Chapter Contents

6.1 Basics of the Microarchitecture
6.2 A Microarchitecture for the ARC
6.3 Hardwired Control
6.4 Case Study: The VHDL Hardware Description Language
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.
  (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.
## ARC Instruction Subset

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

**SETHI Format**
```
0 0  rd  op2  imm22
```

**Branch Format**
```
0 0  cond  op2  disp22
```

**CALL format**
```
0 1  disp30
```

**Arithmetic Formats**
```
1 0  rd  op3  rs1  0 0 0 0 0 0 0 0 0 0 0 0 rs2
```

```
1 0  rd  op3  rs1  1  simm13
```

**Memory Formats**
```
1 1  rd  op3  rs1  0 0 0 0 0 0 0 0 0 0 0 0 rs2
```

```
1 1  rd  op3  rs1  1  simm13
```

<table>
<thead>
<tr>
<th>op</th>
<th>Format</th>
<th>op2</th>
<th>Inst.</th>
<th>op3 (op=10)</th>
<th>op3 (op=11)</th>
<th>cond</th>
<th>branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>SETHI/Branch</td>
<td>010</td>
<td>branch</td>
<td>0100000 addcc</td>
<td>000000 1d</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>sethi</td>
<td>0100001 andcc</td>
<td>000100 st</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>CALL</td>
<td></td>
<td></td>
<td>010010 orcc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Arithmetic</td>
<td></td>
<td></td>
<td>010110 orncc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Memory</td>
<td></td>
<td></td>
<td>100110 srl</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111000 jmpl</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PSR**
```
```

ARC Datapath
## ARC ALU Operations

<table>
<thead>
<tr>
<th>$F_3$</th>
<th>$F_2$</th>
<th>$F_1$</th>
<th>$F_0$</th>
<th>Operation</th>
<th>Changes Condition Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ANDCC (A, B)</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>ORCC (A, B)</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>NORCC (A, B)</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>ADDCC (A, B)</td>
<td>yes</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>SRL (A, B)</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>AND (A, B)</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>OR (A, B)</td>
<td>no</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>NOR (A, B)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ADD (A, B)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LSHIFT2 (A)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>LSHIFT10 (A)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>SIMM13 (A)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>SEXT13 (A)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>INC (A)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>INCPC (A)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>RSHIFT5 (A)</td>
<td>no</td>
</tr>
</tbody>
</table>
Block Diagram of ALU
Gate-Level Layout of Barrel Shifter

Principles of Computer Architecture by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring
## Truth Table for (Most of the) ALU LUTs

<table>
<thead>
<tr>
<th>$F_3$</th>
<th>$F_2$</th>
<th>$F_1$</th>
<th>$F_0$</th>
<th>Carry In</th>
<th>$a_i$</th>
<th>$b_i$</th>
<th>$z_i$</th>
<th>Carry Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**ANDCC**

**ORCC**
Design of Register $r_1$

Data inputs from C Bus

CLK

Write select (from $c_1$ bit of C Decoder)

A bus enable (from $a_1$ bit of A Decoder)

Data outputs to A Bus

B bus enable (from $b_1$ bit of B Decoder)

Data outputs to B Bus
Outputs to Control Unit from Register \%ir

Data inputs from C Bus

Instruction Register \%ir

Instruction fields

C_{31} \quad C_0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

op \quad \text{rd} \quad \text{op2} \quad \text{rs1} \quad \text{op3} \quad \text{rs2} \quad \text{bit 13}
Microarchitecture of
the ARC
Microword Format

A M U A
B M U X
C M U RW

A X B X C XDR ALU COND JUMP ADDR
### Settings for the COND Field of the Microword

<table>
<thead>
<tr>
<th>$C_2$</th>
<th>$C_1$</th>
<th>$C_0$</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Use NEXT ADDR</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Use JUMP ADDR if $n = 1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Use JUMP ADDR if $z = 1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Use JUMP ADDR if $v = 1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Use JUMP ADDR if $c = 1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Use JUMP ADDR if $IR[13] = 1$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Use JUMP ADDR</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>DECODE</td>
</tr>
</tbody>
</table>
DECODE Format for Microinstruction Address

IR bits → op  op2  op3

1 31 30 24 23 22 21 20 19

0 0
Timing Relationships for the Registers

Settling time for slave sections of registers. Perform ALU functions. \( n, z, v, \) and \( c \) flags become stable.

Master sections settle.

Clock

Master sections of registers loaded on rising edge

Slave sections of registers loaded on falling edge
Partial ARC Microprogram

Address Operation Statements Comment

0: R[ir] ← AND(R[pc],R[pc]); READ; /* Read an ARC instruction from main memory */
1: DECODE; /* 256-way jump according to opcode */
1152: R[rd] ← LSHIFT10(ir); GOTO 2047; /* Copy imm22 field to target register */
1280: R[15] ← AND(R[pc],R[pc]); /* Save %pc in %r15 */
1281: R[temp0] ← ADD(R[ir],R[ir]); /* Shift disp30 field left */
1282: R[pc] ← ADD(R[temp0],R[temp0]); /* Shift again */
1283: R[pc] ← ADD(R[pc],R[temp0]); /* Jump to subroutine */

GOTO 0;
1600: IF R[IR[13]] THEN GOTO 1602; /* Is second source operand immediate? */
1601: R[rd] ← ADDCC(R[rs1],R[rs2]); /* Perform ADDCC on register sources */
1602: R[temp0] ← SEXT13(R[ir]); /* Get sign extended simm13 field */
1603: R[rd] ← ADDCC(R[rs1],R[temp0]); /* Perform ADDCC on register/simm13 sources */
1604: IF R[IR[13]] THEN GOTO 1606; /* Is second source operand immediate? */
1605: R[rd] ← ANDCC(R[rs1],R[rs2]); /* Perform ANDCC on register sources */
1606: R[temp0] ← SIMM13(R[ir]); /* Get simm13 field */
1607: R[rd] ← ANDCC(R[rs1],R[temp0]); /* Perform ANDCC on register/simm13 sources */
1608: IF R[IR[13]] THEN GOTO 1610; /* Is second source operand immediate? */
1609: R[rd] ← ORCC(R[rs1],R[rs2]); /* Perform ORCC on register sources */
1610: R[temp0] ← SIMM13(R[ir]); /* Get simm13 field */
1611: R[rd] ← ORCC(R[rs1],R[temp0]); /* Perform ORCC on register/simm13 sources */
1624: IF R[IR[13]] THEN GOTO 1626; /* Is second source operand immediate? */
1625: R[rd] ← NORCC(R[rs1],R[rs2]); /* Perform ORNCC on register sources */
1626: R[temp0] ← SIMM13(R[ir]); /* Get simm13 field */
1627: R[rd] ← NORCC(R[rs1],R[temp0]); /* Perform NORCC on register/simm13 sources */
1688: IF R[IR[13]] THEN GOTO 1690; /* Is second source operand immediate? */
1689: R[rd] ← SRL(R[rs1],R[rs2]); /* Perform SRL on register sources */
1690: R[temp0] ← SIMM13(R[ir]); /* Get simm13 field */
1691: R[rd] ← SRL(R[rs1],R[temp0]); /* Perform SRL on register/simm13 sources */
1760: IF R[IR[13]] THEN GOTO 1762; /* Is second source operand immediate? */
1761: R[pc] ← ADD(R[rs1],R[rs2]); /* Perform ADD on register sources */
GOTO 0;
Partial ARC Microprogram (cont’)

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

- Decoding tree for branch instructions shows corresponding microprogram lines:

Line 8 \rightarrow IR[28]

0 1

IR[27] \leftarrow Line 9

0 1

Line 10 be

Line 12 ba

IR[26] \leftarrow Line 13

0 1

Line 14 bcs

IR[25] \leftarrow Line 16

0 1

Line 17 bneg

Line 19

Line 18

cond
28 27 26 25
0 0 0 1
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0

branch
be
bcs
bvs
ba
Assembled ARC Microprogram

<table>
<thead>
<tr>
<th>Microstore Address</th>
<th>A</th>
<th>M</th>
<th>U</th>
<th>X</th>
<th>B</th>
<th>M</th>
<th>U</th>
<th>X</th>
<th>C</th>
<th>M</th>
<th>U</th>
<th>W</th>
<th>X</th>
<th>D</th>
<th>R</th>
<th>ALU</th>
<th>COND</th>
<th>JUMP ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1152</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1280</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1281</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1282</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1283</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1600</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1601</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1602</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1603</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1604</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1605</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1606</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1607</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1608</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1609</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1610</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1611</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1624</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1625</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1626</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1627</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1688</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1689</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1690</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1691</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1760</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1761</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1762</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1763</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1792</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Chapter 6: Datapath and Control

### Assembled ARC Microprogram (cont’)

<table>
<thead>
<tr>
<th>A</th>
<th>M</th>
<th>B</th>
<th>M</th>
<th>U</th>
<th>U</th>
<th>W</th>
<th>ALU</th>
<th>COND</th>
<th>JUMP ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000010</td>
<td>1000001</td>
<td>0000000</td>
<td>0000000</td>
<td>0110101</td>
<td>1101111111</td>
<td>1111111111</td>
<td>0000000000</td>
<td>0000000000</td>
<td>0000000000</td>
</tr>
</tbody>
</table>
Example: Add the **subcc** Instruction

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

1584: \( R[\text{temp}0] \leftarrow \text{SEXT13}(R[\text{ir}]) \);  
   / Extract rs2 operand

\( \text{IF } IR[13] \text{ THEN GOTO 1586;} \)  
   / Is second source immediate?

1585: \( R[\text{temp}0] \leftarrow R[\text{rs}2]; \)  
   / Extract sign extended immediate operand

1586: \( R[\text{temp}0] \leftarrow \text{NOR}(R[\text{temp}0], R[0]); \)  
   / Form one's complement of subtrahend

1587: \( R[\text{temp}0] \leftarrow \text{INC}(R[\text{temp}0]); \text{ GOTO 1603;} \)  
   / Form two's complement of subtrahend

---

<table>
<thead>
<tr>
<th>1584</th>
<th>100101</th>
<th>00000000</th>
<th>01000010</th>
<th>00011001</th>
<th>10111000110010</th>
</tr>
</thead>
<tbody>
<tr>
<td>1585</td>
<td>000000</td>
<td>00000000</td>
<td>01100001</td>
<td>00010000</td>
<td>000000000000000</td>
</tr>
<tr>
<td>1586</td>
<td>100010</td>
<td>00000000</td>
<td>01000010</td>
<td>00001110</td>
<td>000000000000000</td>
</tr>
<tr>
<td>1587</td>
<td>100001</td>
<td>00000000</td>
<td>01000010</td>
<td>00011011</td>
<td>1101110010000011</td>
</tr>
</tbody>
</table>
### Branch Table

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
<th>Trap Handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>JUMP TO 2000</td>
<td>Illegal instruction</td>
</tr>
<tr>
<td>64</td>
<td>JUMP TO 3000</td>
<td>Overflow</td>
</tr>
<tr>
<td>68</td>
<td>JUMP TO 3600</td>
<td>Underflow</td>
</tr>
<tr>
<td>72</td>
<td>JUMP TO 5224</td>
<td>Zerodivide</td>
</tr>
<tr>
<td>76</td>
<td>JUMP TO 4180</td>
<td>Disk</td>
</tr>
<tr>
<td>80</td>
<td>JUMP TO 5364</td>
<td>Printer</td>
</tr>
<tr>
<td>84</td>
<td>JUMP TO 5908</td>
<td>TTY</td>
</tr>
<tr>
<td>88</td>
<td>JUMP TO 6048</td>
<td>Timer</td>
</tr>
</tbody>
</table>

*Principles of Computer Architecture* by M. Murdocca and V. Heuring © 1999 M. Murdocca and V. Heuring
Microprogramming vs. Nanoprogramming

- (a) Microprogramming vs. (b) nanoprogramming.

Original Microprogram

\[ n = 2048 \text{ words} \]
\[ w = 41 \text{ bits} \]

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

\[ k = \lceil \log_2(n) \rceil = \lceil \log_2(100) \rceil = 7 \text{ bits} \]

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

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

Total Area = 14,336 + 4100 = 18,436 bits

Microprogram

\[ n = 2048 \text{ words} \]
\[ w = 41 \text{ bits} \]

\[ m = 100 \text{ nanowords} \]
Hardware Description Language

- HDL sequence for a resettable modulo 4 counter.

Preamble

\[
\text{MODULE: MOD_4_COUNTER.} \\
\text{INPUTS: } x. \\
\text{OUTPUTS: } Z[2]. \\
\text{MEMORY:}
\]

Statements

\[
0: Z \leftarrow 0,0; \\
\text{GOTO } \{0 \text{ CONDITIONED ON } x, \text{ 1 CONDITIONED ON } \overline{x}\}. \\
1: Z \leftarrow 0,1; \\
\text{GOTO } \{0 \text{ CONDITIONED ON } x, \text{ 2 CONDITIONED ON } \overline{x}\}. \\
2: Z \leftarrow 1,0; \\
\text{GOTO } \{0 \text{ CONDITIONED ON } x, \text{ 3 CONDITIONED ON } \overline{x}\}. \\
3: Z \leftarrow 1,1; \\
\text{GOTO 0.}
\]

Epilogue

\[
\text{END SEQUENCE.} \\
\text{END MOD_4_COUNTER.}
\]
Circuit Derived from HDL

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

- HDL description of the ARC control unit.

```vhdl
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.
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]×XOR(ir[19:22]), ! addcc
   SIM13(ir) CONDITIONED ON ir[13]×XOR(ir[19:22]), ! Remaining
   ! 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 ir[28] + ir[28]×ir[27]×Z +
    INCPC(pc) CONDITIONED ON ir[28]×ir[27]×Z +
    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.

<table>
<thead>
<tr>
<th>Minterm Index</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

a) Truth table

b) AND-OR implementation

c) Black box representation
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;
-- 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;