

# מערכות משובצות מחשב Computer Embedded Systems

## PIC 32 Assembly

## שפת אסמבלי

- אסמבלי הינה שפת סף שנועדה לתכנון מחשבים, מיקרו-מעבדים ומיקרו-בקרים.
- לכל מעבד (משפחת המעבדים) ישנה שפת אסמבלי שהינה ייחודית לו.
- השוני בין שפות אסמבלי יכול להתבטא בגישה לזיכרון, גישה למשתנים, ואפילו בתחביר השפה.
- בתוכנת אסמבלר ממירה את הפקודות בשפת אסמבלי לשפת.
- בשונה לשפת על, באסמבלי כל פקודה מתורגמת לפקודה אחת בלבד, בעוד שבשפות על רוב הפקודות מתורגמת למספר רב של פקודות בשפת המכונה.

## למה להשתמש בשפת אסמבלי ?

תכנות בשפת אסמבלי אמנם קשה יותר מתכנות בשפת על, אך תכנות באסמבלי מניב מספר יתרונות:

- כל פקודה באסמבלי מתורגמת לפקודה אחת בלבד ולכן קל יותר לחשב את מספר המחזורים המדויק הדרושים לביצוע התוכנית (יתרון חשוב במערכות זמן אמת).
- בשפת אסמבלי למתכנת יש שליטה מלאה על המתרחש במעבד, ובכך ניתן לבצע פעולות בצורה יעילה יותר.

## אסמבלי – גישה לרגיסטרים Integer

Register Name	Software Name (from regdef.h)	Use and Linkage
\$0		Always has the value 0.
\$at		Reserved for the assembler.
\$2..\$3	v0-v1	Used for expression evaluations and to hold the integer type function results. Also used to pass the static link when calling nested procedures.
\$4..\$7	a0-a3	Used to pass the first 4 words of integer type actual arguments, their values are not preserved across procedure calls.
\$8..\$15	t0-t7	Temporary registers used for expression evaluations; their values aren't preserved across procedure calls.
\$16..\$23	s0-s7	Saved registers. Their values must be preserved across procedure calls.
\$24..\$25	t8-t9	Temporary registers used for expression evaluations; their values aren't preserved across procedure calls.
\$26..\$27 or \$kt0..\$kt1	k0-k1	Reserved for the operating system kernel.
\$28 or \$gp	gp	Contains the global pointer.
\$29 or \$sp	sp	Contains the stack pointer.
\$30 or \$fp	fp	Contains the frame pointer (if needed); otherwise a saved register (like s0-s7).
\$31	ra	Contains the return address and is used for expression evaluation.

## אסמבלי – גישה לרגיסטרים Float

Register Name	Use and Linkage
\$f0..\$f2	Used to hold floating-point type function results (\$f0) and complex type function results (\$f0 has the real part, \$f2 has the imaginary part. \$f4..\$f10 Temporary registers, used for expression evaluation, whose values are not preserved across procedure calls.
\$f12..\$f14	Used to pass the first two single or double precision actual arguments, whose values are not preserved across procedure calls.
\$f16..\$f18	Temporary registers, used for expression evaluation, whose values are not preserved across procedure calls.
\$f20..\$f30	Saved registers, whose values must be preserved across procedure calls.

## פקודות אסמבלי סדר אלפבית שקף 1/5

Instruction	Description	Function
ADD	Integer Add	Rd = Rs + Rt
ADDI	Integer Add Immediate	Rt = Rs + Immed
ADDIU	Unsigned Integer Add Immediate	Rt = Rs +u Immed
ADDIUPC	Unsigned Integer Add Immediate to PC (MIPS16e™ only)	Rt = PC +u Immed
ADDU	Unsigned Integer Add	Rd = Rs +u Rt
AND	Logical AND	Rd = Rs & Rt
ANDI	Logical AND Immediate	Rt = Rs & (016    Immed)
B	Unconditional Branch (Assembler idiom for: BEQ r0, r0, offset)	PC += (int)offset
BAL	Branch and Link (Assembler idiom for: BGEZAL r0, offset)	GPR[31] = PC + 8 PC += (int)offset
BEQ	Branch On Equal	if Rs == Rt PC += (int)offset
BEQL	Branch On Equal Likely	if Rs == Rt PC += (int)offset else Ignore Next Instruction
BGEZ	Branch on Greater Than or Equal To Zero	if !Rs[31] PC += (int)offset
BGEZAL	Branch on Greater Than or Equal To Zero And Link	GPR[31] = PC + 8 if !Rs[31] PC += (int)offset
BGEZALL	Branch on Greater Than or Equal To Zero And Link Likely	GPR[31] = PC + 8 if !Rs[31] PC += (int)offset else Ignore Next Instruction
BGEZL	Branch on Greater Than or Equal To Zero Likely	if !Rs[31] PC += (int)offset else Ignore Next Instruction
BGTZ	Branch on Greater Than Zero	if !Rs[31] && Rs != 0 PC += (int)offset
BGTZL	Branch on Greater Than Zero Likely	if !Rs[31] && Rs != 0 PC += (int)offset else Ignore Next Instruction

## פקודות אסמבלי סדר אלפבית שקף 2/5

Instruction	Description	Function
BLEZ	Branch on Less Than or Equal to Zero	if $R_s[31] \wedge R_s == 0$ PC += (int)offset
BLEZL	Branch on Less Than or Equal to Zero Likely	if $R_s[31] \wedge R_s == 0$ PC += (int)offset else Ignore Next Instruction
BLTZ	Branch on Less Than Zero	if $R_s[31]$ PC += (int)offset
BLTZAL	Branch on Less Than Zero And Link	GPR[31] = PC + 8 if $R_s[31]$ PC += (int)offset
BLTZALL	Branch on Less Than Zero And Link Likely	GPR[31] = PC + 8 if $R_s[31]$ PC += (int)offset else Ignore Next Instruction
BLTZL	Branch on Less Than Zero Likely	if $R_s[31]$ PC += (int)offset else Ignore Next Instruction
BNE	Branch on Not Equal	if $R_s \neq R_t$ PC += (int)offset
BNEZL	Branch on Not Equal Likely	if $R_s \neq R_t$ PC += (int)offset else Ignore Next Instruction
BREAK	Breakpoint	Break Exception
CLO	Count Leading Ones	Rd = NumLeadingOnes(Rs)
CLE	Count Leading Zeros	Rd = NumLeadingZeros(Rs)
COFO	Coprocessor 0 Operation	See Software User's Manual
DEPC	Return from Debug Exception	PC = DEPC Exit Debug Mode
DI	Atomically Disable Interrupts	PC = Status; Status <sub>IE</sub> = 0
DIV	Divide	LO = (int)Rs / (int)Rt HI = (int)Rs % (int)Rt
DIVU	Unsigned Divide	LO = (uns)Rs / (uns)Rt HI = (uns)Rs % (uns)Rt
EHB	Execution Hazard Barrier	Stop instruction execution until execution hazards are cleared
EI	Atomically Enable Interrupts	Rt = Status; Status <sub>IE</sub> = 1
EREI	Return from Exception	if SR[2] PC = ErrorEPC else PC = EPC SR[1] = 0 SR[2] = 0 LL = 0
EXT	Extract Bit Field	Rt = ExtractField(Rs, pos, size)

## פקודות אסמבלי סדר אלפבית שקף 3/5

Instruction	Description	Function
INS	Insert Bit Field	Rt = InsertField(Rs, Rt, pos, size)
J	Unconditional Jump	PC = PC[31:28]    offset<<2
JAL	Jump and Link	GPR[31] = PC + 8 PC = PC[31:28]    offset<<2
JALR	Jump and Link Register	Rd = PC + 8 PC = Rs
JALR.HB	Jump and Link Register with Hazard Barrier	Like JALR, but also clears execution and instruction hazards
JALRC	Jump and Link Register Compact – do not execute instruction in jump delay slot (MIPS16e™ only)	Rd = PC + 2 PC = Rs
JR	Jump Register	PC = Rs
JR.HB	Jump Register with Hazard Barrier	Like JR, but also clears execution and instruction hazards
JRC	Jump Register Compact – do not execute instruction in jump delay slot (MIPS16e only)	PC = Rs
LB	Load Byte	Rt = (byte)Mem[Rs+offset]
LBU	Unsigned Load Byte	Rt = (ubyte)Mem[Rs+offset]
LH	Load Halfword	Rt = (half)Mem[Rs+offset]
LHU	Unsigned Load Halfword	Rt = (uhalf)Mem[Rs+offset]
LL	Load Linked Word	Rt = Mem[Rs+offset] LL = 1 LLAddr = Rs + offset
LUI	Load Upper Immediate	Rt = immediate << 16
LW	Load Word	Rt = Mem[Rs+offset]
LWPC	Load Word, PC relative	Rt = Mem[PC+offset]
LWL	Load Word Left	See Architecture Reference Manual
LWR	Load Word Right	See Architecture Reference Manual
MADD	Multiply-Add	HI   LO += (int)Rs * (int)Rt
MADDU	Multiply-Add Unsigned	HI   LO += (uns)Rs * (uns)Rt
MFC0	Move From Coprocessor 0	Rt = CPR[0, Rd, sel]
MFHI	Move From HI	Rd = HI
MFL0	Move From LO	Rd = LO
MOVN	Move Conditional on Not Zero	if Rt < 0 then Rd = Rs
MOVZ	Move Conditional on Zero	if Rt = 0 then Rd = Rs
MSUB	Multiply-Subtract	HI   LO -= (int)Rs * (int)Rt
MSUBU	Multiply-Subtract Unsigned	HI   LO -= (uns)Rs * (uns)Rt
MTC0	Move To Coprocessor 0	CPR[0, n, Sel] = Rt
MTHI	Move To HI	HI = Rs
MTL0	Move To LO	LO = Rs
MUL	Multiply with register write	HI   LO = unpredictable Rd = ((int)Rs * (int)Rt) <sub>31:0</sub>
MULT	Integer Multiply	HI   LO = (int)Rs * (int)Rt
MULTU	Unsigned Multiply	HI   LO = (uns)Rs * (uns)Rt

## פקודות אסמבלי סדר אלפבית שקף 4/5

Instruction	Description	Function
NOP	No Operation (Assembler idiom for: SLL r0, r0, r0)	
NOR	Logical NOR	$Rd = \neg(Rs \mid Rt)$
OR	Logical OR	$Rd = Rs \mid Rt$
ORI	Logical OR Immediate	$Rt = Rs \mid \text{Immed}$
RDHWR	Read Hardware Register	Allows unprivileged access to registers enabled by HWREna register
RDFGPR	Read GPR from Previous Shadow Set	$Rt = SGPR[\text{SRSCtl}_{123}, Rd]$
RESTORE	Restore registers and deallocate stack frame (MIPS16e™ only)	See Architecture Reference Manual
ROTR	Rotate Word Right	$Rd = RT_{24:1,0} \parallel RT_{31:25}$
ROTRV	Rotate Word Right Variable	$Rd = RT_{24:1,0} \parallel RT_{31:25}$
SAVE	Save registers and allocate stack frame (MIPS16e only)	See Architecture Reference Manual
SB	Store Byte	$(\text{byte})\text{Mem}[Ra+\text{offset}] = Rt$
SC	Store Conditional Word	if $IL = 0$ $\text{mem}[Ra+\text{offset}] = Rt$ $Rt = IL$
SDBBP	Software Debug Break Point	Trap to SW Debug Handler
SEB	Sign-Extend Byte	$Rd = (\text{byte})Rs$
SEH	Sign-Extend Half	$Rd = (\text{half})Rs$
SH	Store Half	$(\text{half})\text{Mem}[Ra+\text{offset}] = Rt$
SLL	Shift Left Logical	$Rd = Rt \ll sa$
SLLV	Shift Left Logical Variable	$Rd = Rt \ll Rs[4:0]$
SLE	Set on Less Than	if $(\text{int})Rs < (\text{int})Rt$ $Rd = 1$ else $Rd = 0$
SLEI	Set on Less Than Immediate	if $(\text{int})Rs < (\text{int})\text{Immed}$ $Rt = 1$ else $Rt = 0$
SLEIU	Set on Less Than Immediate Unsigned	if $(\text{uns})Rs < (\text{uns})\text{Immed}$ $Rt = 1$ else $Rt = 0$
SLEU	Set on Less Than Unsigned	if $(\text{uns})Rs < (\text{uns})Rt$ $Rd = 1$ else $Rd = 0$
SRA	Shift Right Arithmetic	$Rd = (\text{int})Rt \gg sa$
SRAV	Shift Right Arithmetic Variable	$Rd = (\text{int})Rt \gg Rs[4:0]$
SRL	Shift Right Logical	$Rd = (\text{uns})Rt \gg sa$
SRLV	Shift Right Logical Variable	$Rd = (\text{uns})Rt \gg Rs[4:0]$
SSNOP	Superscalar Inhibit No Operation	NOP
SUB	Integer Subtract	$Rt = (\text{int})Rs - (\text{int})Rd$
SUBW	Unsigned Subtract	$Rt = (\text{uns})Rs - (\text{uns})Rd$
SW	Store Word	$\text{Mem}[Ra+\text{offset}] = Rt$
SNL	Store Word Left	See Architecture Reference Manual

## פקודות אסמבלי סדר אלפבית שקף 5/5

Instruction	Description	Function
SWR	Store Word Right	See Architecture Reference Manual
SYNC	Synchronize	See Software User's Manual
SYSCALL	System Call	SystemCallException
TEQ	Trap if Equal	if $Rs == Rt$ TrapException
TEQI	Trap if Equal Immediate	if $Rs == (\text{int})\text{Immed}$ TrapException
TGE	Trap if Greater Than or Equal	if $(\text{int})Rs \geq (\text{int})Rt$ TrapException
TGEI	Trap if Greater Than or Equal Immediate	if $(\text{int})Rs \geq (\text{int})\text{Immed}$ TrapException
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	if $(\text{uns})Rs \geq (\text{uns})\text{Immed}$ TrapException
TGEU	Trap if Greater Than or Equal Unsigned	if $(\text{uns})Rs \geq (\text{uns})Rt$ TrapException
TLT	Trap if Less Than	if $(\text{int})Rs < (\text{int})Rt$ TrapException
TLTI	Trap if Less Than Immediate	if $(\text{int})Rs < (\text{int})\text{Immed}$ TrapException
TLTIU	Trap if Less Than Immediate Unsigned	if $(\text{uns})Rs < (\text{uns})\text{Immed}$ TrapException
TLTU	Trap if Less Than Unsigned	if $(\text{uns})Rs < (\text{uns})Rt$ TrapException
TNE	Trap if Not Equal	if $Rs != Rt$ TrapException
TNEI	Trap if Not Equal Immediate	if $Rs != (\text{int})\text{Immed}$ TrapException
WAIT	Wait for Interrupts	Stall until interrupt occurs
WRFGPR	Write to GPR in Previous Shadow Set	$SGPR[\text{SRSCtl}_{123}, Rd] = Rt$
WSBH	Word Swap Bytes Within Halfwords	$Rd = RT_{23:16} \parallel RT_{15:8} \parallel RT_{7:0}$ $\parallel RT_{31:24}$
XOR	Exclusive OR	$Rd = Rs \wedge Rt$
XORI	Exclusive OR Immediate	$Rt = Rs \wedge (\text{uns})\text{Immed}$
ZEB	Zero-extend byte (MIPS16e™ only)	$Rt = (\text{ubyte})Rs$
ZEH	Zero-extend half (MIPS16e only)	$Rt = (\text{half})Rs$

## פקודות אסמבלי – פעולות אריתמטיות

ARITHMETIC OPERATIONS		
ADD	Rd, Rs, Rt	$Rd = Rs + Rt$ (OVERFLOW TRAP)
ADDI	Rd, Rs, CONST16	$Rd = Rs + CONST16^{\pm}$ (OVERFLOW TRAP)
ADDIU	Rd, Rs, CONST16	$Rd = Rs + CONST16^{\pm}$
ADDU	Rd, Rs, Rt	$Rd = Rs + Rt$
CLO	Rd, Rs	$Rd = COUNTLEADINGONES(Rs)$
CLZ	Rd, Rs	$Rd = COUNTLEADINGZEROS(Rs)$
LA	Rd, LABEL	$Rd = ADDRESS(LABEL)$
LI	Rd, IMM32	$Rd = IMM32$
LUI	Rd, CONST16	$Rd = CONST16 \ll 16$
MOVE	Rd, Rs	$Rd = Rs$
NEGU	Rd, Rs	$Rd = -Rs$
SEB <sup>R2</sup>	Rd, Rs	$Rd = Rs_{7:0}^{\pm}$
SEH <sup>R2</sup>	Rd, Rs	$Rd = Rs_{15:0}^{\pm}$
SUB	Rd, Rs, Rt	$Rd = Rs - Rt$ (OVERFLOW TRAP)
SUBU	Rd, Rs, Rt	$Rd = Rs - Rt$

### מקרא פקודות אסמבלי

- Rd – DESTINATION REGISTER
- Rs, Rt – SOURCE OPERAND REGISTERS
- RA – RETURN ADDRESS REGISTER (R31)
- PC – PROGRAM COUNTER
- Acc – 64-BIT ACCUMULATOR
- Lo, Hi – ACCUMULATOR LOW (ACC<sub>31:0</sub>) AND HIGH (ACC<sub>63:32</sub>) PARTS
- ± – SIGNED OPERAND OR SIGN EXTENSION
- ∅ – UNSIGNED OPERAND OR ZERO EXTENSION
- :: – CONCATENATION OF BIT FIELDS
- R2 – MIPS32 RELEASE 2 INSTRUCTION
- DOTTED – ASSEMBLER PSEUDO-INSTRUCTION

## פקודות אסמבלי – פעולות הזזה

SHIFT AND ROTATE OPERATIONS		
ROTR <sup>R2</sup>	Rd, Rs, BITS5	$Rd = Rs_{BITS5-1:0} :: Rs_{31:BITS5}$
ROTRV <sup>R2</sup>	Rd, Rs, Rt	$Rd = Rs_{RT4:0-1:0} :: Rs_{31:RT4:0}$
SLL	Rd, Rs, SHIFT5	$Rd = Rs \ll SHIFT5$
SLLV	Rd, Rs, Rt	$Rd = Rs \ll Rt_{4:0}$
SRA	Rd, Rs, SHIFT5	$Rd = Rs^{\pm} \gg SHIFT5$
SRAV	Rd, Rs, Rt	$Rd = Rs^{\pm} \gg Rt_{4:0}$
SRL	Rd, Rs, SHIFT5	$Rd = Rs^{\emptyset} \gg SHIFT5$
SRLV	Rd, Rs, Rt	$Rd = Rs^{\emptyset} \gg Rt_{4:0}$

### מקרא פקודות אסמבלי

- Rd – DESTINATION REGISTER
- Rs, Rt – SOURCE OPERAND REGISTERS
- RA – RETURN ADDRESS REGISTER (R31)
- PC – PROGRAM COUNTER
- Acc – 64-BIT ACCUMULATOR
- Lo, Hi – ACCUMULATOR LOW (ACC<sub>31:0</sub>) AND HIGH (ACC<sub>63:32</sub>) PARTS
- ± – SIGNED OPERAND OR SIGN EXTENSION
- ∅ – UNSIGNED OPERAND OR ZERO EXTENSION
- :: – CONCATENATION OF BIT FIELDS
- R2 – MIPS32 RELEASE 2 INSTRUCTION
- DOTTED – ASSEMBLER PSEUDO-INSTRUCTION

## פקודות אסמבלי – פעולות לוגיות

LOGICAL AND BIT-FIELD OPERATIONS		
AND	Rd, Rs, Rt	$Rd = Rs \& Rt$
ANDI	Rd, Rs, CONST16	$Rd = Rs \& CONST16^{\circ}$
EXT <sup>R2</sup>	Rd, Rs, P, S	$Rs = R_{Sp-5:1P}^{\circ}$
INS <sup>R2</sup>	Rd, Rs, P, S	$R_{Dp-5:1P} = R_{S5:10}$
NO <sup>OP</sup>		No-OP
NOR	Rd, Rs, Rt	$Rd = \sim(Rs   Rt)$
NOT	Rd, Rs	$Rd = \sim Rs$
OR	Rd, Rs, Rt	$Rd = Rs   Rt$
ORI	Rd, Rs, CONST16	$Rd = Rs   CONST16^{\circ}$
WSBH <sup>R2</sup>	Rd, Rs	$Rd = R_{S23:16} :: R_{S31:24} :: R_{S7:0} :: R_{S15:8}$
XOR	Rd, Rs, Rt	$Rd = Rs \oplus Rt$
XORI	Rd, Rs, CONST16	$Rd = Rs \oplus CONST16^{\circ}$

### מקרא פקודות אסמבלי

Rd	– DESTINATION REGISTER
Rs, Rt	– SOURCE OPERAND REGISTERS
RA	– RETURN ADDRESS REGISTER (R31)
PC	– PROGRAM COUNTER
Acc	– 64-BIT ACCUMULATOR
Lo, Hi	– ACCUMULATOR LOW (ACC <sub>31:0</sub> ) AND HIGH (ACC <sub>63:32</sub> ) PARTS
±	– SIGNED OPERAND OR SIGN EXTENSION
∅	– UNSIGNED OPERAND OR ZERO EXTENSION
::	– CONCATENATION OF BIT FIELDS
R2	– MIPS32 RELEASE 2 INSTRUCTION
DOTTED	– ASSEMBLER PSEUDO-INSTRUCTION

## פקודות אסמבלי – פעולות התניה

CONDITION TESTING AND CONDITIONAL MOVE OPERATIONS		
MOVN	Rd, Rs, Rt	IF $Rt = 0$ , $Rd = Rs$
MOVZ	Rd, Rs, Rt	IF $Rt = 0$ , $Rd = Rs$
SLT	Rd, Rs, Rt	$Rd = (Rs^+ < Rt^+) ? 1 : 0$
SLTI	Rd, Rs, CONST16	$Rd = (Rs^+ < CONST16^+) ? 1 : 0$
SLTIU	Rd, Rs, CONST16	$Rd = (Rs^{\circ} < CONST16^{\circ}) ? 1 : 0$
SLTU	Rd, Rs, Rt	$Rd = (Rs^{\circ} < Rt^{\circ}) ? 1 : 0$

### מקרא פקודות אסמבלי

Rd	– DESTINATION REGISTER
Rs, Rt	– SOURCE OPERAND REGISTERS
RA	– RETURN ADDRESS REGISTER (R31)
PC	– PROGRAM COUNTER
Acc	– 64-BIT ACCUMULATOR
Lo, Hi	– ACCUMULATOR LOW (ACC <sub>31:0</sub> ) AND HIGH (ACC <sub>63:32</sub> ) PARTS
±	– SIGNED OPERAND OR SIGN EXTENSION
∅	– UNSIGNED OPERAND OR ZERO EXTENSION
::	– CONCATENATION OF BIT FIELDS
R2	– MIPS32 RELEASE 2 INSTRUCTION
DOTTED	– ASSEMBLER PSEUDO-INSTRUCTION

## פקודות אסמבלי – פעולות כפל וחילוק

MULTIPLY AND DIVIDE OPERATIONS		
DIV	Rs, Rt	Lo = $Rs^+ / Rt^+$ ; Hi = $Rs^+ \text{ MOD } Rt^+$
DIVU	Rs, Rt	Lo = $Rs^\circ / Rt^\circ$ ; Hi = $Rs^\circ \text{ MOD } Rt^\circ$
MADD	Rs, Rt	Acc += $Rs^+ \times Rt^+$
MADDU	Rs, Rt	Acc += $Rs^\circ \times Rt^\circ$
MSUB	Rs, Rt	Acc -= $Rs^+ \times Rt^+$
MSUBU	Rs, Rt	Acc -= $Rs^\circ \times Rt^\circ$
MUL	Rd, Rs, Rt	Rd = $Rs^+ \times Rt^+$
MULT	Rs, Rt	Acc = $Rs^+ \times Rt^+$
MULTU	Rs, Rt	Acc = $Rs^\circ \times Rt^\circ$

### מקרא פקודות אסמבלי

- Rd – DESTINATION REGISTER
- Rs, Rt – SOURCE OPERAND REGISTERS
- RA – RETURN ADDRESS REGISTER (R31)
- PC – PROGRAM COUNTER
- Acc – 64-BIT ACCUMULATOR
- Lo, Hi – ACCUMULATOR LOW (ACC<sub>31:0</sub>) AND HIGH (ACC<sub>63:32</sub>) PARTS
- ± – SIGNED OPERAND OR SIGN EXTENSION
- ∅ – UNSIGNED OPERAND OR ZERO EXTENSION
- :: – CONCATENATION OF BIT FIELDS
- R2 – MIPS32 RELEASE 2 INSTRUCTION
- DOTTED – ASSEMBLER PSEUDO-INSTRUCTION

## פקודות אסמבלי – פעולות Accumulator

### מקרא פקודות אסמבלי

ACCUMULATOR ACCESS OPERATIONS		
MFHI	Rd	Rd = Hi
MFLO	Rd	Rd = Lo
MTHI	Rs	Hi = Rs
MTLO	Rs	Lo = Rs

- Rd – DESTINATION REGISTER
- Rs, Rt – SOURCE OPERAND REGISTERS
- RA – RETURN ADDRESS REGISTER (R31)
- PC – PROGRAM COUNTER
- Acc – 64-BIT ACCUMULATOR
- Lo, Hi – ACCUMULATOR LOW (ACC<sub>31:0</sub>) AND HIGH (ACC<sub>63:32</sub>) PARTS
- ± – SIGNED OPERAND OR SIGN EXTENSION
- ∅ – UNSIGNED OPERAND OR ZERO EXTENSION
- :: – CONCATENATION OF BIT FIELDS
- R2 – MIPS32 RELEASE 2 INSTRUCTION
- DOTTED – ASSEMBLER PSEUDO-INSTRUCTION

## פקודות אסמבלי – פעולות ATOMIC-RMW

ATOMIC READ-MODIFY-WRITE OPERATIONS		
LL	Rd, off16(Rs)	Rd = MEM32(Rs + off16 <sup>+</sup> ); LINK
SC	Rd, off16(Rs)	IF ATOMIC, MEM32(Rs + off16 <sup>+</sup> ) = Rd; Rd = ATOMIC ? 1 : 0

## פקודות אסמבלי – פעולות קפיצה

JUMPS AND BRANCHES (NOTE: ONE DELAY SLOT)		
B	OFF18	PC += OFF18 <sup>±</sup>
BAL	OFF18	RA = PC + 8, PC += OFF18 <sup>±</sup>
BEQ	Rs, Rt, OFF18	IF Rs = Rt, PC += OFF18 <sup>±</sup>
BEQZ	Rs, OFF18	IF Rs = 0, PC += OFF18 <sup>±</sup>
BGEZ	Rs, OFF18	IF Rs ≥ 0, PC += OFF18 <sup>±</sup>
BGEZAL	Rs, OFF18	RA = PC + 8; IF Rs ≥ 0, PC += OFF18 <sup>±</sup>
BGTZ	Rs, OFF18	IF Rs > 0, PC += OFF18 <sup>±</sup>
BLEZ	Rs, OFF18	IF Rs ≤ 0, PC += OFF18 <sup>±</sup>
BLTZ	Rs, OFF18	IF Rs < 0, PC += OFF18 <sup>±</sup>
BLTZAL	Rs, OFF18	RA = PC + 8; IF Rs < 0, PC += OFF18 <sup>±</sup>
BNE	Rs, Rt, OFF18	IF Rs ≠ Rt, PC += OFF18 <sup>±</sup>
BNEZ	Rs, OFF18	IF Rs ≠ 0, PC += OFF18 <sup>±</sup>
J	ADDR28	PC = PC <sub>31:28</sub> :: ADDR28 <sup>⊙</sup>
JAL	ADDR28	RA = PC + 8; PC = PC <sub>31:28</sub> :: ADDR28 <sup>⊙</sup>
JALR	Rd, Rs	RD = PC + 8; PC = Rs
JR	Rs	PC = Rs

### מקרא פקודות אסמבלי

- Rd – DESTINATION REGISTER
- Rs, Rt – SOURCE OPERAND REGISTERS
- RA – RETURN ADDRESS REGISTER (R31)
- PC – PROGRAM COUNTER
- Acc – 64-BIT ACCUMULATOR
- Lo, Hi – ACCUMULATOR LOW (ACC<sub>31:0</sub>) AND HIGH (ACC<sub>63:32</sub>) PARTS
- ± – SIGNED OPERAND OR SIGN EXTENSION
- ∅ – UNSIGNED OPERAND OR ZERO EXTENSION
- :: – CONCATENATION OF BIT FIELDS
- R2 – MIPS32 RELEASE 2 INSTRUCTION
- DOTTED – ASSEMBLER PSEUDO-INSTRUCTION

## פקודות אסמבלי – פעולות טעינה/שמירה

LOAD AND STORE OPERATIONS		
LB	Rd, OFF16(Rs)	RD = MEM8(Rs + OFF16 <sup>±</sup> ) <sup>⊙</sup>
LBU	Rd, OFF16(Rs)	RD = MEM8(Rs + OFF16 <sup>±</sup> ) <sup>⊙</sup>
LH	Rd, OFF16(Rs)	RD = MEM16(Rs + OFF16 <sup>±</sup> ) <sup>±</sup>
LHU	Rd, OFF16(Rs)	RD = MEM16(Rs + OFF16 <sup>±</sup> ) <sup>⊙</sup>
LW	Rd, OFF16(Rs)	RD = MEM32(Rs + OFF16 <sup>±</sup> ) <sup>±</sup>
LWL	Rd, OFF16(Rs)	RD = LOADWORDLEFT(Rs + OFF16 <sup>±</sup> ) <sup>±</sup>
LWR	Rd, OFF16(Rs)	RD = LOADWORDRIGHT(Rs + OFF16 <sup>±</sup> ) <sup>±</sup>
SB	Rs, OFF16(Rt)	MEM8(Rt + OFF16 <sup>±</sup> ) = Rs <sub>7:0</sub>
SH	Rs, OFF16(Rt)	MEM16(Rt + OFF16 <sup>±</sup> ) = Rs <sub>15:0</sub>
SW	Rs, OFF16(Rt)	MEM32(Rt + OFF16 <sup>±</sup> ) = Rs
SWL	Rs, OFF16(Rt)	STOREWORDLEFT(Rt + OFF16 <sup>±</sup> , Rs)
SWR	Rs, OFF16(Rt)	STOREWORDRIGHT(Rt + OFF16 <sup>±</sup> , Rs)
ULW	Rd, OFF16(Rs)	RD = UNALIGNED_MEM32(Rs + OFF16 <sup>±</sup> ) <sup>±</sup>
USW	Rs, OFF16(Rt)	UNALIGNED_MEM32(Rt + OFF16 <sup>±</sup> ) = Rs

### מקרא פקודות אסמבלי

- Rd – DESTINATION REGISTER
- Rs, Rt – SOURCE OPERAND REGISTERS
- RA – RETURN ADDRESS REGISTER (R31)
- PC – PROGRAM COUNTER
- Acc – 64-BIT ACCUMULATOR
- Lo, Hi – ACCUMULATOR LOW (ACC<sub>31:0</sub>) AND HIGH (ACC<sub>63:32</sub>) PARTS
- ± – SIGNED OPERAND OR SIGN EXTENSION
- ∅ – UNSIGNED OPERAND OR ZERO EXTENSION
- :: – CONCATENATION OF BIT FIELDS
- R2 – MIPS32 RELEASE 2 INSTRUCTION
- DOTTED – ASSEMBLER PSEUDO-INSTRUCTION

## תוכנית לדוגמא: הצגת 0x55 על גבי LED

```
#include <p32xxx.h>
.global main
.text
.set noreorder
.end main /* directive that marks symbol 'main' as function in ELF output*/
main:
nop /* No Operation*/
andi $s0,$s0,0 /* Logical AND with immediate value, s0 ← s0 & 0 */
sw $s0,TRISF /* Store Word TRISF, TRISF ← s0 */
sw $s0,TRISE /* Store Word TRISE, TRISE ← s0 */
sw $s0,TRISD /* Store Word TRISD, TRISD ← s0 */
ori $s0,$s0,0x4 /* Logical OR with immediate value, s0 ← s0 | 0x4 */
sw $s0,PORTF /* Store Word PORTF, PORTF ← s0, Decoder 3x8 LED Select */
li $2,0x55 /* Load Immediate, v0 ← 0x55 */
sw $2,PORTE /* Store Word PORTE, PORTE ← v0 */
li $2,0x10 /* Load Immediate, v0 ← 0x10 */
sw $2,PORTD /* Store Word PORTD, PORTD ← v0, Enable Pulse 1→0 */
li $2,0x00 /* Load Immediate, v0 ← 0x00 */
sw $2,PORTD /* Store Word PORTD, PORTD ← v0 */
endless: j endless /* Endless Loop */
nop
.end main /* directive that marks end of 'main' function and registers size in ELF output*/
```

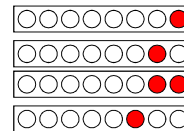


## תוכנית לדוגמא: מונה מעלה על גבי LED

```
#include <p32xxx.h>
.global main
.text
.set noreorder
.end main /* directive that marks symbol 'main' as function in ELF output*/
main:
nop /* No Operation*/
andi $s0,$s0,0 /* Logical AND with immediate value, s0 ← s0 & 0 */
sw $s0,TRISF /* Store Word TRISF, TRISF ← s0 */
sw $s0,TRISE /* Store Word TRISE, TRISE ← s0 */
sw $s0,TRISD /* Store Word TRISD, TRISD ← s0 */
ori $s0,$s0,0x4 /* Logical OR with immediate value, s0 ← s0 | 0x4 */
sw $s0,PORTF /* Store Word PORTF, PORTF ← s0, Decoder 3x8 LED Select */
li $3,0 /* Load Immediate, v1 ← 0 */

endless: /* Endless Loop */
jal print /* Jump And Link print */
nop /* No Operation*/
addi $3,$3,1 /* Integer ADD Immediate, v1 ← v1 + 1 */
jal delay /* Jump And Link delay */
nop /* No Operation*/
j endless /* Unconditional JUMP endless */
nop /* No Operation*/

.end main /* directive that marks end of 'main' function and registers size in ELF output*/
```



```

.ent delay          /* Delay Loop */
delay:
    andi $s1,$s1,0    /* Logical AND with immediate value, s1 ← s1 & 0 */
    andi $s2,$s2,0    /* Logical AND with immediate value, s2← s2& 0 */
    ori $s2,$s2,0xffff /* Logical OR with immediate value, s2← s2| 0xffff */
x:
    addi $s1,$s1,1    /* Integer ADD Immediate, s1← s1+ 1 */
    bne $s1,$s2,x     /* Branch On Not Equal, if s1 ≠ s2, jump to x */
    nop               /* No Operation*/
/* return to caller */
    jr $ra           /* Jump Register, Return from function */
    nop             /* No Operation*/
.end delay          /* End Delay Loop*/

.ent print          /* Print Loop*/
print:
    sw $3,PORTE      /* Store Word PORTE, PORTE ← v0 */
    li $2,0x10       /* Load Immediate, v1 ← 0x10 */
    sw $2,PORTD      /* Store Word PORTD, PORTD ← v1, Enable Pulse 1→0 */
    li $2,0x00       /* Load Immediate, v1 ← 0x00 */
    sw $2,PORTD      /* Store Word PORTD, PORTD ← v1 */
/* return to caller */
    jr $ra           /* Jump Register, Return from function */
    nop             /* No Operation*/
.end print         /*End Print Loop*/

```

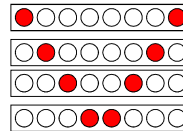
```

#include <p32xxx.h>
.global main
.global num
.data
num:
.byte 0x81,0x42,0x24,0x18
.text
.set noreorder
.ent main /* directive that marks symbol 'main' as function in ELF output*/

main:
    nop             /* No Operation*/
    andi $s0,$s0,0 /* Logical AND with immediate value, s0 ← s0 & 0 */
    sw $s0,TRISF   /* Store Word TRISF, TRISF ← s0 */
    sw $s0,TRISE   /* Store Word TRISE, TRISE ← s0 */
    sw $s0,TRISD   /* Store Word TRISD, TRISD ← s0 */
    ori $s0,$s0,0x4 /* Logical OR with immediate value, s0 ← s0 | 0x4 */
    sw $s0,PORTF   /* Store Word PORTF, PORTF ← s0, Decoder 3x8 LED Select */

start:
    li $s0,0x0     /* Load Immediate, s0 ← 0x0 */
    la $s4,num     /* Load Address, s4 ← [num] */
st_1:
    lb $3,($s4)    /* Load Byte, v1 ← byte(s4) */
    jal print      /* Jump And Link print */
    nop           /* No Operation*/
    jal delay     /* Jump And Link delay */
    nop           /* No Operation*/
    addi $s4,$s4,1 /* Integer ADD Immediate, s4 ← s4 + 1 */

```



```

addi $s0,$s0,1 /* Integer ADD Immediate, s0 ← s0 + 1 */
bne $s0,4,st_1 /* Branch On Not Equal, if s0 ≠ 4, jump to st_1 */
nop /* No Operation*/
j start /* Unconditional JUMP start*/
nop /* No Operation*/
.end main /* directive that marks end of 'main' function and registers size in ELF output*/
.ent delay /* Delay Loop */
delay:
    andi $s1,$s1,0 /* Logical AND with immediate value, s1 ← s1 & 0 */
    andi $s2,$s2,0 /* Logical AND with immediate value, s2 ← s2 & 0 */
    ori $s2,$s2,0xffff /* Logical OR with immediate value, s2 ← s2 | 0xffff */
x: addi $s1,$s1,1 /* Integer ADD Immediate, s0 ← s0 + 1 */
    bne $s1,$s2,x /* Branch On Not Equal, if s1 ≠ s2, jump to x*/
    nop /* No Operation*/
/* return to caller */
jr $ra /* Jump Register, Return from function */
nop /* No Operation*/
.end delay /* End Delay Loop */
.ent print /* Print Function*/
print: sw $3,PORTE /* Store Word PORTE, PORTE ← v1 */
    li $2,0x10 /* Load Immediate, v0 ← 0x10 */
    sw $2,PORTD /* Store Word PORTD, PORTD ← v1, Enable Pulse 1→0 */
    li $2,0x00 /* Load Immediate, v0 ← 0x00 */
    sw $2,PORTD /* Store Word PORTD, PORTD ← v1 */
/* return to caller */
jr $ra /* Jump Register, Return from function */
nop /* No Operation*/
.end print /* End Print Function*/

```

### מונה מעלה \ מטה בתלות במפסק

```

#include <p32xxxx.h>
.global main
.text
.set noreorder
.end main /* directive that marks symbol 'main' as function in ELF output*/
main:
    nop /* No Operation*/
    andi $s0,$s0,0 /* Logical AND with immediate value, s0 ← s0 & 0 */
    sw $s0,TRISF /* Store Word TRISF, TRISF ← s0 */
    sw $s0,TRISD /* Store Word TRISD, TRISD ← s0 */
    li $7,1 /* Load Immediate, a3 ← 1 */
    li $3,0 /* Load Immediate, v1 ← 0 */

start: /* Endless Loop */
    li $2,0xff /* Load Immediate, v0 ← 0xff */
    sb $2,TRISE /* Store Byte TRISE, TRISE ← v0 */
    li $2,0x3 /* Load Immediate, v0 ← 0x3 */
    sb $2,PORTF /* Store Byte PORTF, PORTF ← v0, Decoder 3x8 Select Switches */
    jal in_sw /* Jump And Link in_sw*/
    nop /* No Operation*/
    lb $2,PORTE /* Load Byte, v0 ← byte(PORTE) */
    andi $2,$2,1 /* Logical AND with immediate value, v0 ← v0 & 1 */
    bne $2,1,down /* Branch On Not Equal, if v0 ≠ 1, jump to down*/
    nop /* No Operation*/
    addi $3,$3,1 /* Integer ADD Immediate, v1 ← v1 + 1 */
    j sof /* Unconditional JUMP sof*/
    nop /* No Operation*/

```

```

down:
  sub $3,$3,$7          /* Integer Subtract, v1 ← v1 – a3 */
sof:
  jal print             /* Jump And Link print*/
  nop                  /* No Operation*/
  jal delay            /* Jump And Link delay*/
  nop                  /* No Operation*/
  j start             /* Unconditional JUMP start */
  nop                  /* No Operation*/
.end main /* directive that marks end of 'main' function and registers size in ELF output*/
.ent delay
delay:
  andi $s1,$s1,0       /* Logical AND with immediate value, s1 ← s1 & 0 */
  andi $s2,$s2,0       /* Logical AND with immediate value, s2 ← s2 & 0 */
  ori $s2,$s2,0xffff   /* Logical OR with immediate value, s2 ← s2 | 0xffff */
x:
  addi $s1,$s1,1       /* Integer ADD Immediate, s1 ← s1 + 1 */
  bne $s1,$s2,x        /* Branch On Not Equal, if s1 ≠ s2, jump to x*/
  nop                  /* No Operation*/
/* return to caller */
  jr $ra              /* Jump Register, Return from function */
  nop                  /* No Operation*/
.end delay

```

```

.ent print
print:
  li $2,0x4           /* Load Immediate, v0 ← 0x4 */
  sw $2,PORTF        /* Store Word PORTF, PORTF ← s0 */
  li $2,0             /* Load Immediate, v0 ← 0x0 */
  sw $2,TRISE        /* Store Word TRISE, TRISE ← s0 */
  sw $3,PORTE        /* Store Word PORTE, PORTE ← s1 */
in_sw:
  li $2,0x10         /* Load Immediate, v0 ← 0x10 */
  sw $2,PORTD        /* Store Word PORTD, PORTD ← v1, Enable Pulse 1→0 */
  li $2,0x00         /* Load Immediate, v0 ← 0x00 */
  sw $2,PORTD        /* Store Word PORTD, PORTD ← v1 */
/* return to caller */
  jr $ra             /* Jump Register, Return from function */
  nop                /* No Operation*/
.end print

```