| | | | |
|---|---|---|---|
| **Fetch** | PC_out, MAR_in, Read, Clear Y, Set Carry, F=ADD, Z_in | | |
| | Z_out, PC_in, WMFC | | |
| **Decode** | MDR_out, IR_in | | |
| **Branch to Instruction** | μ-PC <= PLA(IR)$<br>[Source Fetching]::(Addressing Mode),<br>[Destination Fetching]::(Addressing Mode),<br>[No OP]::(Instruction),<br>[Jump Subroutine]::(Instruction)<br>[Branch]::Branch Offset,<br>END [1] | 0 | This branches to the corresponding type of instruction to fetch the operands if there are any |
| | **Source Fetching** | | |
| **Direct Register Mode** | (Rsrc)_out, SRC_in | | Source fetching is skipped in case of single operand instructions. |
| | μ-PC <= PLA(IR)$ [Destination Fetching]::(Addressing Mode) [2] | 1 | |
| **Indirect Register Mode** | (Rsrc)_out, MAR_in, Read, WMFC | | |
| | μ-PC <= PLA(IR)$ [Source Fetching]::Move MDR to SRC | 1 | |
| **Autoincrement Mode** | (Rsrc)_out, MAR_in, Clear Y, Set Carry, F=ADD, Z_in | | |
| | Z_out, (Rsrc)_in, WMFC | | |
| | μ-PC <= PLA(IR)$ [Source Fetching]::Indirect Mode, Move MDR to SRC | 1 | If indirect mode go to Indirect Mode, else go to Move MDR to SRC or Move MDR to DST |
| **Autodecrement Mode** | (Rsrc)_out, SET Carry, Clear Y, F=SUB, Z_in | | |
| | Z_out, (Rsrc)_in, MAR_in, Read, WMFC | | |
| | μ-PC <= PLA(IR)$ [Source Fetching]::Indirect Mode, Move MDR to SRC | 1 | |
| **Indexed** | PC_out, MAR_in, Read, Clear Y, Set Carry, F=ADD, Z_in | | |
| | Z_out, PC_in, WMFC | | |
| | MDR_out, Y_in | | |
| | (Rsrc)_out, F=ADD, Z_in | | |
| | Z_out, MAR_in, Read, WMFC | | |
| | μ-PC <= PLA(IR)$ [Source Fetching]::Indirect Mode, Move MDR to SRC | 1 | |
| **Indirect Mode** | MDR_out, MAR_in, Read, WMFC | | |
| **Move MDR to SRC** | MDR_out, SRC_in | | |
| | **Destination Fetching** | | |
| **Direct Register Mode** | (Rdst)_out, DST_in | | |
| | μ-PC <= PLA(IR)$ [Double Operand]::(Instruction) [3] | 2 | |
| **Indirect Register Mode** | (Rdst)_out, MAR_in, Read, WMFC | | |
| | μ-PC <= PLA(IR)$ [Destination Fetching]::Move MDR to DST | 2 | |
| **Autoincrement Mode** | (Rdst)_out, MAR_in, Read, SET Carry, Clear Y, F=ADD, Z_in | | |
| | Z_out, (Rdst)_in, WMFC | | |
| | μ-PC <= PLA(IR)$ [Destination Fetching]::Indirect Mode, Move MDR to DST | 2 | If indirect mode go to Indirect Mode, else go to Move MDR to SRC or Move MDR to DST |
| **Autodecrement Mode** | (Rdst)_out, SET Carry, Clear Y, F=SUB, Z_in | | |
| | Z_out, (Rdst)_in, MAR_in, Read, WMFC | | |
| | μ-PC <= PLA(IR)$ [Destination Fetching]::Indirect Mode, Move MDR to DST | 2 | |
| **Indexed** | PC_out, MAR_in, Read, Set Carry, Clear Y, F=ADD, Z_in | | |
| | Z_out, PC_in, WMFC | | |
| | MDR_out, Y_in | | |
| | (Rdst)_out, F = ADD, Z_in | | |
| | Z_out, MAR_in, Read, WMFC | | |
| | μ-PC <= PLA(IR)$ [Destination Fetching]::Indirect Mode, Move MDR to DST | 2 | |
| **Indirect Mode** | MDR_out, MAR_in, Read, WMFC | | |
| **Move MDR to DST** | MDR_out, DST_in | | |
| | μ-PC <= PLA(IR)$ [Single Operand]::(Instruction), [Double Operand]::(Instruction) | 3 | |
| | **Double Operand** | | |
| **MOV SRC, DST** | SRC_out, DST_in | | |

| | | | |
|---|---|---|---|
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 4 | |
| **ADD SRC, DST** | SRC_out, Y_in | | |
| | DST_out, Z_in, CLR Carry, F=ADD | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 4 | |
| **ADC SRC, DST** | SRC_out, Y_in | | |
| | DST_out, Z_in, SET Carry, F=ADD | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 4 | |
| **SUB SRC, DST** | SRC_out, Y_in | | |
| | DST_out, Z_in, CLR Carry, F=SUB | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 4 | |
| **SBC SRC, DST** | SRC_out, Y_in | | |
| | DST_out, Z_in, SET Carry, F=SUB | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 4 | |
| **AND SRC, DST** | SRC_out, Y_in | | |
| | DST_out, Z_in, F=AND | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 4 | |
| **OR SRC, DST** | SRC_out, Y_in | | |
| | DST_out, Z_in, F=OR | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 4 | |
| **XOR SRC, DST** | SRC_out, Y_in | | |
| | DST_out, Z_in, F=XOR | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 4 | |
| **CMP SRC, DST** | SRC_out, Y_in | | |
| | DST_out, Z_in, CLR Carry, F=SUB | | |
| | μ-PC <= PLA(IR)$ END | 4 | |
| | **Single Operand** | | |
| **INC DST** | DST_out, SET Carry, Clear Y, F=ADD, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| **DEC DST** | DST_out, SET Carry, Clear Y, F=SUB, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| **CLR DST** | DST_out, Clear Y, F=AND, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| **INV DST** | DST_out, F=INV, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| **LSR DST** | DST_out, F=LSR, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| **ROR DST** | DST_out, F=ROR, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| **ASR DST** | DST_out, F=ASR, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| **LSL DST** | DST_out, F=LSL, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| **ROL DST** | DST_out, F=ROL, Z_in | | |
| | μ-PC <= PLA(IR)$ [Move Z to Rdst]::Direct Register Mode, Indirect Write Mode | 3 | |
| | **Branch** | | |
| **Branch Offset** | (IRoffset)_out, SRC_in | | |
| | PC_out, DST_in | | |
| | μ-PC <= PLA(IR)$ [Double Operand]::ADD SRC, DST | 1 | |
| | **No Operand** | | |
| **HALT** | It is implemented by a HALT flag raised in the PLA to set a signal to stop PC from updating | | |

| | | | |
|---|---|---|---|
| **RESET** | Raise a reset signal in the PLA to reset all devices | | |
| **NOP** | μ-PC <= PLA(IR)$ END | 1 | |
| | **Jump Subroutine** | | |
| **JSR** | PC_out, MAR_in, Read, Clear Y, Set Carry, F=ADD, Z_in | | Here we rewrote the Fetch μ-program to save the clock cycle resulting from μ-branching, with each execution of the Fetch μ-program, trading off two words in the control store. |
| | Z_out, PC_in, WMFC | | |
| | MDR_out, SRC_in | | |
| | PC_out, MDRin | | |
| | μ-PC <= PLA(IR)$ PUSH | 1 | |
| **JSR after PUSH** | (IRAddress)_out, PC_in | | |
| | μ-PC <= PLA(IR)$ END | 3 | |
| **INTERRUPT** | STATUS_out, MDRin | | ied we cannot determine which instance to branch back to based solely on the instruction. There |
| | μ-PC <= PLA(IR)$ PUSH | 1 | |
| **INTERRUPT after PUSH** | PC_out, MDRin | | |
| | SP_out, Clear Y, Set Carry, F=SUB, Z_in | | This is another PUSH microprogram, but we do not know how to branch back from PUSH |
| | Z_out, SP_in, MAR_in, Write, WMFC | | |
| | (InterruptAddress)_out, PC_in | | |
| | μ-PC <= PLA(IR)$ END | 3 | |
| **Start IRET/RTS/POP PC** | SP_out, MAR_in, Read, Clear Y, Set Carry, F=ADD, Z_in | | |
| | Z_out, SP_out, WMFC | | |
| | MDR_out, PC_in | | |
| | μ-PC <= PLA(IR)$ END, Continue IRET | 1 | |
| **Continue IRET** | SP_out, MAR_in, Read, Clear Y, Set Carry, F=ADD, Z_in | | This is a POP microprogram but we do not know how to branch back here |
| | Z_out, SP_out, WMFC | | |
| | MDR_out, STATUS_in | | |
| | μ-PC <= PLA(IR)$ END | 2 | |
| **PUSH** | SP_out, Clear Y, Set Carry, F=SUB, Z_in | | |
| | Z_out, SP_in, MAR_in, Write, WMFC | | |
| | μ-PC <= PLA(IR)$ INTERRUPT after PUSH, JSR after PUSH | 2 | |
| | **Move Z back to Rdst** | | |
| **Direct Register Mode** | Z_out, (Rdst)_in | | etermined by a decoder, controlled by IR, which sets the enable line for the corresponding destin |
| | μ-PC <= PLA(IR)$ END | 1 | |
| **Indirect Write Mode** | Z_out, MDRin, Write, WMFC | | |
| | **End** | | |
| **END** | Send END signals | | |
| | | | |

[1] If double or single operand:
  branch to corresponding addressing mode
else if no op/branch/jsr
  branch to no op/branch/jsr

For HALT: raise HALT signal
For NOP: branch to Fetch again
For RTS: jump directly to POP

[2] Jumps to instruction microprogram using PLA

[3] Jumps to instruction microprogram using PLA