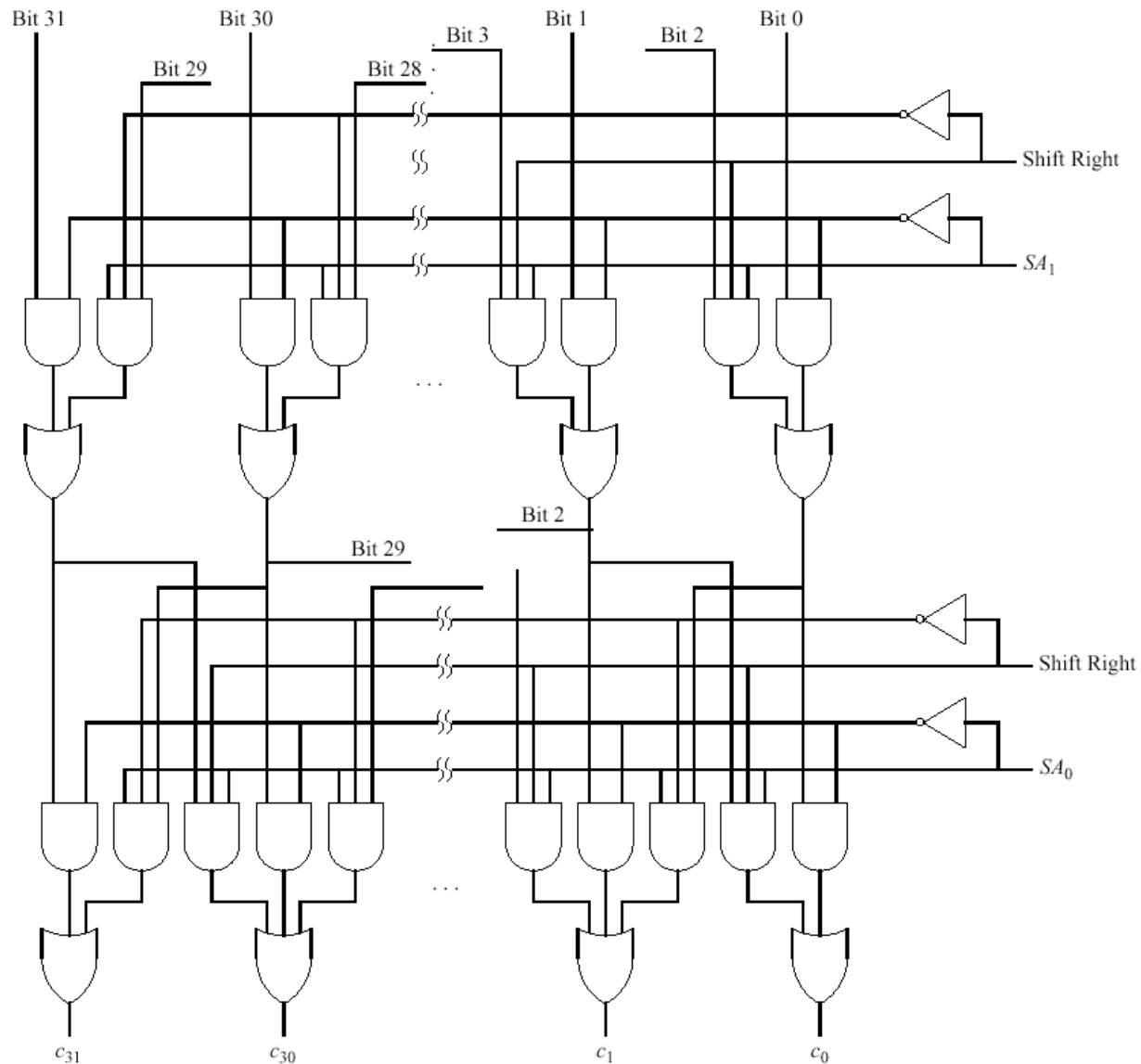
# ARC ALU Operations

$F_3$ $F_2$ $F_1$ $F_0$	Operation	Changes Condition Codes
0 0 0 0	ANDCC (A, B)	yes
0 0 0 1	ORCC (A, B)	yes
0 0 1 0	ORNCC (A, B)	yes
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
1 0 0 0	ADD (A, B)	no
1 0 0 1	LSHIFT2 (A)	no
1 0 1 0	LSHIFT10 (A)	no
1 0 1 1	SIMM13 (A)	no
1 1 0 0	SEXT13 (A)	no
1 1 0 1	INC (A)	no
1 1 1 0	INCPC (A)	no
1 1 1 1	RSHIFT5 (A)	no

# Block Diagram of ALU



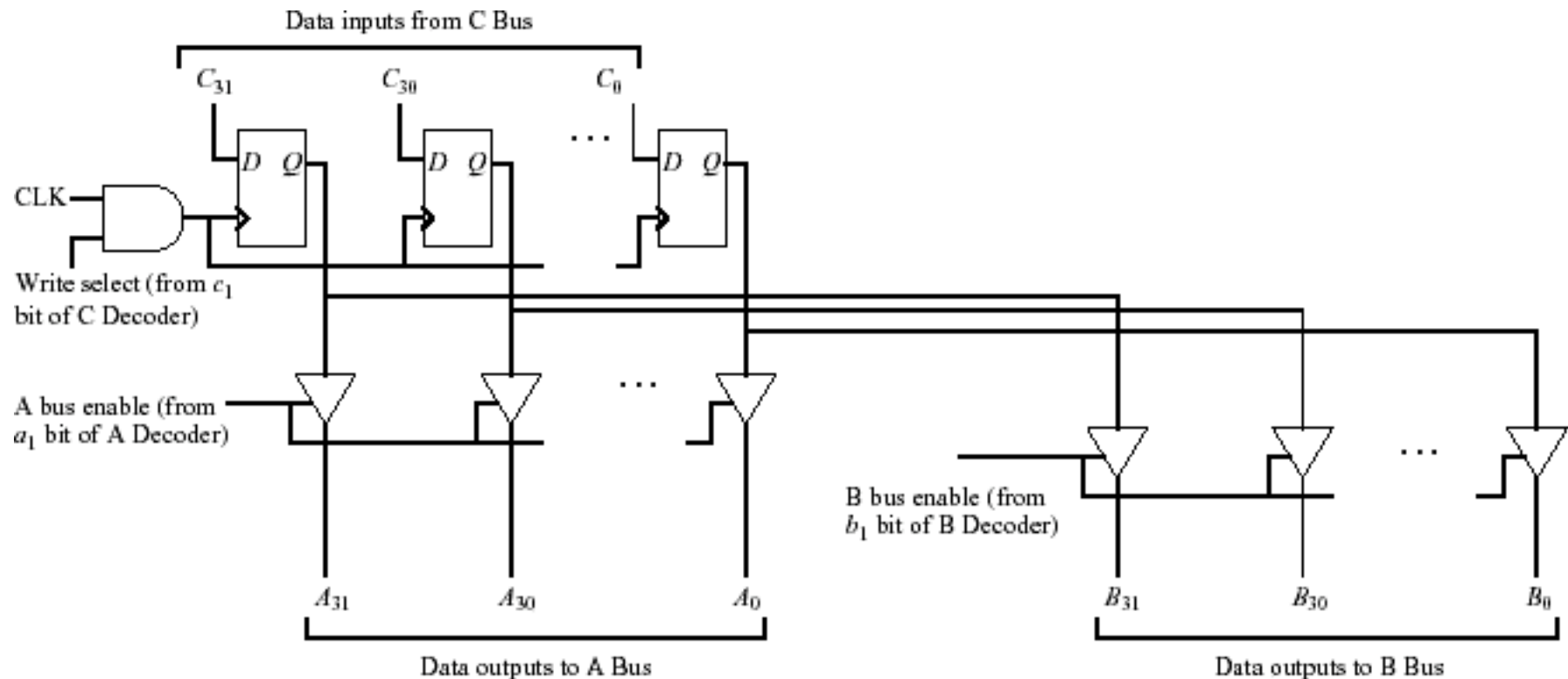
# Gate-Level Layout of Barrel Shifter



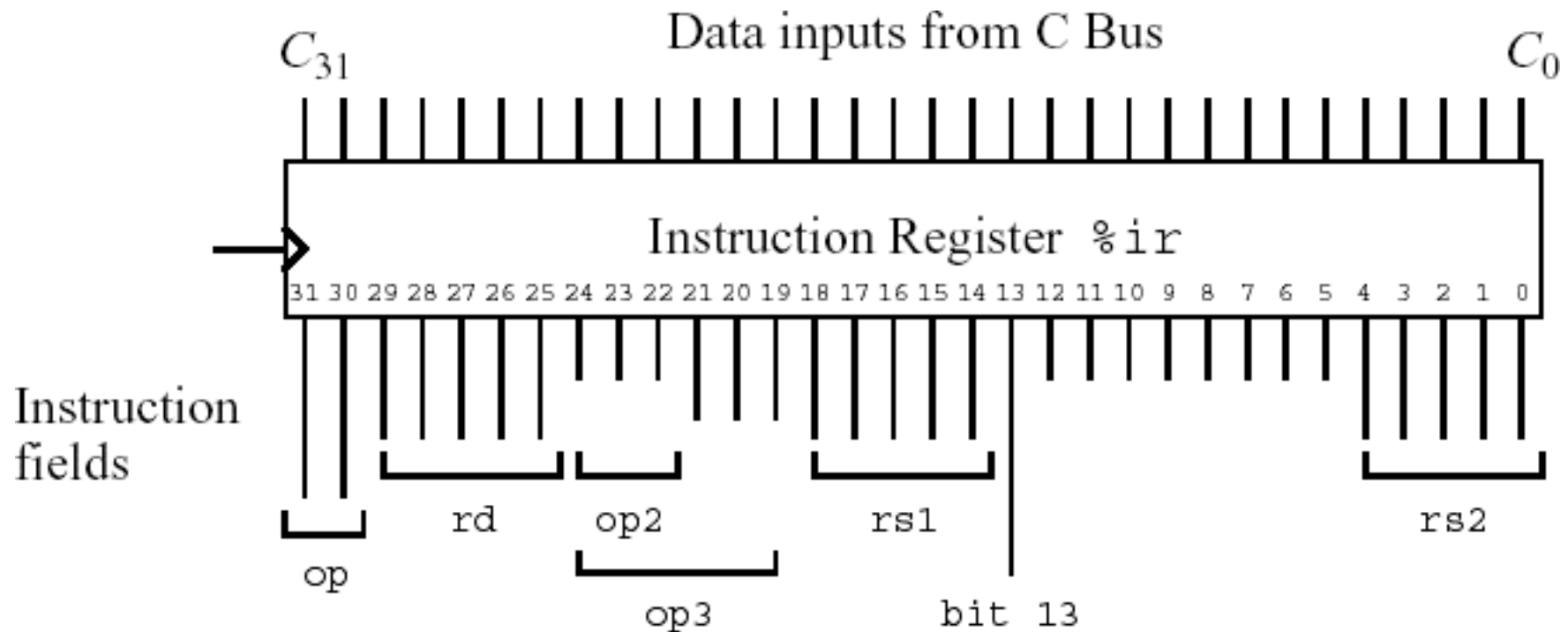
# Truth Table for (Most of the) ALU LUTs

	$F_3$	$F_2$	$F_1$	$F_0$	Carry In	$a_i$	$b_i$	$z_i$	Carry Out
ANDCC	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	1	0	0
	0	0	0	0	0	1	0	0	0
	0	0	0	0	0	1	1	1	0
	0	0	0	0	1	0	0	0	0
	0	0	0	0	1	0	1	0	0
	0	0	0	0	1	1	0	0	0
	0	0	0	0	1	1	1	1	0
ORCC	0	0	0	1	0	0	0	0	0
	0	0	0	1	0	0	1	1	0
	0	0	0	1	0	1	0	1	0
	0	0	0	1	0	1	1	1	0
	0	0	0	1	1	0	0	0	0
	0	0	0	1	1	0	1	1	0
					.				.
				.				.	
				.				.	

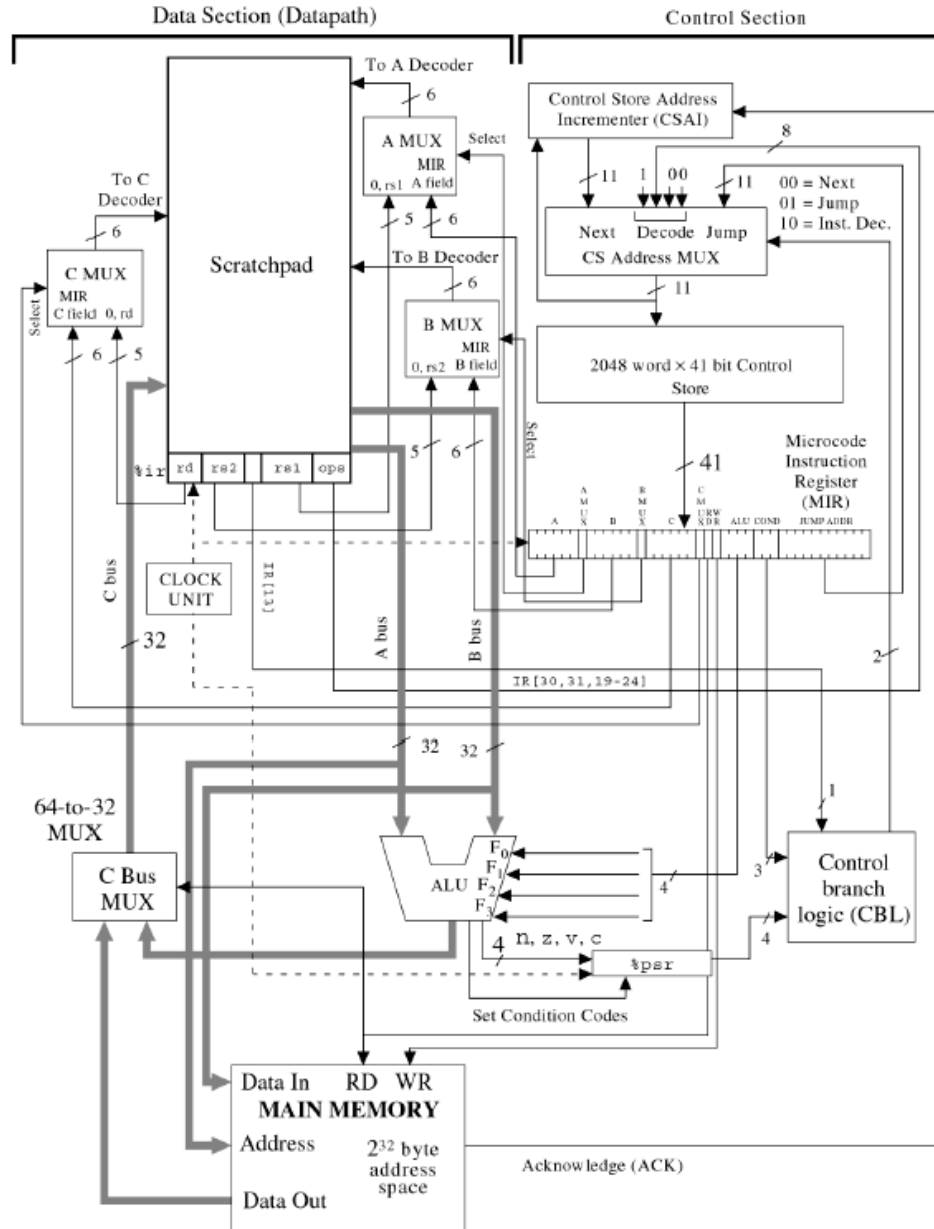
# Design of Register %r1



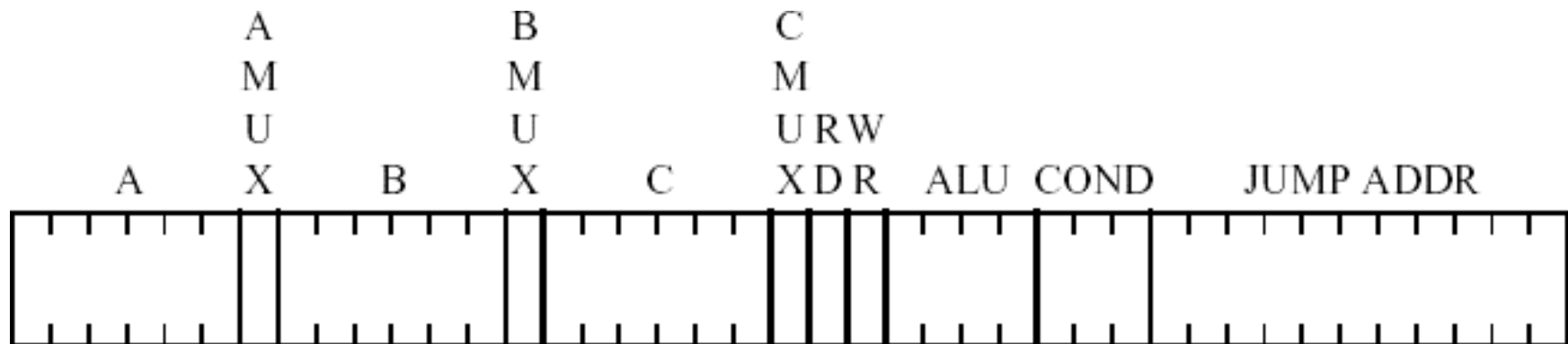
# Outputs to Control Unit from Register %ir



# Microarchitecture of the ARC



# Microword Format

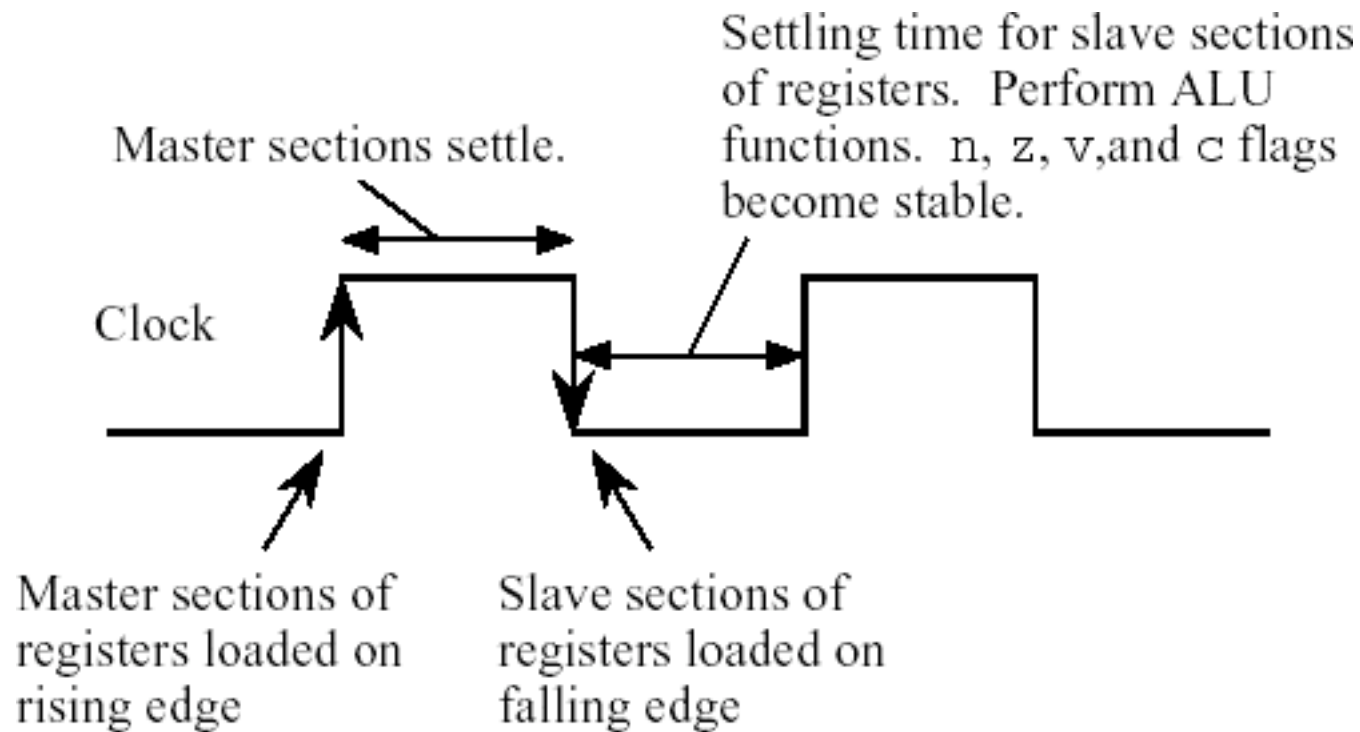


# Settings for the COND Field of the Microword

$C_2$	$C_1$	$C_0$	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



# Timing Relationships for the Registers



# Partial ARC Micro- program

Address	Operation Statements	Comment
0:	$R[ir] \leftarrow \text{AND}(R[pc], R[pc]);$ READ;	/ Read an ARC instruction from main memory
1:	DECODE;	/ 256-way jump according to opcode
	/ <b>sethi</b>	
1152:	$R[rd] \leftarrow \text{LSHIFT10}(ir);$ GOTO 2047;	/ Copy imm22 field to target register
	/ <b>call</b>	
1280:	$R[15] \leftarrow \text{AND}(R[pc], R[pc]);$	/ Save %pc in %r15
1281:	$R[temp0] \leftarrow \text{ADD}(R[ir], R[ir]);$	/ Shift disp30 field left
1282:	$R[temp0] \leftarrow \text{ADD}(R[temp0], R[temp0]);$	/ Shift again
1283:	$R[pc] \leftarrow \text{ADD}(R[pc], R[temp0]);$	/ Jump to subroutine
	GOTO 0;	
	/ <b>addcc</b>	
1600:	IF R[IR[13]] THEN GOTO 1602;	/ Is second source operand immediate?
1601:	$R[rd] \leftarrow \text{ADDCC}(R[rs1], R[rs2]);$	/ Perform ADDCC on register sources
	GOTO 2047;	
1602:	$R[temp0] \leftarrow \text{SEXT13}(R[ir]);$	/ Get sign extended simm13 field
1603:	$R[rd] \leftarrow \text{ADDCC}(R[rs1], R[temp0]);$	/ Perform ADDCC on register/simm13 sources
	GOTO 2047;	
	/ <b>andcc</b>	
1604:	IF R[IR[13]] THEN GOTO 1606;	/ Is second source operand immediate?
1605:	$R[rd] \leftarrow \text{ANDCC}(R[rs1], R[rs2]);$	/ Perform ANDCC on register sources
	GOTO 2047;	
1606:	$R[temp0] \leftarrow \text{SIMM13}(R[ir]);$	/ Get simm13 field
1607:	$R[rd] \leftarrow \text{ANDCC}(R[rs1], R[temp0]);$	/ Perform ANDCC on register/simm13 sources
	GOTO 2047;	
	/ <b>orcc</b>	
1608:	IF R[IR[13]] THEN GOTO 1610;	/ Is second source operand immediate?
1609:	$R[rd] \leftarrow \text{ORCC}(R[rs1], R[rs2]);$	/ Perform ORCC on register sources
	GOTO 2047;	
1610:	$R[temp0] \leftarrow \text{SIMM13}(R[ir]);$	/ Get simm13 field
1611:	$R[rd] \leftarrow \text{ORCC}(R[rs1], R[temp0]);$	/ Perform ORCC on register/simm13 sources
	GOTO 2047;	
	/ <b>orncc</b>	
1624:	IF R[IR[13]] THEN GOTO 1626;	/ Is second source operand immediate?
1625:	$R[rd] \leftarrow \text{NORCC}(R[rs1], R[rs2]);$	/ Perform ORNCC on register sources
	GOTO 2047;	
1626:	$R[temp0] \leftarrow \text{SIMM13}(R[ir]);$	/ Get simm13 field
1627:	$R[rd] \leftarrow \text{NORCC}(R[rs1], R[temp0]);$	/ Perform NORCC on register/simm13 sources
	GOTO 2047;	
	/ <b>srl</b>	
1688:	IF R[IR[13]] THEN GOTO 1690;	/ Is second source operand immediate?
1689:	$R[rd] \leftarrow \text{SRL}(R[rs1], R[rs2]);$	/ Perform SRL on register sources
	GOTO 2047;	
1690:	$R[temp0] \leftarrow \text{SIMM13}(R[ir]);$	/ Get simm13 field
1691:	$R[rd] \leftarrow \text{SRL}(R[rs1], R[temp0]);$	/ Perform SRL on register/simm13 sources
	GOTO 2047;	
	/ <b>jmp1</b>	
1760:	IF R[IR[13]] THEN GOTO 1762;	/ Is second source operand immediate?
1761:	$R[pc] \leftarrow \text{ADD}(R[rs1], R[rs2]);$	/ Perform ADD on register sources
	GOTO 0;	

# Partial ARC Microprogram (cont')

```

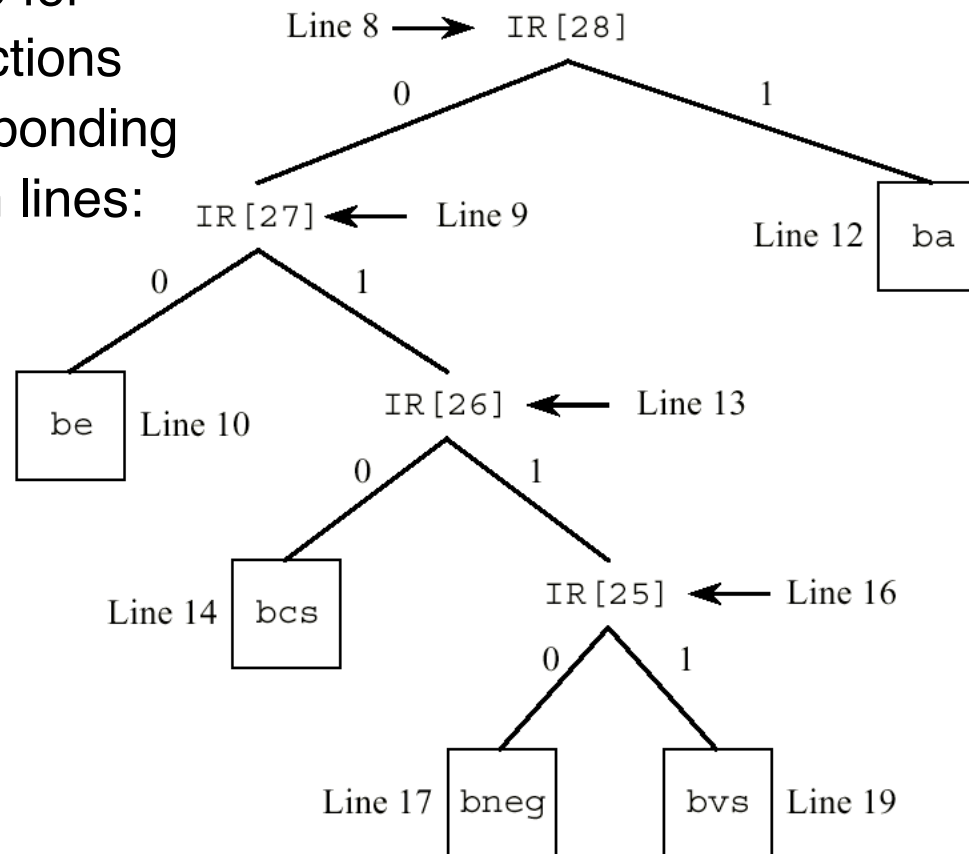
1762: R[temp0] ← SEXT13(R[ir]);           / Get sign extended simm13 field
1763: R[pc] ← ADD(R[rs1],R[temp0]);       / Perform ADD on register/simm13 sources
      GOTO 0;
      / ld
1792: R[temp0] ← ADD(R[rs1],R[rs2]);     / Compute source address
      IF R[IR[13]] THEN GOTO 1794;
1793: R[rd] ← AND(R[temp0],R[temp0]);    / Place source address on A bus
      READ; GOTO 2047;
1794: R[temp0] ← SEXT13(R[ir]);           / Get simm13 field for source address
1795: R[temp0] ← ADD(R[rs1],R[temp0]);   / Compute source address
      GOTO 1793;
      / st
1808: R[temp0] ← ADD(R[rs1],R[rs2]);     / Compute destination address
      IF R[IR[13]] THEN GOTO 1810;
1809: R[ir] ← RSHIFTS(R[ir]); GOTO 40;   / Move rd field into position of rs2 field
      / by shifting to the right by 25 bits.
      40: R[ir] ← RSHIFTS(R[ir]);
      41: R[ir] ← RSHIFTS(R[ir]);
      42: R[ir] ← RSHIFTS(R[ir]);
      43: R[ir] ← RSHIFTS(R[ir]);
      44: R[0] ← AND(R[temp0], R[rs2]);    / Place destination address on A bus and
      WRITE; GOTO 2047;                 / place operand on B bus
1810: R[temp0] ← SEXT13(R[ir]);           / Get simm13 field for destination address
1811: R[temp0] ← ADD(R[rs1],R[temp0]);   / Compute destination address
      GOTO 1809;
      / Branch instructions: ba, be, bcs, bvs, bneg
1088: GOTO 2;                             / Decoding tree for branches
      2: R[temp0] ← LSHIFT10(R[ir]);      / Sign extend the 22 LSB's of %temp0
      3: R[temp0] ← RSHIFTS(R[temp0]);    / by shifting left 10 bits, then right 10
      4: R[temp0] ← RSHIFTS(R[temp0]);    / bits. RSHIFTS does sign extension.
      5: R[ir] ← RSHIFTS(R[ir]);          / Move COND field to IR[13] by
      6: R[ir] ← RSHIFTS(R[ir]);          / applying RSHIFTS three times. (The
      7: R[ir] ← RSHIFTS(R[ir]);          / sign extension is inconsequential.)
      8: IF R[IR[13]] THEN GOTO 12;       / Is it ba?
      R[ir] ← ADD(R[ir],R[ir]);
      9: IF R[IR[13]] THEN GOTO 13;       / Is it not be?
      R[ir] ← ADD(R[ir],R[ir]);
      10: IF Z THEN GOTO 12;              / Execute be
      R[ir] ← ADD(R[ir],R[ir]);
      11: GOTO 2047;                       / Branch for be not taken
      12: R[pc] ← ADD(R[pc],R[temp0]);    / Branch is taken
      GOTO 0;
      13: IF R[IR[13]] THEN GOTO 16;     / Is it bcs?
      R[ir] ← ADD(R[ir],R[ir]);
      14: IF C THEN GOTO 12;              / Execute bcs
      15: GOTO 2047;                       / Branch for bcs not taken
      16: IF R[IR[13]] THEN GOTO 19;     / Is it bvs?
      17: IF N THEN GOTO 12;              / Execute bneg
      18: GOTO 2047;                       / Branch for bneg not taken
      19: IF V THEN GOTO 12;              / Execute bvs
      20: GOTO 2047;                       / Branch for bvs not taken
2047: R[pc] ← INCPC(R[pc]); GOTO 0;     / Increment %pc and start over

```



# Branch Decoding

- Decoding tree for branch instructions shows corresponding microprogram lines:



cond				branch
28	27	26	25	
0	0	0	1	be
0	1	0	1	bcs
0	1	1	0	bneg
0	1	1	1	bvs
1	0	0	0	ba





# Example: Add the subcc Instruction

- Consider adding instruction `subcc` (subtract) to the ARC instruction set. `subcc` uses the Arithmetic format and `op3 = 001100`.

```

1584: R[temp0] ← SEXT13(R[ir]);           / Extract rs2 operand
      IF IR[13] THEN GOTO 1586;           / Is second source immediate?
1585: R[temp0] ← R[rs2];                 / Extract sign extended immediate operand
1586: R[temp0] ← NOR(R[temp0], R[0]);     / Form one's complement of subtrahend
1587: R[temp0] ← INC(R[temp0]); GOTO 1603; / Form two's complement of subtrahend

```

	A	X	B	X	C	XDR	ALU	COND	JUMP	ADDR
1584	100101	00000000	0	100001	000	1100	101	110001	10010	
1585	000000	00000000	1	100001	000	1000	000	000000	000000	
1586	100001	00000000	0	100001	000	0111	000	000000	000000	
1587	100001	00000000	0	100001	000	1101	110	110010	000011	

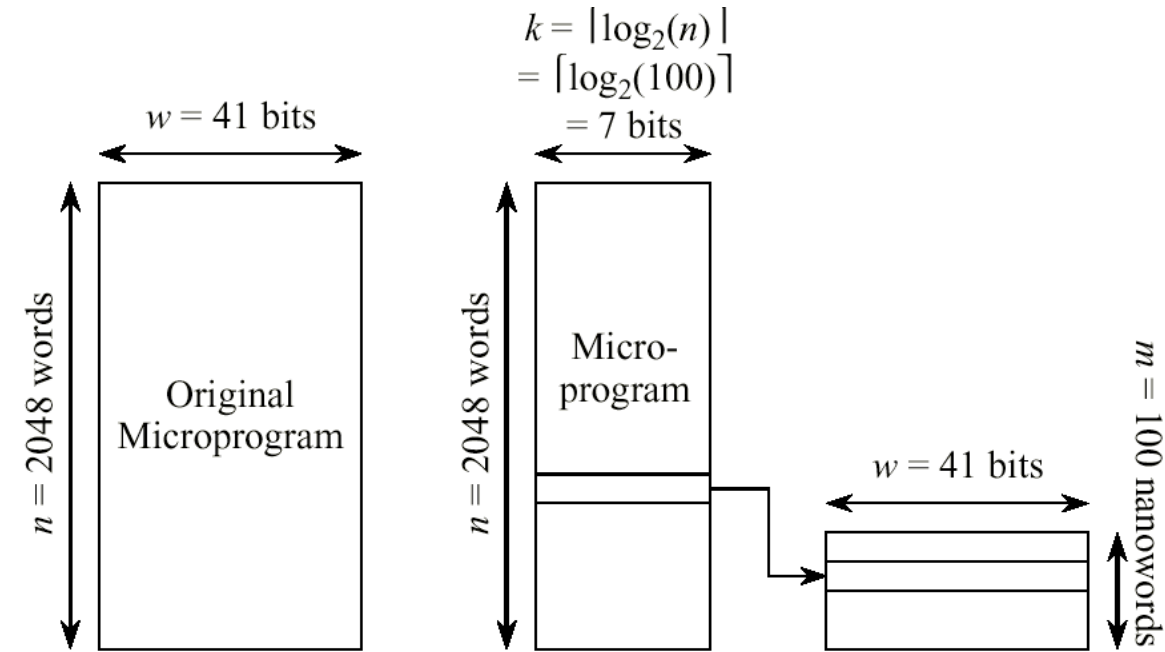
# Branch Table

- A branch table for trap handlers and interrupt service routines:

Address	Contents	Trap Handler
	⋮	
60	JUMP TO 2000	Illegal instruction
64	JUMP TO 3000	Overflow
68	JUMP TO 3600	Underflow
72	JUMP TO 5224	Zerodivide
76	JUMP TO 4180	Disk
80	JUMP TO 5364	Printer
84	JUMP TO 5908	TTY
88	JUMP TO 6048	Timer
	⋮	

# Microprogramming vs. Nanoprogramming

- (a) Microprogramming,  
(b) nanoprogramming.



$$\text{Total Area} = n \times w = 2048 \times 41 = 83,968 \text{ bits}$$

$$\text{Microprogram Area} = n \times k = 2048 \times 7 = 14,336 \text{ bits}$$

$$\text{Nanoprogram Area} = m \times w = 100 \times 41 = 4100 \text{ bits}$$

$$\text{Total Area} = 14,336 + 4100 = 18,436 \text{ bits}$$

(a)

(b)

# Hardware Description Language

- HDL sequence for a resettable modulo 4 counter.

```

Preamble {
MODULE: MOD_4_COUNTER.
INPUTS: x.
OUTPUTS: Z[2].
MEMORY:

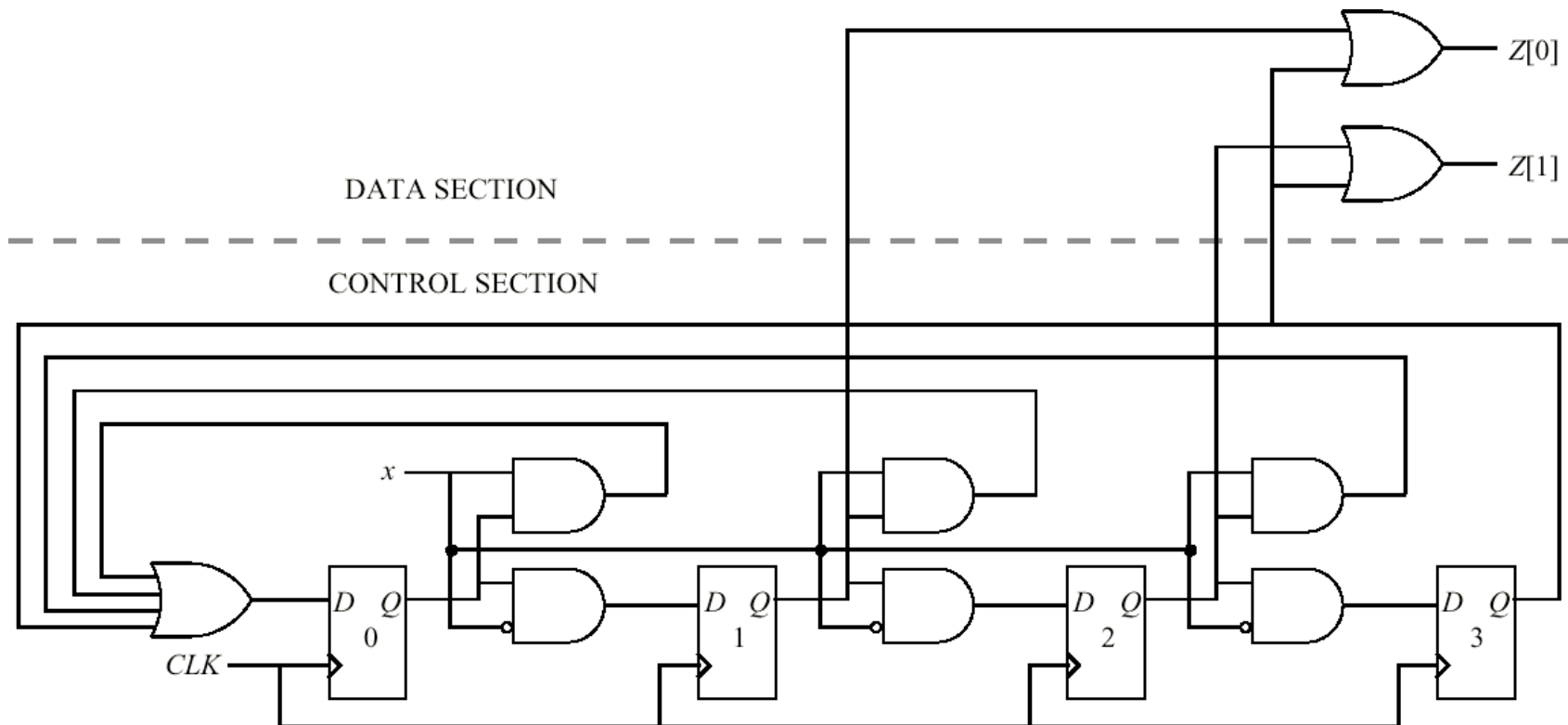
Statements {
0: Z ← 0, 0;
   GOTO {0 CONDITIONED ON x,
         1 CONDITIONED ON  $\bar{x}$ }.
1: Z ← 0, 1;
   GOTO {0 CONDITIONED ON x,
         2 CONDITIONED ON  $\bar{x}$ }.
2: Z ← 1, 0;
   GOTO {0 CONDITIONED ON x,
         3 CONDITIONED ON  $\bar{x}$ }.
3: Z ← 1, 1;
   GOTO 0.

Epilogue {
END SEQUENCE.
END MOD_4_COUNTER.

```

# Circuit Derived from HDL

- Logic design for a modulo 4 counter described in HDL.



# HDL for ARC

- HDL description of the ARC control unit.

```

MODULE: ARC_CONTROL_UNIT.
INPUTS:
OUTPUTS: C, N, V, Z. ! These are set by the ALU
MEMORY: R[16][32], pc[32], ir[32], temp0[32], temp1[32], temp2[32],
        temp3[32].

0: ir ← AND(pc, pc); Read ← 1;           ! Instruction fetch
   ! Decode op field
1: GOTO {2 CONDITIONED ON ir[31]×ir[30], ! Branch/sethi format: op=00
      4 CONDITIONED ON ir[31]×ir[30], ! Call format: op=01
      8 CONDITIONED ON ir[31]×ir[30], ! Arithmetic format: op=10
     10 CONDITIONED ON ir[31]×ir[30]}. ! Memory format: op=11
   ! Decode op2 field
2: GOTO 19 CONDITIONED ON ir[24].       ! Goto 19 if Branch format
3: R[rd] ← ir[imm22];                   ! sethi
   GOTO 20.
4: R[15] ← AND(pc, pc).                  ! call: save pc in register 15
5: temp0 ← ADD(ir, ir).                  ! Shift disp30 field left
6: temp0 ← ADD(ir, ir).                  ! Shift again
7: pc ← ADD(pc, temp0); GOTO 0.          ! Jump to subroutine
   ! Get second source operand into temp0 for Arithmetic format
8: temp0 ← { SEXT13(ir) CONDITIONED ON ir[13]×NOR(ir[19:22]), ! addcc
  R[rs2] CONDITIONED ON ir[13]×NOR(ir[19:22]),           ! addcc
  SIMM13(ir) CONDITIONED ON ir[13]×OR(ir[19:22]),       ! Remaining
  R[rs2] CONDITIONED ON ir[13]×OR(ir[19:22])}. ! Arithmetic instructions
   ! Decode op3 field for Arithmetic format
9: R[rd] ← {
  ADDCC(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 010000), ! addcc
  ANDCC(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 010001), ! andcc
  ORCC(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 010010), ! orcc
  NORCC(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 010110), ! orncc
  SRL(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 100110), ! srl
  ADD(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 111000)}; ! jmpl
   GOTO 20.
   ! Get second source operand into temp0 for Memory format
10: temp0 ← {SEXT13(ir) CONDITIONED ON ir[13],
  R[rs2] CONDITIONED ON ir[13]}.
11: temp0 ← ADD(R[rs1], temp0).
   ! Decode op3 field for Memory format
   GOTO {12 CONDITIONED ON ir[21], ! ld
      13 CONDITIONED ON ir[21]}. ! st
12: R[rd] ← AND(temp0, temp0); Read ← 1; GOTO 20.
13: ir ← RSHIFT5(ir).

```

# HDL for ARC (cont')

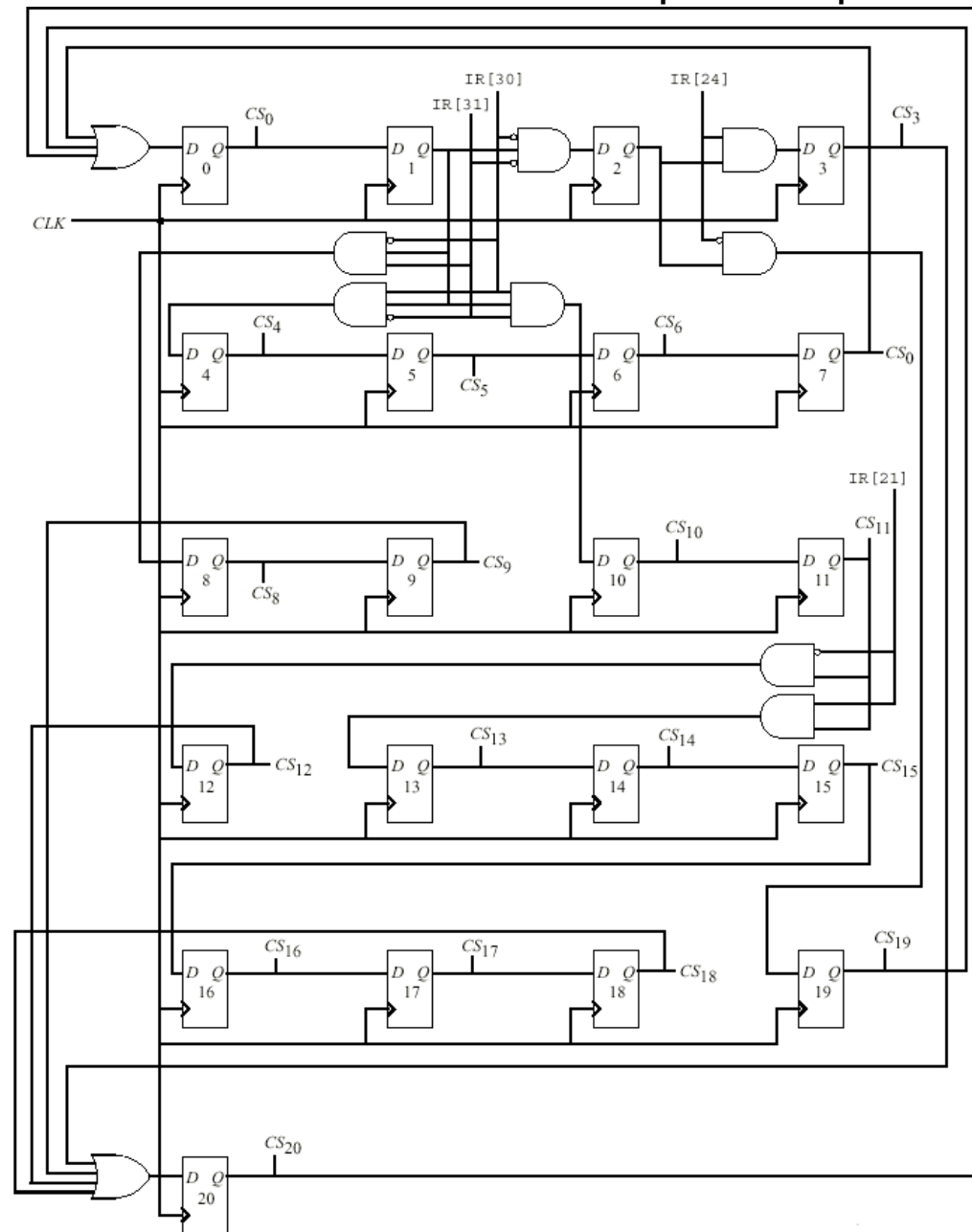
```

14: ir ← RSHIFT5(ir).
15: ir ← RSHIFT5(ir).
16: ir ← RSHIFT5(ir).
17: ir ← RSHIFT5(ir).
18: r0 ← AND(temp0, R[rs2]); Write ← 1; GOTO 20.
19: pc ← { ! Branch instructions
      ADD(pc, temp0) CONDITIONED ON  $\overline{ir[28]} + \overline{ir[28]} \times ir[27] \times Z +$ 
       $\overline{ir[28]} \times ir[27] \times ir[26] \times C + \overline{ir[28]} \times ir[27] \times ir[26] \times ir[25] \times N +$ 
       $\overline{ir[28]} \times ir[27] \times ir[26] \times ir[25] \times V,$ 
      INCPC(pc) CONDITIONED ON  $\overline{ir[28]} \times ir[27] \times Z +$ 
       $\overline{ir[28]} \times ir[27] \times ir[26] \times C + \overline{ir[28]} \times ir[27] \times ir[26] \times ir[25] \times N +$ 
       $\overline{ir[28]} \times ir[27] \times ir[26] \times ir[25] \times V$ };
      GOTO 0.
20: pc ← INCPC(pc); GOTO 0.
END SEQUENCE.
END ARC_CONTROL_UNIT.

```

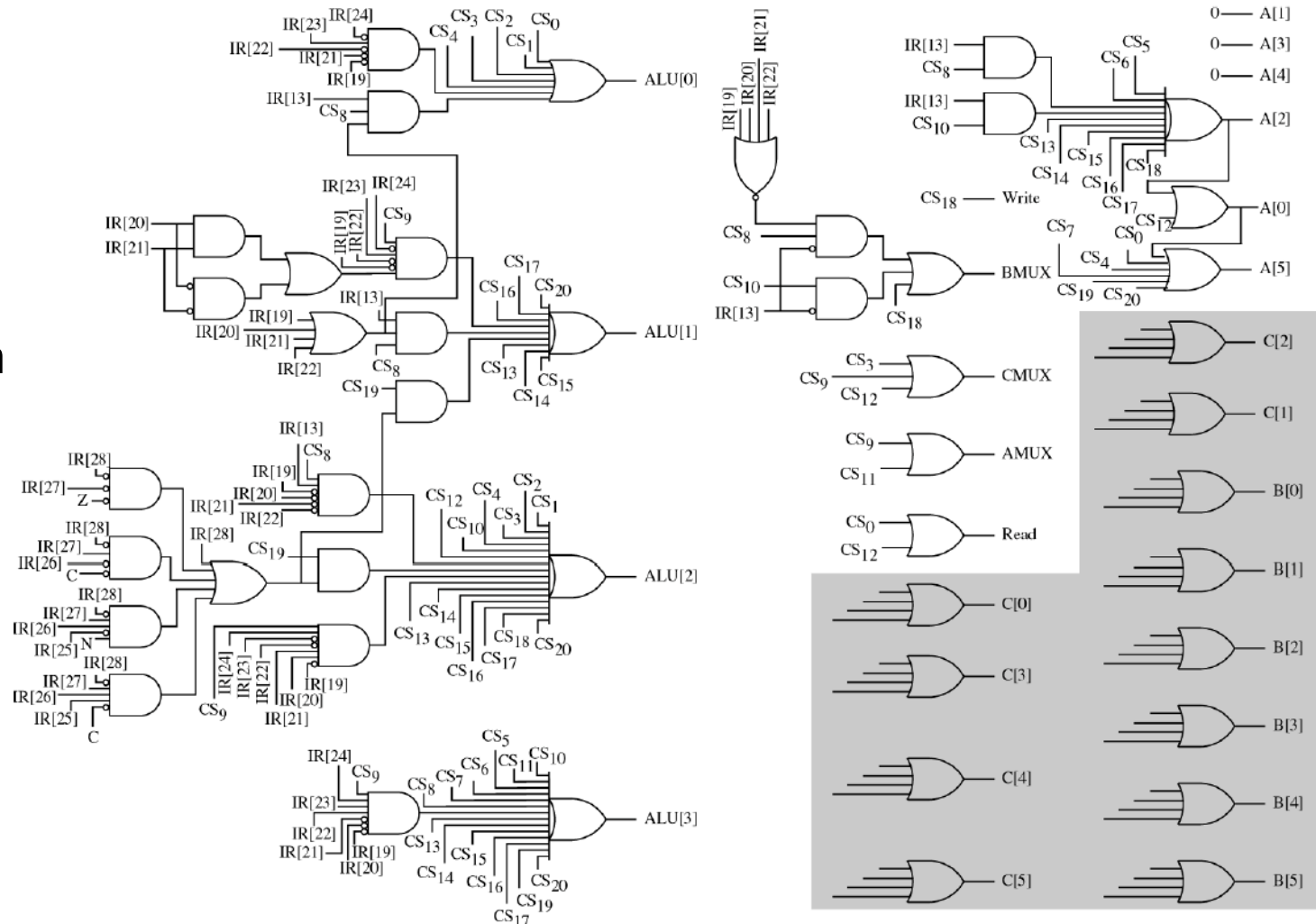
# HDL ARC Circuit

- The hardwired control section of the ARC: generation of the control signals.



# HDL ARC Circuit (cont')

- Hardwired control section of the ARC: signals from the data section of the control unit to the datapath.

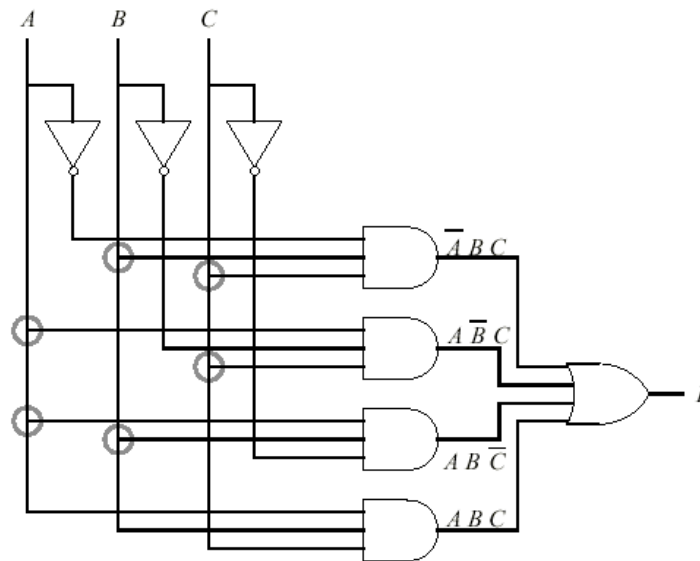


# Case Study: The VHDL Hardware Description Language

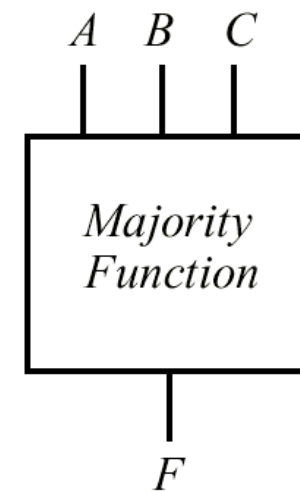
- The majority function. a) truth table, b) AND-OR implementation, c) black box representation.

Minterm Index	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

a)



b)



c)

# VHDL Specification

*Interface specification for the majority component*

```
-- Interface
entity MAJORITY is
    port
        (A_IN, B_IN, C_IN: in BIT
         F_OUT: out BIT);
end MAJORITY;
```

*Behavioral model for the majority component*

```
-- Body
architecture LOGIC_SPEC of MAJORITY is
begin
    -- compute the output using a Boolean expression
    F_OUT <= (not A_IN and B_IN and C_IN) or
              (A_IN and not B_IN and C_IN) or
              (A_IN and B_IN and not C_IN) or
              (A_IN and B_IN and C_IN) after 4 ns;
end LOGIC_SPEC;
```

# VHDL Specification (cont')

```
-- Package declaration, in library WORK
package LOGIC_GATES is
component AND3
    port (A, B, C : in BIT; X : out BIT);
end component;
component OR4
    port (A, B, C, D : in BIT; X : out BIT);
end component;
component NOT1
    port (A : in BIT; X : out BIT);
end component;
-- Interface
entity MAJORITY is
    port
        (A_IN, B_IN, C_IN: in BIT
         F_OUT: out BIT);
end MAJORITY;
```

# VHDL Specification (cont')

```
-- Body
-- Uses components declared in package LOGIC_GATES
-- in the WORK library
-- import all the components in WORK.LOGIC_GATES
use WORK.LOGIC_GATES.all
architecture LOGIC_SPEC of MAJORITY is
-- declare signals used internally in MAJORITY
signal A_BAR, B_BAR, C_BAR, I1, I2, I3, I4: BIT;
begin
-- connect the logic gates
NOT_1 : NOT1 port map (A_IN, A_BAR);
NOT_2 : NOT1 port map (B_IN, B_BAR);
NOT_3 : NOT1 port map (C_IN, C_BAR);
AND_1 : AND3 port map (A_BAR, B_IN, C_IN, I1);
AND_2 : AND3 port map (A_IN, B_BAR, C_IN, I2);
AND_3 : AND3 port map (A_IN, B_IN, C_BAR, I3);
AND_4 : AND3 port map (A_IN, B_IN, C_IN, I4);
OR_1 : OR3 port map (I1, I2, I3, I4, F_OUT);
end LOGIC_SPEC;
```