Espressif Esp32

Assembler Summery

# Load instructions

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| L8UI | 8-bit unsigned load (8-bit offset) |  |
| L16SI | 16-bit signed load (8-bit shifted offset) |  |
| L16UI | 16-bit unsigned load (8-bit shifted offset) |  |
| L32I | 32-bit load (8-bit shifted offset) |  |
| L32R | 32-bit load PC-relative (16-bit negative word offset) |  |

# Store instructions

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| S8I | 8-bit store (8-bit offset) |  |
| S16I | 16-bit store (8-bit shifted offset) |  |
| S32I | 32-bit store (8-bit shifted offset) |  |

# Memory ordening

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| MEMW | Order memory accesses before with memory access after |  |
| EXTW | Order all external effects before with all external effects after |  |

# Jump, Call

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| CALL0 | Call subroutine, PC-relative |  |
| CALLX0 | Call subroutine, address in register |  |
| RET | Unconditional jump, PC-relative |  |
| J | Unconditional jump, address in register |  |
| JX | Subroutine return—jump to return address. Used to return from a routine called by CALL0/CALLX0. |  |

# Conditional Branch

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| BALL | Branch if all of masked bits set |  |
| BNALL | Branch if not all of masked bits set |  |
| BANY | Branch if any of masked bits set |  |
| BNONE | Branch if none of masked bits set (All Clear) |  |
| BBC | Branch if bit clear |  |
| BBCI | Branch if bit clear immediate |  |
| BBS | Branch if bit set |  |
| BBSI | Branch if bit set immediate |  |
| BEQ | Branch if equal |  |
| BEQI | Branch if equal immediate |  |
| BEQZ | Branch if equal to zero |  |
| BNE | Branch if not equal |  |
| BNEI | Branch if not equal immediate |  |
| BNEZ | Branch if not equal to zero |  |
| BGE | Branch if greater than or equal |  |
| BGEI | Branch if greater than or equal immediate |  |
| BGEU | Branch if greater than or equal unsigned |  |
| BGEUI | Branch if greater than or equal unsigned immediate |  |
| BGEZ | Branch if greater than or equal to zero |  |
| BLT | Branch if less than |  |
| BLTI | Branch if less than immediate |  |
| BLTU | Branch if less than Unsigned |  |
| BLTUI | Branch if less than unsigned immediate |  |
| BLTZ | Branch if less than zero |  |

# Move

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| MOVI | Load register with 12-bit signed constant |  |
| MOVEQZ | Conditional move if zero |  |
| MOVGEZ | Conditional move if greater than or equal to zero |  |
| MOVLTZ | Conditional move if less than zero |  |
| MOVNEZ | Conditional move if non-zero |  |

# Arithmetic

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| ADDI | Add signed constant to register | AR[t] ← AR[s] + (imm87 24||imm8) |
| ADDMI | Add signed constant shifted by 8 to register | AR[t] ← AR[s] + (imm87 16||imm8||08) |
| ADD | Add two registers | AR[r] ← AR[s] + AR[t] |
| ADDX2 | Add register to register shifted by 1 | AR[r] ← (AR[s]30..0 || 0) + AR[t] |
| ADDX4 | Add register to register shifted by 2 | AR[r] ← (AR[s]29..0 || 02) + AR[t] |
| ADDX8 | Add register to register shifted by 3 | AR[r] ← (AR[s]28..0 || 03) + AR[t] |
| SUB | Subtract two registers | AR[r] ← AR[s] − AR[t] |
| SUBX2 | Subtract register from register shifted by 1 | AR[r] ← (AR[s]30..0 || 0) − AR[t] |
| SUBX4 | Subtract register from register shifted by 2 | AR[r] ← (AR[s]29..0 || 02) − AR[t] |
| SUBX8 | Subtract register from register shifted by 3 | AR[r] ← (AR[s]28..0 || 03) − AR[t] |
| NEG | Negate | AR[r] ← 0 − AR[t] |
| ABS | Absolute value | AR[r] ← if AR[s]31 then 0 − AR[s] else AR[s] |

# Bitwise logical

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| AND | Bitwise logical AND | AR[r] ← AR[s] and AR[t] |
| OR | Bitwise logical OR | AR[r] ← AR[s] or AR[t] |
| XOR | Bitwise logical exclusive OR | AR[r] ← AR[s] xor AR[t] |

# Shift

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| EXTUI | Extract unsigned field immediate Shifts right by 0..31 and ANDs with a mask of 1..16 ones The operation of this instruction when the number of mask bits exceeds the number of significant bits remaining after the shift is undefined and reserved for future use. |  |
| SRLI | Shift right logical immediate by 0..15 bit positions There is no SRLI for shifts ≥ 16; use EXTUI instead |  |
| SRAI | Shift right arithmetic immediate by 0..31 bit positions |  |
| SLLI | Shift left logical immediate by 1..31 bit positions (see page 525 for encoding of the immediate value). |  |
| SRC | Shift right combined (a funnel shift with shift amount from SAR) The two source registers are catenated, shifted, and the least significant 32 bits returned. |  |
| SLL | Shift left logical (Funnel shift AR[s] and 0 by shift amount from SAR) |  |
| SRL | Shift right logical (Funnel shift 0 and AR[s] by shift amount from SAR) |  |
| SRA | Shift right arithmetic (shift amount from SAR) |  |
| SSL | Set shift amount register (SAR) for shift left logical |  |
| SSR | Set shift amount register (SAR) for shift right logical This instruction differs from WSR to SAR in that only the five least significant bits of the register are used. |  |
| SSAI | Set shift amount register (SAR) immediate |  |
| SSA8B | Set shift amount register (SAR) for big-endian byte align The t field must be zero. |  |
| SSA8L | Set shift amount register (SAR) for little-endian byte align |  |

# Processor control

|  |  |  |
| --- | --- | --- |
| **Instruction** | **Description** | **Example** |
| RSR | Read Special Register |  |
| WSR | Write Special Register |  |
| XSR | Exchange Special Register (combined RSR and WSR) Not present in T1030 and earlier processors |  |
| RUR | RUR reads 32 bits of TIE state into an address register. |  |
| WUR | WUR writes 32 bits to a TIE state register from an address register. |  |
| ISYNC | Instruction fetch synchronize: Waits for all previously fetched load, store, cache, and special register write instructions that affect instruction fetch to be performed before fetching the next instruction. |  |
| RSYNC | Instruction register synchronize: Waits for all previously fetched WSR and XSR instructions to be performed before interpreting the register fields of the next instruction. This operation is also performed as part of ISYNC |  |
| ESYNC | Register value synchronize: Waits for all previously fetched WSR and XSR instructions to be performed before the next instruction uses any register values. This operation is also performed as part of ISYNC and RSYNC. |  |
| DSYNC | Load/store synchronize: Waits for all previously fetched WSR and XSR instructions to be performed before interpreting the virtual address of the next load or store instruction. This operation is also performed as part of ISYNC, RSYNC, and ESYNC. |  |
| NOP | No operation |  |