ARTHMETIC AND LOGIC INSTRUCTIONS	Mnemonics	Operands	Description	Operation
ADDR	ARITHMETIC AND LOGIC IN	ISTRUCTIONS		•
ANDB	ADDB	Rd, K	Add Constant to Register	Rd <- Rd + K
DADUR	ADDR	Rd, Rr	Add two Registers	Rd <- Rd + Rr
DIVR	ANDB	Rd, K	Logical AND Register by Constant	Rd <- Rd & K
DECR Rd Rd Rd Deride two Registers Rd < Rd + Rf	ANDR	Rd, Rr	Logical AND two Registers	Rd <- Rd & Rr
DECR	DIVB	Rd, K	Divide Register by Constant	Rd <- Rd + K
MCR	DIVR	Rd, Rr	Divide two Registers	Rd <- Rd + Rr
MODB Rd, K Modulo Register by Constant Rd < Rd + K MODR Rd, Rr Modulo two Registers Rd < Rd - Rd - R MULB Rd, K Multiply two Registers Rd < Rd + K MULR Rd, R Multiply two Registers Rd < Rd + R MULR Rd, Rr Multiply two Registers Rd < Rd + R MOTT Rd Logical OR Treather Rd < Rd + R+ R ORB Rd, K Logical OR Treather Rd < Rd K ORB Rd, R Logical OR Treather Rd < Rd K ORB Rd, R Logical OR Two Registers Rd < Rd K BORD Rd, R Logical OR Two Registers Rd < Rd K-I BNDR Rd, R Logical ADR Register by Constant Rd < Rd K-I SUBB Rd, K Subtract two Registers Rd < Rd - Rr SWAP Rd, Rr Subtract two Registers Rd < Rd - Rr SWAP Rd, Rr Subtract two Registers Rd < Rd - Rr XOBB Rd, K Logical Register by Constant Rd < Rd - RR - RR - RR - RR - RR - RR -	DECR	Rd	Decrease Register by one	Rd <- Rd – 1
MODB	INCR	Rd	Increase Register by one	Rd <- Rd + 1
MULB	MODB	Rd, K	Modulo Register by Constant	Rd <- Rd + K
MULR Rd, Rr Multiply two Registers Rd < Rd + Rr NOT Rd Logical NOT Register Rd < Rd + Rr	MODR	Rd, Rr	Modulo two Registers	Rd <- Rd + Rr
DOT	MULB	Rd, K	Multiply Register by Constant	Rd <- Rd + K
ORB Rd, K Logical OR Register by Constant Rd < Rd Rr RnNDB Rd, K Random between Register and Constant Rd < Rd Rr RNDB Rd, K Random between Registers Rd < Rd Rr RNDB Rd, K Random between Registers Rd < Rd ? (K - 1) SUBB Rd, K Subtract two Registers Rd < Rd - Rd ? (K - 1) SUBB Rd, K Subtract two Registers Rd < Rd - Rd - Rd - Rd / Rr - 1) SUBB Rd, R Subtract two Registers Rd < Rd -	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) RNDR Rd, Rr Random between Registers Rd < Rd > Rd < Rd ? (K - 1) SUBB Rd, K Subtract tow Registers Rd < Rd × Rd ~ Rd · Rd × Rd · Rd × Rd × Rd × Rd × Rd ×	ORB	Rd, K	Logical OR Register by Constant	Rd <- Rd K
RNDR	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 - K SUBR Rd, Rr Subtract two Registers Rd < Rd - K Rd - K SWAP Rd, Rr Swap Rd to Rr and Rr to Rd Rd Rd - Rd - Rr - Rr < Rd XORB Rd, K Logical XOR Register by Constant Rd < Rd - Rd - Rr - Rr < Rd XORB Rd, K Logical XOR Register by Constant Rd < Rd - Rd - Rr Rd - Rd & Rd - Rd - Rd - Rd & Rd - Rd -	RNDB	Rd, K	Random between Register and Constant	Rd <- Rd ? (K - 1)
SUBB Rd, K Subtract Constant from Register Rd < Rd · K SUBR Rd, Rr Subtract two Registers Rd < Rd · K SUBR Rd, Rr Subtract two Registers Rd < Rd · Rr Rd · Rr SWAP Rd, Rr Swap Rd to Rr and Rr to Rd Rd Rd · Rr, Rr < Rd XORB Rd, K Logical XOR Register by Constant Rd < Rd · Rd · Rr K XORB Rd, K Logical XOR Register by Constant Rd < Rd · Rd · Rd · Rr SWAP Rd, K Logical XOR two Registers Constant Rd < Rd ·	RNDR		-	
SUBR Rd, Rr Subtract two Registers Rd < Rd - Rr SWAP Rd, Rr Swap Rd to Rr and Rr to Rd Rd < Rd - Rr R < Rd < Rd < Rd < Rr Rc < Rd < Rd < Rd < Rd < Rc Rd < Rd < R	SUBB	·		· ·
SWAP				
XORB Rd, K Logical XOR Register by Constant Rd < Rd ^ K K XORR Rd, Rr Logical XOR two Registers Rd < Rd ^ K K XORR Rd, Rr Logical XOR two Registers Rd < Rd ^ K Rd < Rd ^ K Rd	SWAP	Rd, Rr	Swap Rd to Rr and Rr to Rd	Rd <- Rr, Rr <- Rd
XORR	XORB	·	•	
BANDB Rd, K Boolean AND Register by Constant Rd <- Rd && K BANDR Rd, Rr Boolean AND two Registers Rd <- Rd && Rr BANDR Rd, Rr Boolean NOT wo Registers Rd <- Rd && Rr Rd Rd BOOlean NOT Register Rd <- Rd && Rr Rd Rd BOOlean NOT Register Rd <- Rd Rd Rd Rd Rd Rd Rd Rd Rd Rd Rd Rd Rd Rd	XORR			
BANDR Rd, Rr Boolean AND two Registers Rd <- Rd && Rr BNOT Rd Boolean NOT Register Rd <- IRd BORB Rd, K Boolean OR two Registers Rd <- IRd BORB Rd, K Boolean OR two Registers Rd <- Rd K BORB Rd, K Boolean OR two Register by Constant Rd <- Rd K BORB Rd, Rr Boolean OR Register by Constant Rd <- Rd K BORB Rd, Rr Boolean OR Register by Constant Rd <- Rd Rd <- Rd <- Rd Rd <- Rd <- Rd <- Rd Rd <- Rd <- Rd Rd <- Rd <- Rd Rd <- Rd <- Rd <- Rd Rd <-	BANDB	·		
BNOT Register Rd < Rd Boolean NOT Register Rd < Rd K BOORB Rd, K Boolean OR two Registers Rd < Rd K Rd < Rd K Rd BOORB Rd, Rd, Rr Boolean OR two Registers Rd < Rd K Rd < Rd K Rd < Rd Rr Rd < Rd	BANDR			
BORB Rd, K Boolean OR two Registers Rd < Rd K BORR Rd, Rr Boolean OR Register by Constant Rd < Rd K Rd < Rd < K K Rd < Rd < K K Rd < Rd < K K Rd < Rd < K K K Rd < Rd < K Rd & Logical Left Shift Register by Register Rd < Rd	BNOT		_	
BORR Rd, Rr Boolean OR Register by Constant Rd < Rd Rr BIT INSTRUCTIONS LSHFTB Rd, K Logical Left Shift Register by Constant Rd < Rd < K LSHFTR Rd, Rr Logical Left Shift Register by Register Rd < Rd < Rr RSHFTB Rd, K Logical Left Shift Register by Register Rd < Rd < Rd < Rf RSHFTB Rd, K Logical Right Shift Register by Constant Rd < Rd < Rd < Rd < Rd < Rd RSHFTR Rd, Rr Logical Right Shift Register by Register Rd < Rd < Rd < Rd < Rd RSHFTR Rd, Rd, Rr Logical Right Shift Register by Register Rd < Rd < Rd < Rd RACH INSTRUCTIONS CALL W Save program location to stack and Direct Jump STACK < PC < W JMPP W Direct Jump P PC < W JMPE W Direct Jump FEqual If (F = G) then PC < W JMPG W Direct Jump if Greater Than If (F < G) then PC < W JMPG W Direct Jump if Greater or Equal If (F > G) then PC < W JMPLE 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 (Mills-time) PC < W JMPNE W Direct Jump if Specified milliseconds have elapsed. If (Mills-time) PC < W JMPDE W Direct Jump if Specified milroseconds have elapsed. If (Mills-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 unequal. If (data=data2) PC < W JMPDE W Direct Jump if two data arrays are unequal. If (data=data2) PC < W JMPDE W Direct Jump if two data arrays are unequal. If (fata=data2) PC < W JMPDE RET RRY Skip next instruction if Register bit is high If (R & (1 < Rr)) skip SKPBH Rr, Rr Skip next instruction if Register bit is low If ([(RRY) & (1 < RRY)) skip DATA TRANSFER INSTRUCTIONS AREF K Analog reference selection of Constant AREF < K DGTLOR Rr Digital write the Registers value DGTLOR Rr Digital write the Registers value DGTLOR Rr Digital write the Registers value DGTLOR Rd, K In Port Constant to Register MOVR Rd, Rr Move Between Registers Rd < Rr			_	
BIT INSTRUCTIONS	BORR			• • •
LSHFTB Rd, K Logical Left Shift Register by Constant Rd < Rd < Rf LSHFTR Rd, Rr Logical Left Shift Register by Register Rd < Rd < Rf RSHFTB Rd, K Logical Right Shift Register by Constant Rd < Rd < Rf RSHFTB Rd, K Logical Right Shift Register by Constant Rd < Rd < Rf RSHFTR Rd, Rr Logical Right Shift Register by Register Rd < Rd < Rd < Rf RSHFTR Rd, Rr Logical Right Shift Register by Register Rd < Rd	BIT INSTRUCTIONS	,		
RSHFTB Rd, K Logical Right Shift Register by Constant Rd < Rd < K RSHFTR Rd, Rr Logical Right Shift Register by Register Rd < Rd < Rr BRANCH INSTRUCTIONS CALL W Save program location to stack and Direct Jump STACK < PC < W JMP W Direct Jump PC < W 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 or Equal If (F > G) then PC < W JMPL 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 or Equal If (F > G) then PC < W JMPNE W Direct Jump if Less or Equal If (F > G) then PC < W JMPNE 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 milliseconds 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 unequal. If (data=data2) PC < W RET Return to location on stack PC < STACK 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 DGTLOB K Digital write a Constant Value DGTLOR Rr Digital write the Registers value DGTLOR Rr Digital write the Register Rd < P MOVB Rd, K Move Constant to Register MOVB Rd, K In Port Constant to Register MOVB Rd, K Move Constant to Register		Rd, K	Logical Left Shift Register by Constant	Rd <- Rd << K
RSHFTB Rd, K Logical Right Shift Register by Constant Rd < Rd < K RSHFTR Rd, Rr Logical Right Shift Register by Register Rd < Rd < Rr BRANCH INSTRUCTIONS CALL W Save program location to stack and Direct Jump STACK < PC < W JMP W Direct Jump PC < W 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 or Equal If (F > G) then PC < W JMPL 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 or Equal If (F > G) then PC < W JMPNE W Direct Jump if Less or Equal If (F > G) then PC < W JMPNE 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 milliseconds 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 unequal. If (data=data2) PC < W RET Return to location on stack PC < STACK 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 DGTLOB K Digital write a Constant Value DGTLOR Rr Digital write the Registers value DGTLOR Rr Digital write the Register Rd < P MOVB Rd, K Move Constant to Register MOVB Rd, K In Port Constant to Register MOVB Rd, K Move Constant to Register	LSHFTR	Rd, Rr		Rd <- Rd << Rr
RSHFTR Rd, Rr Logical Right Shift Register by Register Rd <- Rd << Rr BRANCH INSTRUCTIONS CALL W Save program location to stack and Direct Jump STACK <- PC <- W JMP W Direct Jump PC <- W 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 Not Equal If (F <= G) then PC <- W JMPNE W Direct Jump if Not Equal If (F <= G) then PC <- W JMPNE W Direct Jump if specified milliseconds have elapsed. If (millis-stime) PC <- W JMPUS W Direct Jump if specified microseconds have elapsed. If (millis-stime) 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 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 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 unequal. If (data=data2) PC <- W JMPDNE W Direct Jump if two data arrays a	RSHFTB		Logical Right Shift Register by Constant	Rd <- Rd << K
CALL W Save program location to stack and Direct Jump STACK <- PC <- W JMPP W Direct Jump PC <- W 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 JMPG W Direct Jump if Greater or Equal If (F > G) then PC <- W JMPG JMPL 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 or Equal If (F < G) then PC <- W JMPL W Direct Jump if Specified milliseconds have elapsed. If (millis>time) PC <- W JMPUS W Direct Jump if specified milliseconds have elapsed. If (millis>time) PC <- W JMPDS 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 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 RET Return to location on stack PC <- STACK 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 Analog reference selection of Constant AREF <- K DGTLOB K Digital write a Constant value DGTLOR Rr Digital write the Registers value DGTLOR Rr Digital write the Register Selection Rd - P INR Rd, K In Port Constant Rd - P INR MOVB Rd, K MOVB Rd, K MOVB Rd, K MOVB BEWeen Registers Rd <- R	RSHFTR	Rd, Rr		Rd <- Rd << Rr
JMPE W Direct Jump PC <- W 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 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 Less Than 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 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 (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 RET Return to location on stack PC <- STACK SKPBH Rr, Rr Skip next instruction if Register bit is high If (Rr & (1 << Rr)) skip DATA TRANSFER INSTRUCTIONS AREF K Analog reference selection of Constant AREF <- K DGTLOB K Digital write a Constant value DGTLOR Rr Digital write the Registers value DGTLOR Rr Digital write the Registers value DGTLOR Rd Digital read into Register INB Rd, K In Port Register Rd <- P MOVB Rd, K Move Constant to Register Rd <- R MOVB Rd, Rr Move Between Registers	BRANCH INSTRUCTIONS	<u> </u>		
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 JMPL W Direct Jump if Less Than If (F > G) then PC <- W JMPL W Direct Jump if Less or Equal 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 JMPNE 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 unequal. If (data=data2) PC <- W RET Return to location on stack PC <- STACK 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 (IRr) & (1 << Rr)) skip DATA TRANSFER INSTRUCTIONS AREF K Analog reference selection of Constant AREF <- K DGTLOB K Digital write a Constant value DGTLIN Rd 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 INR Rd, R In Port Register Rd <- R MOVB Rd, K Move Constant to Register MOVB Rd, K Move Constant to Register MOVB Rd, Rr	CALL	W	Save program location to stack and Direct Jump	STACK <- PC <- W
JMPGWDirect Jump if Greater ThanIf (F > G) then PC <- WJMPGEWDirect Jump if Greater or EqualIf (F >= G) then PC <- W	JMP	W	Direct Jump	PC <- W
JMPGWDirect Jump if Greater ThanIf (F > G) then PC <- WJMPGEWDirect Jump if Greater or EqualIf (F >= G) then PC <- W	JMPE	W	Direct Jump if Equal	If (F = G) then PC <- W
JMPLWDirect Jump if Less ThanIf (F < G) then PC <- WJMPLEWDirect Jump if Less or EqualIf (F <= G) then PC <- W	JMPG	W	Direct Jump if Greater 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	JMPGE	W	Direct Jump if Greater or Equal	If (F >= G) then PC <- W
JMPNEWDirect Jump if Not EqualIf (F!=G) 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
JMPMS 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 JMPDNE W Direct Jump if two data arrays are unequal. If (data=data2) PC <- W RET Return to location on stack PC <- STACK 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 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 <- R MOVB Rd, K Move Constant to Register Rd <- Rr MOVB Rd, Rr Move Between Registers Rd <- Rr	JMPLE	W	Direct Jump if Less or Equal	If (F <= G) then PC <- W
JMPUS 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 RET Return to location on stack PC <- STACK 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 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 MOVB Rd, K Move Constant to Register Rd <- Rr MOVB Rd, Rr Move Between Registers Rd <- Rr	JMPNE	W	Direct Jump if Not Equal	If (F!=G) then PC <- W
JMPUS 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 RET Return to location on stack PC <- STACK 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 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 MOVB Rd, K Move Constant to Register Rd <- Rr MOVB Rd, Rr Move Between Registers Rd <- Rr	JMPMS	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 RET Return to location on stack PC <- STACK 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 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 MOVB Rd, K Move Constant to Register Rd <- Rr MOVB Rd, Rr Move Between Registers Rd <- Rr	JMPUS	W		
JMPDNEWDirect Jump if two data arrays are unequal.If (data!=data2) PC <-WRETReturn to location on stackPC <- STACK	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 (!(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 MOVB Rd, K Move Constant to Register Rd <- Rr	JMPDNE	W	Direct Jump if two data arrays are unequal.	If (data!=data2) PC <-W
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 Rd <- Rr	RET			PC <- STACK
DATA TRANSFER INSTRUCTIONSAREFKAnalog reference selection of ConstantAREF <- K	SKPBH	Rr, Rr	Skip next instruction if Register bit is high	If (Rr & (1 << Rr)) skip
DATA TRANSFER INSTRUCTIONSAREFKAnalog reference selection of ConstantAREF <- K	SKPBL			
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 <- R MOVR Rd, Rr Move Between Registers Rd <- Rr	DATA TRANSFER INSTRUCT	· ·	-	<u> </u>
DGTLOBKDigital write a Constant valueDGTLORRrDigital write the Registers valueDGTLINRdDigital read into RegisterINBRd, KIn Port ConstantRd <- P		I	Analog reference selection of Constant	AREF <- K
DGTLOR Rr Digital write the Registers value DGTLIN Rd Digital read into Register INB Rd, K In Port Constant Rd <- P		К	-	
DGTLINRdDigital read into RegisterINBRd, KIn Port ConstantRd <- P	DGTLOR	Rr		
INB Rd, K In Port Constant Rd <- P	DGTLIN	Rd		
INR Rd, Rr In Port Register Rd <- P	INB	Rd, K	-	Rd <- P
MOVB Rd, K Move Constant to Register Rd <- K MOVR Rd, Rr Move Between Registers Rd <- Rr	INR		In Port Register	Rd <- P
	MOVB	· ·	-	Rd <- K
OUT K, K Out Port Constant P <- K	MOVR	Rd, Rr	Move Between Registers	Rd <- Rr
	OUT	K, K	Out Port Constant	P <- K

OLITE	15.4	10.0.0.0	T 5 . 14
OUTB	Rr, K	Out Port Register	P <- K
OUTBR	K, Rr	Out Port Constant	P <- Rr
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	Rd	Analog read into register	
ANLGOB	K	Analog write the Register value	
ANLGOR	Rr	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 IN	STRUCTIONS		· · · · · · · · · · · · · · · · · · ·
BEGB	K	Loop Counter Equals Constant. Save Address to LPA	LPAH:LPAL <- PC, LPT<-
BEGR	Rr	Loop Counter Equals Register. Save Address to LPA	LPAH:LPAL <- PC, LPT<-
LOOP		Increments LPC by One Compares if Equal to LPT	If (LPC+1 < LPT) P <- LP/
HALTB	K	Pause Execution for Constant Value in Milliseconds	,
HALTR	Rr	Pause Execution for Register Value in Milliseconds	
HLTMB	K	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	1 * 0 10 * 1
STOP		Stops program Execution	
SERIAL CONTROL	INSTRUCTIONS	otopo program zneoddion	
AVAL	Rd	If Serial is Available, Register Returns Bytes in Buffer	If (available) Rd <- BFS
CLEAN	Tio Tio	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		
	W	Prints a data array in readable format till null byte	Serial Out Char <- &W Serial Out Hex <- &W
PRINTDB		Prints a entire data array in hex.	
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:

To select analog reference this is what the number represent:
0 = DEFAULT
1=INTERNAL
2=EXTERNAL

1 = OUTPUT

2 = INPUT_PULLUP

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 $5\,$