**Introduction:**

This 4-bit microcontroller can perform 14 arithmetic and logic operations, for a total of 16 code lines, since there can be a max. of 16 addresses inside ROM. Single CLK is provided to the entire microcontroller. CLK is originally set at 0.5 Hz.

**Registers:**

We have used 5 registers in this microprocessor, two of which are always enabled on CLK and the other three are enabled based on decoder circuit.

**Ra:** General Purpose Register A

**Rb**: General Purpose Register B

**R0**: Z-register

**Rc**: Accumulator

**MDR:** Memory Data Register

Ra, Rb and Ro are controlled by Decoder, while Rc serves as accumulator after Adder, it is used for flags and displays Sum/Difference in 7-segment display. MDR is used to temporarily store the current memory data for 1 c.c., used for time synchronization.

**ROM Lines:**

Since each address contains 8-bit data, for every iteration of the PC, we have 8 bits coming out of ROM designated as,

**J**: Jump Control

**C**: Conditional Jump Control

**E1, E0**: Decoder Control

**S**: Add/Subtract Control OR I2: Immediate Value bit-2

**I1**: Immediate Value bit-1

**I0**: Immediate Value bit-0

**Conditional Jump:**

This microprocessor offers two flags for conditional jumps:

**CF**: Carry Flag obtained from Cout of Adder

**ZF**: Zero Flag obtained by NAND-gate of Cout and all 4 Sum bits

Based on these two flags, we have 4 Branch Instructions, **BRCC, BRCS, BRNE** and **BREQ** with similar functions as in AVR architecture. NOTE: branch instructions require previous instruction to have involved adder e.g., we cannot perform branch instructions after LDI, which is limitation of 4-bit architecture.

**Jump:**

This microprocessor offers two unconditional jumps:

**JMP**: Jump to immediate value (3-bit)

**IJMP**: Jump to value stored in R0 register (4-bit)

IJMP requires that R0 has values stored in it, via the PUSH command.

To choose between both, a MUX is used whose selection pin is the enable pin of R0 from decoder.

**PUSH:** copies data from Ra register and stores in R0 register.

**Decoder:**

ENABLE achieved by AND Gate with CLK supplied to registers.

|  |  |  |
| --- | --- | --- |
| E1 | E0 | ENABLE |
| 0 | 0 | None |
| 0 | 1 | Ro |
| 1 | 0 | Ra |
| 1 | 1 | Rb |

**MUX and Sreg pin:**

Sreg pin serves as selection pin for MUX (Quad 2x1). Also, while Sreg is clear, S pin coming out of ROM can safely act as an immediate value pin since MUX will select ROM lines, and when Sreg is set, S pin will act as subtract/add control, since MUX selects ALU lines.

|  |  |  |
| --- | --- | --- |
| Sreg | D2 | MUX |
| 0 | I2 | Memory Lines (ROM) |
| 1 | S | ALU |

**Branch and Jump:**

When LOAD of PC is 0, parallel data inputs are loaded into counter, and when LOAD is 1, PC increments synchronously.

Don’t Cares in CF and ZF are implemented according to code. For e.g., when code has instruction BRCS, value of ZF is considered don’t care.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| J | C | CF | ZF | LOAD (active low) |
| 0 | 0 | X | X | 1 |
| 0 | 1 | X | 1 | 0 (perform BREQ) |
| 0 | 1 | X | 0 | 0 (perform BRNE) |
| 0 | 1 | 1 | X | 0 (perform BRCS) |
| 0 | 1 | 0 | X | 0 (perform BRCC) |
| 1 | X | X | X | 0 (perform JMP) |

**Instruction Set:**

Opcode for each instruction varies from 3-4 bits. **MOV**, **INC, DEC** are 2 byte-instruction taking 2 clock cycles due to limitations of 4-bit architecture.

~Rx refers to other register for Rx e.g if Rx is Ra then ~Rx is Rb and vice versa

The opcode of each mnemonic is highlighted in grey. For branch instructions opcode is the same but internal circuitry based on CF and ZF flags determine operation.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|  | J | C | E1 | E0 | Sreg | S/I2 | I1 | I0 |
| NOP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LDI Rx,k3 | 0 | 0 | Rx | Rx | 0 | Kx | Kx | Kx |
| ADD Rd, Rs | 0 | 0 | Rd | Rd | 1 | 0 | 0 | 0 |
| SUB Rd, Rs | 0 | 0 | Rd | Rd | 1 | 1 | 0 | 0 |
| PUSH | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| JMP k3 | 1 | 0 | 0 | 0 | 0 | Kx | Kx | Kx |
| IJMP | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| BRCC k3 | 0 | 1 | 0 | 0 | 0 | Kx | Kx | Kx |
| BREQ k3 | 0 | 1 | 0 | 0 | 0 | Kx | Kx | Kx |
| BRCS k3 | 0 | 1 | 0 | 0 | 0 | Kx | Kx | Kx |
| BRNE k3 | 0 | 1 | 0 | 0 | 0 | Kx | Kx | Kx |
| MOV Rd, Rs | 0  0 | 0  0 | Rd  Rd | Rd  Rd | 0  1 | 0  0 | 0  0 | 0  0 |
| INC Rx | 0  0 | 0  0 | ~Rx  Rx | ~Rx  Rx | 0  1 | 0  0 | 0  0 | 1  0 |
| DEC Rx | 0  0 | 0  0 | ~Rx  Rx | ~Rx  Rx | 0  1 | 0  1 | 0  0 | 1  0 |

**Recommendations:**

POP command (copy from Ro to Ra) can also be implemented by

- using E1:E0 = 00 as well as E1: E0 = 10 as enable for Ra

- adding a MUX (Quad 2x1) at input of Ra with data lines A1:A4 from the Sreg MUX and data lines B1:B4 from the R0 register output. Selection line will be E1 such that when E1 = 0, Ra gets data from Ro and when E1 = 1, Ra gets data from Sreg MUX. This additional circuit has been avoided to prevent complicated hardware.