ARTHMETIC AND LOGIC INSTRUCTIONS	Mnemonics	Operands	Description	Operation
ADDR	ARITHMETIC AND LOGIC IN	STRUCTIONS		
ANDR Rd, K Logical AND register by Constant Rd - Rd & K ANDR Rd, Kr Logical AND two Registers Rd - Rd & Rr DIVB Rd, K Divide Register by Constant Rd - Rd - K DIVB Rd, Rr Divide Register by Constant Rd - Rd - K DIVB Rd, R Divide Workpetter by Constant Rd - Rd - Rd - I NCR Rd Module Needed by Constant Rd - Rd - I NCDR Rd, K Module two Registers Rd - Rd - I MODR Rd, R Module two Registers Rd - Rd - Rr MUUR Rd, K Multiply Register by Constant Rd - Rd - Rr MUR Rd, Rr Multiply Register by Constant Rd - Rd - Rr NOT Rd Logical R Register by Constant Rd - Rd - Rd - Rr NOT Rd Logical R Register by Constant Rd - Rd - Rd - Rr NOT Rd, Rr Logical R Register by Constant Rd - Rd - Rd - Rr NOR Rd, Rr Logical R Register by Constant Rd - Rd - Rd - Rr SUBB Rd, R Random between Reg	ADDB	Rd, K	Add Constant to Register	Rd <- Rd + K
MADUR RG, Rr Divide Not Register by Constant Rd < Rd ≥ Rt ≥	ADDR	Rd, Rr	Add two Registers	Rd <- Rd + Rr
DVPR	ANDB	Rd, K	Logical AND Register by Constant	Rd <- Rd & K
DVPR	ANDR	Rd, Rr	Logical AND two Registers	Rd <- Rd & Rr
DECK Rd	DIVB	Rd, K	Divide Register by Constant	Rd <- Rd + K
NCR	DIVR	Rd, Rr	Divide two Registers	Rd <- Rd + Rr
MODB Rd, K Modulo Register by Constant Rd < Rd + K MODR Rd, K Modulo two Registers Rd < Rd - K MULB Rd, K Multiphy two Registers Rd < Rd + K MULR Rd, Rr Multiphy two Registers Rd < Rd + K MUCR Rd, Rr Multiphy two Registers Rd < Rd K ORR Rd, K Logical OR Register by Constant Rd < Rd K ORR Rd, Rr Logical OR two Registers Rd < Rd K RNDB Rd, K Random between Register and Constant Rd < Rd K S RNDR Rd, Rr Random between Registers Rd < Rd K S SUBB Rd, K Subtract Constant from Register Rd < Rd Rr S SUBB Rd, Rr Subtract Lwo Registers Rd < Rd - Rr Rr S SUBB Rd, Rr Subtract Lwo Registers Rd < Rd - Rr Rr Rr Rr Rr Rr Rr Rr	DECR	Rd	Decrease Register by one	Rd <- Rd – 1
MOULB Rd, K	INCR	Rd	Increase Register by one	Rd <- Rd + 1
MUILB	MODB	Rd, K	Modulo Register by Constant	Rd <- Rd + K
MULIR Rd, Rr	MODR	Rd, Rr	Modulo two Registers	Rd <- Rd + Rr
NOT Rd Logical NOT Register Rd < Rd K ORB Rd, K Logical OR Register by Constant Rd < Rd K	MULB	Rd, K	Multiply Register by Constant	Rd <- Rd + K
ORB Rd, K Logical OR Register by Constant Rd < Rd Rr ORR Rd, Rr Logical OR two Registers Rd < Rd Rr	MULR	Rd, Rr	Multiply two Registers	Rd <- Rd + Rr
ORR Rd, Rr Logical OR two Registers Rd < Rd Rr RNDB Rd, K Random between Register and Constant Rd < Rd ? (R - 1)	NOT	Rd	Logical NOT Register	Rd <- ~Rd
RNDB Rd, K Random between Register and Constant Rd < Rd ? (K - 1) RNDR Rd, Rr Random between Registers Rd < Rd ? (K - 1)	ORB	Rd, K	Logical OR Register by Constant	Rd <- Rd K
RNDR Rd, Rr Random between Registers Rd < Rd ? (Rr - 1) SUBB Rd, K Subtract Constant from Register Rd < Rd - Rr	ORR	Rd, Rr	Logical OR two Registers	Rd <- Rd Rr
SUBB Rd, K Subtract Constant from Register Rd < Rd - K SUBR Rd, Rr Subtract two Registers Rd < Rd - Rr SWAPA Rd, Rr Swap Rd to Rr and Rr to Rd Rd < - Rt, Rr < - Rd XORB Rd, K Logical XOR Register by Constant Rd < - Rd ^ K KXORR BANDB Rd, K Boolean AND Register by Constant Rd < - Rd & & K KB BANDB Rd, Kr Boolean AND two Registers Rd < - Rd & & K BANDT Rd Boolean AND Register Rd < - Rd & Rd & Rr BNOT Rd Boolean OR wo Register Rd < - Rd IK BORB Rd, K Boolean OR wo Register Rd < - Rd IK BORR Rd, Rr Boolean OR wo Register by Constant Rd < Rd < Rd < K LSHFTB Rd, K Logical Left Shift Register by Constant Rd < Rd < K KI SHT LSHFTB Rd, K Logical Left Shift Register by Constant Rd < Rd < Rd < K Kd < Rd < Rd < K LSHFTB Rd, K Logical Left Shift Register by Constant Rd < Rd < Rd < K Rd < Rd < Rd < K LSHFT	RNDB	Rd, K	Random between Register and Constant	Rd <- Rd ? (K - 1)
SUBR Rd, Rr Subtract two Registers Rd < Rd - Rr SWAP Rd, Rr Swap Rd to Rr and Rr to Rd Rd < Rr, Rr < Rd	RNDR	Rd, Rr	Random between Registers	Rd <- Rd ? (Rr - 1)
SWAP Rd, Rr Swap Rd to Rr and Rr to Rd Rd < Rr, Rr < Rd XORB Rd, K Logical XOR Register by Constant Rd < Rd ^ Rr SANDB Rd, Rr Logical XOR New Registers Rd < Rd ^ Rr BANDB Rd, K Boolean AND Register by Constant Rd < Rd & & K BANDR Rd, Rr Boolean AND two Registers Rd < Rd & & K BORD Rd, Rr Boolean AND two Registers Rd < Rd & & Rd & Rd BORB Rd, K Boolean AND two Registers Rd < Rd < Rd BORB Rd, K Boolean OR Register by Constant Rd < Rd IR BORB Rd, Rr Boolean OR Register by Constant Rd < Rd IR BORB Rd, Rr Boolean OR Register by Constant Rd < Rd < Rd IR Rd, Rr Logical Left Shift Register by Constant Rd < Rd < R Rd, K Logical Right Shift Register by Register Rd < Rd < Rr Rd, K Logical Right Shift Register by Register Rd < Rd < Rr Rd, K Logical Right Shift Register by Register Rd < Rd < Rd <	SUBB	Rd, K	Subtract Constant from Register	
SWAP Rd, Rr Swap Rd to Rr and Rr to Rd Rd < Rr, Rr < Rd XORB Rd, K Logical XOR Register by Constant Rd < Rd ^R R XORB Rd, R Logical XOR New Registers Rd < Rd ^R R BANDB Rd, K Boolean AND Register by Constant Rd < Rd & & K BANDB Rd, Rr Boolean AND two Registers Rd < Rd & & K BANDB Rd, Rr Boolean AND two Registers Rd < Rd & & K BANDB Rd, Rr Boolean AND two Registers Rd < Rd & Rd & Rd Rd BORB Rd, K Boolean OR Two Registers Rd < Rd IR BORB Rd, Rr Boolean OR Register by Constant Rd < Rd IR BORB Rd, Rr Boolean OR Register by Constant Rd < Rd < Rd IR Rd, Rr Logical Right Shift Register by Constant Rd < Rd < Rd < Rr Rd, Rr Logical Right Shift Register by Register Rd < Rd < Rd < Rr SHFTR Rd, Rr Logical Right Shift Register by Constant Rd < Rd < Rd < Rr SHFTR Rd, Rr Logical Right Shift Register by Register <td>SUBR</td> <td></td> <td>Subtract two Registers</td> <td>Rd <- Rd - Rr</td>	SUBR		Subtract two Registers	Rd <- Rd - Rr
XORR Rd, Rr Logical XOR two Registers Rd < Rd ^Rr BANDB Rd, K Boolean AND Register by Constant Rd < Rd & & K	SWAP	Rd, Rr	Swap Rd to Rr and Rr to Rd	Rd <- Rr, Rr <- Rd
BANDB Rd, K Boolean AND Register by Constant Rd < Rd && K BANDR Rd, Rr Boolean AND two Registers Rd < Rd && Rr BNOT Rd Boolean NOT Register Rd < Rd && Rd BNOBB Rd, K Boolean NOT Register by Constant Rd < Rd K BORR Rd, K Boolean OR Register by Constant Rd < Rd Rr BIT INSTRUCTIONS LSHFTB Rd, K Logical Left Shift Register by Constant Rd < Rd < K LSHFTB Rd, K Logical Right Shift Register by Constant Rd < Rd < Rd < Rr RSHFTB Rd, K Logical Right Shift Register by Constant Rd <	XORB	Rd, K	Logical XOR Register by Constant	Rd <- Rd ^ K
BANDR Rd, Rr Boolean AND two Registers Rd < Rd & Rr BNOT Rd Boolean NOT Register Rd < Rd Rd Boolean NOT Register Rd < Rd Rd Rd Boolean NOT Register Rd < Rd < Rd K Rd BOORR Rd, K Boolean OR two Registers Rd < Rd K Rd < Rd Rd < Rd Rd < Rd K Rd < Rd Rd < Rd < Rd < Rd Rd < Rd <	XORR	Rd, Rr	Logical XOR two Registers	Rd <- Rd ^ Rr
BANDR Rd, Rr Boolean AND two Registers Rd <- Rd & Rr BNOT Rd Boolean NOT Register Rd <- Rd BORB Rd, K Boolean OR Two Registers Rd <- Rd K BORB Rd, Rr Rd, Rr Boolean OR Register by Constant Rd <- Rd K BORB Rd, Rr Rd, Rr Boolean OR Register by Constant Rd <- Rd K BIT INSTRUCTIONS LSHFTB Rd, K Logical Left Shift Register by Constant Rd <- Rd <- Rd <- Rd K BSHFTB Rd, Rr Logical Left Shift Register by Register Rd <- R	BANDB	Rd, K		Rd <- Rd && K
BORB Rd, K Boolean OR two Registers Rd < Rd K BORR Rd, Rr Boolean OR Register by Constant Rd < Rd K BIT INSTRUCTIONS USHFTB Rd, K Logical Left Shift Register by Constant Rd < Rd << K LSHFTB Rd, K Logical Right Shift Register by Register Rd < Rd << RSHFTB	BANDR			Rd <- Rd && Rr
BORB Rd, K Boolean OR two Registers Rd < Rd K BORR Rd, Rr Boolean OR Register by Constant Rd < Rd K BIT INSTRUCTIONS LSHFTB Rd, K Logical Left Shift Register by Constant Rd < Rd << K LSHFTB Rd, Rr Logical Right Shift Register by Register Rd < Rd << K RSHFTB Rd, R Logical Right Shift Register by Register Rd < Rd << K RSHFTB Rd, Rr Logical Right Shift Register by Register Rd < Rd << K RSHFTB Rd, Rr Logical Right Shift Register by Register Rd < Rd << K BRANCH INSTRUCTIONS JMP W Direct Jump PC < W JMP W Direct Jump if Greater Than If (F = G) then PC < W JMPGE W Direct Jump if Greater of Equal If (F > G) then PC < W JMPL W Direct Jump if Greater of Equal If (F > G) then PC < W JMPLE W Direct Jump if Sec of Equal If (F > G) then PC < W JMPLE W Direct Jump if Sec if Equal If (F = G) then P	BNOT	Rd	Boolean NOT Register	Rd <- !Rd
BORR Rd, Rr Bolean OR Register by Constant Rd < Rd Rr BIT INSTRUCTIONS LSHFTR Rd, K Logical Left Shift Register by Constant Rd < Rd < K < K < K < K < K < K < K < K < K <	BORB	Rd, K	_	Rd <- Rd K
BIT INSTRUCTIONS	BORR			
Logical Left Shift Register by Register Rd < Rd < Rr RSHFTB Rd, K Logical Right Shift Register by Constant Rd < Rd < Rd < Rd < RSHFTB Rd, K Logical Right Shift Register by Constant Rd < Rd < Rd < Rd < RSHFTR Rd, Rr Logical Right Shift Register by Register Rd < Rd	BIT INSTRUCTIONS	·	, ,	
RSHFTB Rd, K Rd, K Logical Right Shift Register by Constant Rd < Rd < K RSHFTR Rd, Rd, Rr Logical Right Shift Register by Register Rd < Rd < Rr BRANCH INSTRUCTIONS JMPP W Direct Jump JMPE W Direct Jump if Equal If (F = G) then PC < W JMPG W Direct Jump if Greater Than If (F > G) then PC < W JMPGE W Direct Jump if Greater or Equal If (F > G) then PC < W JMPGE W Direct Jump if Greater or Equal If (F > G) then PC < W JMPL W Direct Jump if Less or Equal If (F > G) then PC < W JMPL W Direct Jump if Less Than If (F < G) then PC < W JMPLE W Direct Jump if Less or Equal If (F = G) then PC < W JMPNE W Direct Jump if Not Equal If (F = G) then PC < W JMPNS W Direct Jump if Specified milliseconds have elapsed. If (millis-time) PC < W JMPUS W Direct Jump if specified microseconds have elapsed. If (millis-time) PC < W JMPDE W Direct Jump if two data arrays are equal. If (data=data2) PC < W JMPDE W Direct Jump if two data arrays are envelual. If (data=data2) PC < W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC < W JMPDNE W Direct Jump if two data arrays are unequal. If (Rm (R (1 < Rr)) skip SKPBH Rr, Rr Skip next instruction if Register bit is high If (Rr & (1 < Rr)) skip SKPBL Rr, Rr Skip next instruction if Register bit is low If (I(Rr) & (1 < Rr)) skip DATA TRANSFER INSTRUCTIONS AREF K A Analog reference selection of Constant AREF < K DGTLOB R R Digital write the Register svalue DGTLOR Rr Digital write the Register svalue DGTLOR Rr Digital write the Register Svalue DGTLOR Rr Digital read into Register INB Rd, R Rd, R In Port Constant Rd < P MOVB Rd, K Move Between Register MOVB Rd, K Move Between Registers Rd < Rr OUT K, K Out Port Constant Out Port Register	LSHFTB	Rd, K	Logical Left Shift Register by Constant	Rd <- Rd << K
RSHFTB Rd, K Rd, K Logical Right Shift Register by Constant Rd < Rd < K RSHFTR Rd, Rd, Rr Logical Right Shift Register by Register Rd < Rd < Rr BRANCH INSTRUCTIONS JMPP W Direct Jump JMPE W Direct Jump if Equal If (F = G) then PC < W JMPG W Direct Jump if Greater Than If (F > G) then PC < W JMPGE W Direct Jump if Greater or Equal If (F > G) then PC < W JMPGE W Direct Jump if Greater or Equal If (F > G) then PC < W JMPL W Direct Jump if Less or Equal If (F > G) then PC < W JMPL W Direct Jump if Less Than If (F < G) then PC < W JMPLE W Direct Jump if Less or Equal If (F = G) then PC < W JMPNE W Direct Jump if Not Equal If (F = G) then PC < W JMPNS W Direct Jump if Specified milliseconds have elapsed. If (millis-time) PC < W JMPUS W Direct Jump if specified microseconds have elapsed. If (millis-time) PC < W JMPDE W Direct Jump if two data arrays are equal. If (data=data2) PC < W JMPDE W Direct Jump if two data arrays are envelual. If (data=data2) PC < W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC < W JMPDNE W Direct Jump if two data arrays are unequal. If (Rm (R (1 < Rr)) skip SKPBH Rr, Rr Skip next instruction if Register bit is high If (Rr & (1 < Rr)) skip SKPBL Rr, Rr Skip next instruction if Register bit is low If (I(Rr) & (1 < Rr)) skip DATA TRANSFER INSTRUCTIONS AREF K A Analog reference selection of Constant AREF < K DGTLOB R R Digital write the Register svalue DGTLOR Rr Digital write the Register svalue DGTLOR Rr Digital write the Register Svalue DGTLOR Rr Digital read into Register INB Rd, R Rd, R In Port Constant Rd < P MOVB Rd, K Move Between Register MOVB Rd, K Move Between Registers Rd < Rr OUT K, K Out Port Constant Out Port Register	LSHFTR	Rd, Rr	Logical Left Shift Register by Register	Rd <- Rd << Rr
BRANCH INSTRUCTIONS JMP W Direct Jump if Equal If (F = G) then PC <- W JMPG W Direct Jump if Greater Than If (F > G) then PC <- W JMPGE W Direct Jump if Greater Than If (F > G) then PC <- W JMPGE W Direct Jump if Greater or Equal If (F > G) then PC <- W JMPGE W Direct Jump if Greater or Equal If (F > G) then PC <- W JMPL W Direct Jump if Less Than If (F < G) then PC <- W JMPLE W Direct Jump if Less or Equal If (F > G) then PC <- W JMPNE W Direct Jump if Specified milliseconds have elapsed. If (millisstane) PC <- W JMPNIS W Direct Jump if specified milliseconds have elapsed. If (millisstane) PC <- W JMPUS W Direct Jump if specified milliseconds have elapsed. If (data=data2) PC <- W JMPDE W Direct Jump if two data arrays are equal. If (data=data2) PC <- W SKPBH Rr, Rr Skip next instruction if Register bit is high If (Rr & (1 << Rr)) skip SKPBL Rr, Rr Skip next instruction if Register bit is low If (I(Rr) & (1 << Rr)) skip DATA TRANSFER INSTRUCTIONS AREF K Analog reference selection of Constant AREF <- K DGTLOR Rr Digital write the Registers value DGTUN Rd Digital read into Register INB Rd, K In Port Constant Rd - P MOVB Rd, K Move Constant to Register Rd <- P MOVB Rd, Rr Move Between Registers Rd <- R OUT K, K Out Port Constant P <- K	RSHFTB	Rd, K	Logical Right Shift Register by Constant	Rd <- Rd << K
JMP W Direct Jump PC <- W JMPE W Direct Jump if Equal If (F = G) then PC <- W	RSHFTR	Rd, Rr	Logical Right Shift Register by Register	Rd <- Rd << Rr
JMPE W Direct Jump if Equal If (F = G) then PC <- W JMPG W Direct Jump if Greater Than If (F > G) then PC <- W JMPGE W Direct Jump if Greater or Equal If (F > G) then PC <- W JMPGE W Direct Jump if Greater or Equal If (F > G) then PC <- W JMPL W Direct Jump if Less Than If (F < G) then PC <- W JMPL W Direct Jump if Less Than If (F < G) then PC <- W JMPLE W Direct Jump if Less or Equal If (F <- G) then PC <- W JMPNE W Direct Jump if specified milliseconds have elapsed. If (millis>time) PC <- W JMPNS W Direct Jump if specified milliseconds have elapsed. If (millis>time) PC <- W JMPDS W Direct Jump if specified microseconds have elapsed. If (micros>time) PC <- W JMPDE W Direct Jump if two data arrays are equal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if specified microseconds have elapsed. If (micros>time) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are equal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are equal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are equal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are equal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are equal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays are eq	BRANCH INSTRUCTIONS			I
JMPG W Direct Jump if Greater Than If (F > G) then PC < W JMPGE W Direct Jump if Greater or Equal If (F >= G) then PC < W	JMP	W	Direct Jump	PC <- W
JMPGE W Direct Jump if Greater or Equal If (F > G) then PC < W JMPL W Direct Jump if Less Than If (F < G) then PC < W	JMPE	W	Direct Jump if Equal	If (F = G) then PC <- W
JMPL W Direct Jump if Less Than If (F < G) then PC <- W JMPLE W Direct Jump if Less or Equal If (F <= G) then PC <- W	JMPG	W	Direct Jump if Greater Than	If (F > G) then PC <- W
JMPLEWDirect Jump if Less or EqualIf (F <= G) then PC <- WJMPNEWDirect Jump if Not EqualIf (F != G) then PC <- W	JMPGE	W	Direct Jump if Greater or Equal	If (F >= G) then PC <- W
JMPNEWDirect Jump if Not EqualIf (F != 6) then PC <- WJMPMSWDirect Jump if specified milliseconds have elapsed.If (millis>time) PC <- W	JMPL	W	Direct Jump if Less Than	If (F < G) then PC <- W
JMPMSWDirect Jump if specified milliseconds have elapsed.If (millis>time) PC <- WJMPUSWDirect Jump if specified microseconds have elapsed.If (micros>time)PC <- W	JMPLE	W	Direct Jump if Less or Equal	If (F <= G) then PC <- W
JMPUSWDirect Jump if specified microseconds have elapsed.If (micros>time)PC <- WJMPDEWDirect Jump if two data arrays are equal.If (data=data2) PC <- W	JMPNE	W	Direct Jump if Not Equal	If (F!=G) then PC <- W
JMPDEWDirect Jump if two data arrays are equal.If (data=data2) PC <- WJMPDNEWDirect Jump if two data arrays are unequal.If (data=data2) PC <- W	JMPMS	W	Direct Jump if specified milliseconds have elapsed.	If (millis>time) PC <- W
JMPDNEWDirect Jump if two data arrays are unequal.If (data!=data2) PC <-WSKPBHRr, RrSkip next instruction if Register bit is highIf (Rr & (1 << Rr)) skip	JMPUS	W	Direct Jump if specified microseconds have elapsed.	If (micros>time)PC <- W
SKPBH Rr, Rr Skip next instruction if Register bit is high If (Rr & (1 << Rr)) skip SKPBL Rr, Rr Skip next instruction if Register bit is low If (!(Rr) & (1 << Rr)) skip DATA TRANSFER INSTRUCTIONS AREF K K Analog reference selection of Constant AREF <- K DGTLOB K Digital write a Constant value DGTLOR Rr Digital write the Registers value DGTLIN Rd Digital read into Register INB Rd, K In Port Constant Rd <- P INR Rd, Rr In Port Register Rd <- P MOVB Rd, K Move Constant to Register MOVR Rd, Rr Move Between Registers OUT K, K Out Port Constant P <- K OUT B Rr, K Out Port Register P <- K	JMPDE	W	Direct Jump if two data arrays are equal.	If (data=data2) PC <- W
SKPBLRr, RrSkip next instruction if Register bit is lowIf (!(Rr) & (1 << Rr)) skipDATA TRANSFER INSTRUCTIONSAREFKAnalog reference selection of ConstantAREF <- K	JMPDNE	W	Direct Jump if two data arrays are unequal.	If (data!=data2) PC <-W
DATA TRANSFER INSTRUCTIONS AREF K K Analog reference selection of Constant AREF <- K DGTLOB K Digital write a Constant value DGTLOR Rr Digital write the Registers value DGTLIN Rd Digital read into Register INB Rd, K In Port Constant Rd <- P INR Rd, Rr In Port Register Rd <- P MOVB Rd, K Move Constant to Register Rd <- K MOVR Rd, Rr Move Between Registers Rd <- Rr OUT K, K Out Port Constant P <- K OUT B Rr, K Out Port Register P <- K	SKPBH	Rr, Rr	Skip next instruction if Register bit is high	If (Rr & (1 << Rr)) skip
AREF K Analog reference selection of Constant AREF <- K DGTLOB K Digital write a Constant value DGTLOR Rr Digital write the Registers value DGTLIN Rd Digital read into Register INB Rd, K In Port Constant Rd <- P INR Rd, Rr In Port Register Rd <- P MOVB Rd, K Move Constant to Register Rd <- K MOVR Rd, Rr Move Between Registers Rd <- Rr OUT K, K Out Port Constant P <- K	SKPBL	Rr, Rr	Skip next instruction if Register bit is low	If (!(Rr) & (1 << Rr)) skip
DGTLOB K Digital write a Constant value DGTLOR Rr Digital write the Registers value DGTLIN Rd Digital read into Register INB Rd, K In Port Constant Rd <- P INR Rd, Rr In Port Register Rd <- P MOVB Rd, K Move Constant to Register Rd <- K MOVR Rd, Rr Move Between Registers Rd <- Rr OUT K, K Out Port Constant P <- K OUT P <- K	DATA TRANSFER INSTRUCT	IONS		
DGTLORRrDigital write the Registers valueDGTLINRdDigital read into RegisterINBRd, KIn Port ConstantRd <- P			Analog reference selection of Constant	AREF <- K
DGTLINRdDigital read into RegisterINBRd, KIn Port ConstantRd <- P	DGTLOB	K	Digital write a Constant value	
DGTLINRdDigital read into RegisterINBRd, KIn Port ConstantRd <- P	DGTLOR	Rr	Digital write the Registers value	
INR Rd, Rr In Port Register Rd <- P MOVB Rd, K Move Constant to Register Rd <- K	DGTLIN	Rd		
MOVBRd, KMove Constant to RegisterRd <- KMOVRRd, RrMove Between RegistersRd <- Rr	INB	Rd, K		Rd <- P
MOVR Rd, Rr Move Between Registers Rd <- Rr OUT K, K Out Port Constant P <- K	INR	Rd, Rr	In Port Register	Rd <- P
OUT K, K Out Port Constant P <- K OUTB Rr, K Out Port Register P <- K	MOVB	Rd, K	Move Constant to Register	Rd <- K
OUTB Rr, K Out Port Register P <- K	MOVR	Rd, Rr	Move Between Registers	Rd <- Rr
· · · · · · · · · · · · · · · · · · ·	OUT	K, K	Out Port Constant	P <- K
OUTBR K, Rr Out Port Constant P <- Rr	OUTB	Rr, K	Out Port Register	P <- K
	OUTBR	K, Rr	Out Port Constant	P <- Rr

OLITE	D- D-	Out Dart Darietan	D c D.
OUTR	Rr, Rr	Out Port Register	P <- Rr
POP	Rd	Pop Register from Stack	Rd <- STACK
PUSH	Rr	Push Register on Stack	STACK <- Rr
POPALL		Pop All Registers off Stack	Rs <- STACK
PSHALL		Push All Register on Stack	STACK <- Rs
PNMDB	K	Sets the pin mode of the pin in r0 via Constant	pinMode(r0, K)
PNMDR	Rr	Sets the pin mode of the pin in r0 via Rr	pinMode(r0, Rr)
ANLGIN		Analog read into register	
ANLGOB		Analog write the Register value	
ANLGOR		Analog write a Constant value	
GET	W	Get byte from Constant address	r1 <- STACK(K + r0)
PUT	W	Put a byte to Constant address	STACK(K + r0) <- r1
DATA	K	Save data array the specified Constant number	Data byte 128
WIPE	W	Null a complete data array at Constant address	Data array = null
MCU CONTROL INS	STRUCTIONS	•	
BEGB	K	Loop Counter Equals Constant. Save Address to LPA	LPAH:LPAL <- PC, LPT<-K
BEGR	Rr	Loop Counter Equals Register. Save Address to LPA	LPAH:LPAL <- PC, LPT<-K
LOOP		Increments LPC by One Compares if Equal to LPT	If (LPC+1 < LPT) P <- LPA
HALTB	К	Pause Execution for Constant Value in Milliseconds	
HALTR	Rr	Pause Execution for Register Value in Milliseconds	
HLTMB	К	Pause Execution for Constant Value in Microseconds	
HLTMR	Rr	Pause Execution for Register Value in Microseconds	
MILLIS	Rr	Put current Millis() to the place pointed.	Rr -> STACK <- Millis
MICROS	Rr	Put current Micros() to the place pointed.	Rr -> STACK <- Micros
NOP		Does nothing	
STOP		Stops program Execution	
SERIAL CONTROL I	NSTRUCTIONS	1 2 2	
AVAL	Rd	If Serial is Available, Register Returns Bytes in Buffer	If (available) Rd <- BFS
CLEAN		Clears All Waiting Bytes from Serial Buffer	Serial Clear
PRINT	Rr	Prints Register Contents to Serial	Serial Out <- Rr
PRINTCB	K	Prints Constant as Char to Serial	Serial Out Char <- K
PRINTCR	Rr	Prints Register as Char to Serial	Serial Out Char <- Rr
PRINTDC	W	Prints a data array in readable format till null byte	Serial Out Char <- &W
PRINTDB	W	Prints a entire data array in hex.	Serial Out Hex <- &W
READ	Rd	Reads Non-Blocking from Serial into Register	Rd <- Serial Read
SERCTRL	K	Turns Serial On if K > 0 and Off if K = 0	Serial <- K
WREAD	Rd	Reads Blocking from Serial into Register	Rd <- Serial Read

	Register Summary				
Address	Description	Specific Usage			
0	General Purpose Register	GET/PUT, pin select for analogWrite, digitalWrite, pinMode			
1	General Purpose Register	GET and PUT register			
2	General Purpose Register	MSB of address for JMPDE/JMPDNE			
3	General Purpose Register	Pointer for JMPMS/JMPUS to millis instance returned by MSINIT/USINIT			
		LSB of address for JMPDE/JMPDNE			
4	Jump Comparison Register	All Jump Comparisons/ Millis to wait MSB(Most Significant Byte)			
5	Jump Comparison Register	All Jump Comparisons / Millis to wait LSB(Least Significant Byte)			
6	Stack Pointer	PUSH/POP register			
7	Loop Register	Loop until register			
8	Loop Register	Loop return address			
9	Loop Register	Loop return address			
10	Loop Register	Loop counter			

Notes:

To select pin mode this is what the numbers represent:

0 = INPUT

1 = OUTPUT

0 = DEFAULT
1=INTERNAL
2=EXTERNAL
To compare data spaces/arrays copy one data space/array address into rC and rD,
And the other data space/array address into rE and rF.
To compare, copy the registers in question into registers 4 and 5

To select analog reference this is what the number represent:

2 = INPUT_PULLUP