# The Assembly Language of the Boz–5

The Boz–5 uses bits 31 – 27 of the IR as a five–bit opcode.

Of the possible 32 opcodes, only 26 are implemented.

| **Op-Code** | **Mnemonic** | **Description** |
|---|---|---|
| 00000 | HLT | Halt the Computer |
| 00001 | LDI | Load Register from Immediate Operand |
| 00010 | ANDI | Logical AND Register with Immediate Operand |
| 00011 | ADDI | Add Signed Immediate Operand to Register |
| 00100 | NOP | Not Yet Defined – At Present it is does nothing |
| 00101 | NOP | Not Yet Defined – At Present it is does nothing |
| 00110 | NOP | Not Yet Defined – At Present it is does nothing |
| 00111 | NOP | Not Yet Defined – At Present it is does nothing |

This strange gap in the opcodes caused by not using 4, 5, 6, and 7 is an example of adjusting the ISA to facilitate a simpler design of the control unit.

With this grouping, the instructions with a “00” prefix either have no argument or have an immediate argument; in short, no memory reference.

# More Opcodes

Here are the opcodes in the range 8 to 15.

| Op-Code | Mnemonic | Description |
|---|---|---|
| 01000 | GET | Input from I/O Device Register to Register |
| 01001 | PUT | Output from Register to I/O Device Register |
| 01010 | RET | Return from Subroutine |
| 01011 | RTI | Return from Interrupt (Not Implemented) |
| 01100 | LDR | Load Register from Memory |
| 01101 | STR | Store Register into Memory |
| 01110 | JSR | Call a subroutine. |
| 01111 | BR | Branch on Condition Code to Address |

Here we begin to see some structure in the ISA.

The “01” prefix is shared by all instructions that generate memory address references. This grouping simplifies the design of the Major State Register, which is an integral part of the control unit.

# The Last Opcodes

| Op-Code | Mnemonic | Description |
|---|---|---|
| 10000 | LLS | Logical Left Shift |
| 10001 | LCS | Circular Left Shift |
| 10010 | RLS | Logical Right Shift |
| 10011 | RAS | Arithmetic Right Shift |
| 10100 | NOT | Logical NOT (One's Complement) |
| 10101 | ADD | Addition |
| 10110 | SUB | Subtraction |
| 10111 | AND | Logical AND |
| 11000 | OR | Logical OR |
| 11001 | XOR | Logical Exclusive OR |

The instructions with the "10" and "11" prefix, in short all with the single–bit prefix of "1", are register–to–register operations, with no memory reference.

Again, one sees this grouping in an attempt to simplify the control unit.

# Syntax of the Assembly Language

## Immediate Operations

The syntax of the immediate operations is quite simple.

| Syntax | Example |
|---|---|
| LDI %RD, value | LDI %R3, 100 |
| ANDI %RD, %RS, value | ANDI %R5, %R3 0xFFA08 |
| ADDI %RD, %RS, value | ADDI %R5, %R2, 200 |
| NOP | NOP |

Most implementations of immediate addressing have only one register in the machine language instruction. Our implementation with two registers is again due to considerations of simplifying the control unit.

In the AND Immediate instruction, the immediate operand is zero extended to 32 bits before being applied. Let R3 contains the 32–bit number 0x77777777.

| R3 | 0111 | 0111 | 0111 | 0111 | 0111 | 0111 | 0111 | 0111 |
|---|---|---|---|---|---|---|---|---|
| FFA08 | 0000 | 0000 | 0000 | 1111 | 1111 | 1010 | 0000 | 1000 |
| Result | 0000 | 0000 | 0000 | 0111 | 0111 | 0010 | 0000 | 0000 |

# Syntax of the Assembly Language (Part 2)

## Input / Output Operations

| Syntax | Example |
|---|---|
| GET %Rn, I/O_Reg | GET %R2, XX |
| | // Load the general-purpose register from |
| | // the I/O device register. |
| PUT %Rn, I/O_Reg | PUT %R0, YY |
| | // Store the contents of register into the |
| | // I/O device register. |

## Load/Store Operations

The syntax of these is fairly simple. For direct addressing we have the following.

| Syntax | Example | |
|---|---|---|
| LDR %Rn, address | LDR %R3, X | Loads %R3 from address X |
| STR %Rn, address | STR %R0, Z | Loads %R0 into address Z |
| | | This clears M[Z] |

# Syntax of the Assembly Language (Part 3)

## Branch

The syntax of this instruction, and its variants, is quite simple.

| Syntax | Example |
|---|---|
| BR Condition_Code, address | BRU 5, W |

The condition code field determines under which conditions the Branch instruction is executed. The eight possible options are.

Condition Code

| Decimal | Binary | Action |
|---|---|---|
| 0 | 000 | Branch Always (Unconditional Jump) |
| 1 | 001 | Branch on negative result |
| 2 | 010 | Branch on zero result |
| 3 | 011 | Branch if result not positive |
| 4 | 100 | Branch if carry–out is 0 |
| 5 | 101 | Branch if result not negative |
| 6 | 110 | Branch if result is not zero |
| 7 | 111 | Branch on positive result |

# Syntax of the Assembly Language (Part 4)

## Unary Register-To-Register

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 – 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op Code | | | | | | Destination Register | | | Source Register | | | 5–bit Shift Count (0 – 31) | | | | | Not Used |

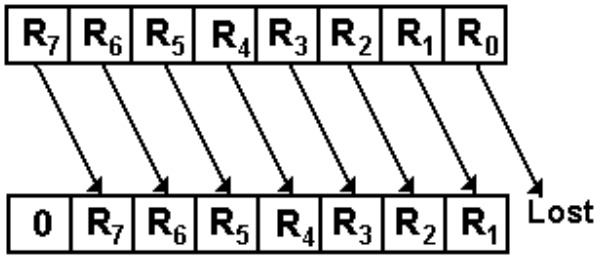
| | | | |
|---|---|---|---|
| Opcode = | 10000 | LLS | Logical Left Shift |
| | 10001 | LCS | Circular Left Shift |
| | 10010 | RLS | Logical Right Shift |
| | 10011 | RAS | Arithmetic Right Shift |
| | 10100 | NOT | Logical NOT (Shift count ignored) |

NOTES:
1. If (Count Field) = 0, a shift operation becomes a register move.
2. If (Source Register = 0), the operation becomes a clear.
3. Circular right shifts are not supported, because they may be implemented using circular left shifts.
4. The shift count, being a 5 bit number, has values 0 to 31 inclusive.

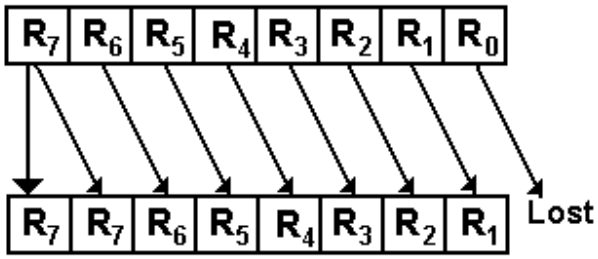
# Shift Examples

Here are figures showing the varieties of right shifts on signed 8–bit integers.

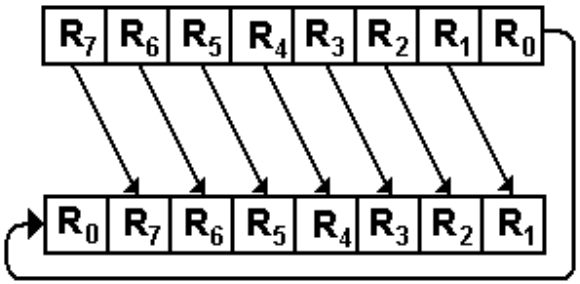
**Logical Right Shift**




**Arithmetic Right Shift $R_7$ (sign bit) is preserved.**




**Circular Right Shift Nothing is lost.**



# Syntax of the Assembly Language (Part 5)

Binary Register-To-Register

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 – 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Op Code | | | | | | Destination Register | | | Source Register 1 | | | Source Register 2 | | | Not used |

| | | | |
|---|---|---|---|
| Opcode = | 10101 | ADD | Addition |
| | 10110 | SUB | Subtraction |
| | 10111 | AND | Logical AND |
| | 11000 | OR | Logical OR |
| | 11001 | XOR | Logical Exclusive OR |

NOTES: Subtract with (Destination Register) = 0 becomes a compare to set condition codes.

# Syntactic Sugar

| Syntactic Sugar | Assembled As | Comments |
|---|---|---|
| CLR %R2 | LDI %R2, 0 | Clears the register |
| CLW X | STR %R0, X | Clears the word at address X |
| INC %R2 | ADDI %R2, 1 | Increments the register |
| DEC %R2 | ADDI %R2, -1 | Decrements the register |
| NOP | LLS %R0, %R0, 0 | No-Operation: Does Nothing |
| RCS %R3, %R1, 3 | LCS %R3, %R1, 29 | |

Right circular shift by N is the same as left circular by (32 – N). The shift count must be a constant number.

| | | |
|---|---|---|
| DBL %R2 | LLS %R2, %R2, 1 | Left shift by one is the same as a multiply by 2. |
| MOV %R2, %R3 | LSH %R2, %R3, 0 | Shift by 0 is a copy. |
| NEG %R4, %R5 | SUB %R4, %R0, %R5 | Subtract from %R0 ≡ 0 is the same as negation. |

# Syntactic Sugar (Part 2)

| Syntactic Sugar | Assembled As | Comments |
|---|---|---|
| TST %R1 | SUB %R0, %R1, %R0 | |

This compares %R1 to zero by subtracting %R0 from it, discarding the result.

| | | |
|---|---|---|
| CMP %R1, %R2 | SUB %R0, %R1, %R2 | |

Determines the sign of(%R1) – (%R2), discarding the result.

| | | |
|---|---|---|
| BRU | BR 000 | Branch always |
| BLT | BR 001 | Branch if negative |
| BEQ | BR 010 | Branch if zero |
| BLE | BR 011 | Branch if not positive |
| BCO | BR 100 | Branch if carry out is 0 |
| BGE | BR 101 | Branch if not negative |
| BNE | BR 110 | Branch if not zero |
| BGT | BR 111 | Branch if positive |