

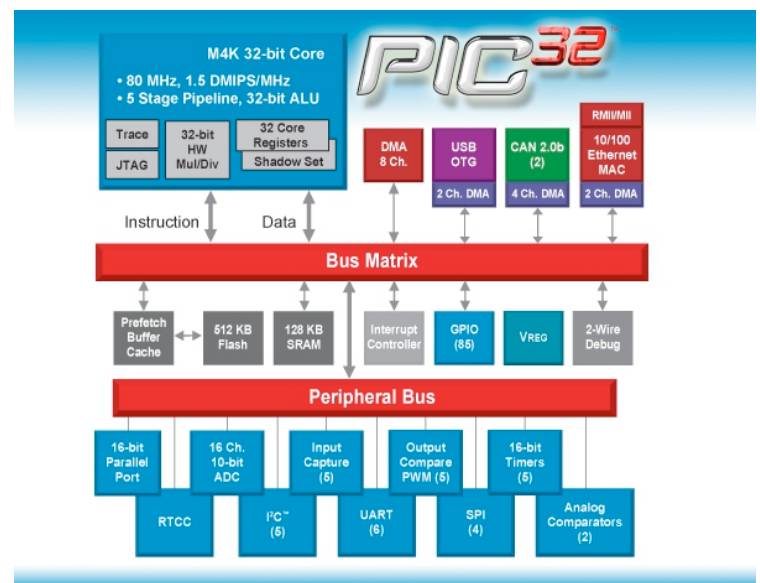


Microva Corporation

MIPS32

Assembler Manual 2.6

(For Microchip PIC32MX/MZ Microcontrollers)



Copyright (c) 2016-2025 Microva Corporation, Sellersville, PA.

The Microva BASIC kernel firmware ("the firmware") and Microva MIPS32 Assembler Software ("the software") are owned and copyrighted by Microva Corporation of Sellersville, PA, USA.

By using the firmware and/or software, you are agreeing to be bound to the terms of this license agreement. The terms are:

1. The firmware is only provided in "machine readable" form within the hardware device and may not be copied without written permission from Microva Corporation.
2. The software is licensed, not sold, to you on a non-exclusive basis.
3. THE FIRMWARE AND SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND EXPRESSED OR IMPLIED INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE FIRMWARE OR SOFTWARE OR THE USE OR OTHER DEALINGS IN THE FIRMWARE OR SOFTWARE.

Microva and Microva BASIC are trademarks of Microva Corporation.

Microsoft and Windows are trademarks of Microsoft Corporation.

Pic32, Microchip and the Microchip logo are trademarks of Microchip Technology Incorporated.

MIPS and the MIPS logo are trademarks of Mips Tech LLC.

Any other trademarks referenced are the properties of their owners.

1.0 Introduction

This is the programmer's reference manual for the Microva assembly software which creates standard 32 bit op codes for MIPS32 type CPUs using simple "pseudo" op code mnemonics which are very similar to the original Motorola 68000 op codes. The Microva BASIC real time kernel firmware was written using this assembly software.

2.0 Memory Map and Operating Mode

MIPS CPUs include a Memory Management Unit (MMU) which creates virtual memory addressing in order to distinguish between "Supervisor" mode addressing (also called Kernel mode) and "User" mode addressing. Most hardware resources are not accessible while in User mode in order to prevent User programs from illegal activity (whether intentional or not).

MIPS CPUs use one or more "coprocessors" to handle hardware configuration, floating point math, interrupts and exceptions (such as RESET). The CPU resets in Kernel mode. The base operating mode can be changed from Kernel to User mode by writing a bit in CP0 (coprocessor 0). All exceptions and interrupts force the CPU into Kernel mode and this is the only method to change from User to Kernel mode since User mode software cannot access CP0.

User mode program addresses correspond to the lower half of the 4GB memory space (i.e. when addr b31 = 0) and Kernel mode program addresses use the upper 2GB of the 4GB memory address range. The User mode address space is also the same as the physical memory address space (except of course User mode programs cannot read/write to the hardware I/O memory locations). The virtual and physical memory addresses are shown in the graphic for the PIC32MX370F512 CPU. The Microva BASIC kernel firmware always runs in Kernel mode since interrupt software forces the CPU to run in Kernel mode and since User mode software cannot access the hardware I/O. This means that the assembler source program must use the virtual memory address in the "org" statement so that the

assembler can calculate the correct address to read, for example, data from a table in ROM. However, the Intel HEX output file must use the physical memory addresses so that the assembled op codes can be properly loaded into the chip programmer. Therefore, the assembler software always clears b31 in the "org" statement before writing the HEX output file. Also, any ROM files (such as Fonts or bitmaps) must be loaded into memory at the physical memory address not the virtual address (but are then referenced in the source program at the virtual address).

VIRTUAL AND PHYSICAL MEMORY MAP FOR PIC32MX370F512		
VIRTUAL		PHYSICAL
0xBF90 0000	HARDWARE I/O (SFRs)	0x1F90 0000
0xBF80 0000		0x1F80 0000
0x9FC0 3000	BOOT FLASH ROM (12K)	0x1FC0 3000
0x9FC0 0000		0x1FC0 0000
0x9D08 0000	MAIN FLASH ROM (512K)	0x1D08 0000
0x9D00 0000		0x1D00 0000
0x8002 0000	SRAM (128K)	0x0002 0000
0x8000 0000		0x0000 0000
0x0000 0000		

3.0 Registers, The Stack and Subroutines

The stack is "full descending" which means new data pushed onto the stack resides at progressively lower addresses. There are (32) registers available in the MIPS32 CPU. All registers are (32) bit and all registers may be used with all op codes. The Program Counter (PC) is not one of these registers. The following registers are reserved for special functions:

r0 = always 0 (by hardware)
r29 = used by the assembler to hold intermediate results
r30 = stack pointer
r31 = link register

The link register (r31) is used to hold the return address during a subroutine call. There are two types of subroutines- those that call other subroutines (i.e. "nested" subroutines) and "stand alone" subroutines. Each nested subroutine must "push r31" (and optionally any other registers) as the first instruction in the subroutine. The return instruction from a nested subroutine ("retn") will pop r31 from the stack and copy it to the PC. Stand alone subroutines do not need to "push r31" and the return instruction for a stand alone subroutine should be "goto r31" (stand alone subroutines can still use the "push r31/retn" statements but it takes less CPU clock cycles to use the simpler "goto r31"). Subroutines can be nested as often as needed (until the stack overflows). See the examples below.

EXAMPLE STAND ALONE SUBROUTINE

```
SetBit3:
    bset 3, r5
    goto r31
```

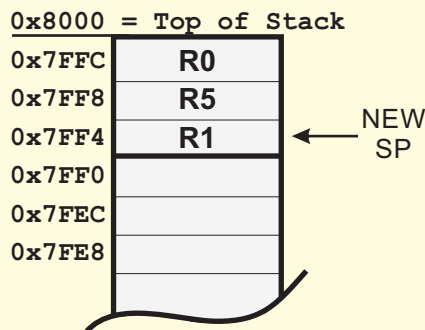
EXAMPLE NESTED SUBROUTINE

```
SetBit3_5:
    push r31
    call SetBit3
    bset 5, r5
    retn
```

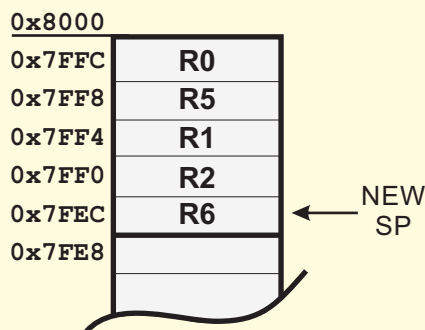
STACK OPERATION

Assumes Top of RAM = 0x7FFF

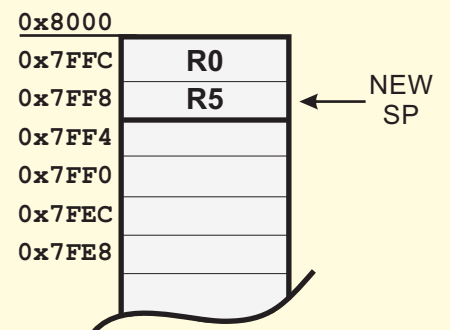
PUSH R0, R5, R1



PUSH R2, R6



POP R6, R2, R1



4.0 Op Code and Data Format

The Microchip PIC32MX MCUs use the "little endian" memory format. This means that data is stored with the LSB at the lowest address. The example below illustrates the little endian data format when storing two data words. The op codes shown on the following pages are listed alphabetically by the pseudo op code. The actual MIPS32 op code is described and shown below the pseudo op code. The "number of clocks" shown for each pseudo op code is the number of CPU clock cycles needed to execute the op code when running out of zero wait state program memory. Just as in the Motorola 68000, the primary op code to move data around is the **MOV** command. Data can be moved a byte at a time using **MOVB** or a half word at a time (16 bits) using **MOVH** or a full word at a time (32 bits) using **MOVL**. The move address must be data aligned, i.e. even for **MOVH** and modulo 4 for **MOVL**.

LITTLE ENDIAN DATA FORMAT

	ADDR	MEMORY
org 0x7C00	0x7C07	0x27
dl 0x826B0A51	0x7C06	0x05
dl 0x27054A3C	0x7C05	0x4A
	0x7C04	0x3C
	0x7C03	0x82
	0x7C02	0x6B
	0x7C01	0x0A
	0x7C00	0x51

add data, rd

rd = rd + data (data must range from -32,768 to +32,767)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
addiu rd,rd,data	0	0	1	0	0	1	rd					rd					immediate															

add rs, rd

$$rd = rd + rs$$

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
addu rd,rs,rd	0	0	0	0	0	0	rs					rd					rd					0	0	0	0	0	1	0	0	0	0	1

align

assembler directive which moves the instruction address pointer to the next "modulo 4" address by inserting 2, 3, or 4 data bytes (all = 0x00) or by doing nothing if the address is already "modulo 4"

and data, rd

rd = rd and data (data must range from 0 to 0xFFFF)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
andi rd,rd,data	0	0	1	1	0	0	rd					rd					immediate															

and rs, rd

rd = rd and rs

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
and rd,rs,rd	0	0	0	0	0	0	rs					rd					rd					0	0	0	0	0	1	0	0	1	0	0

asr n, rd

rd = rd asr n times (n must range from 0 to 31)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
sra rd,rd,data	0	0	0	0	0	0	0	0	0	0	0	rd				rd				data				0				0	0	0	1	1

asr rs, rd

rd = rd asr "rs" times (lower 5 bits in "rs" are used for the shift data)

number clocks: 1

[illegible]

beq rs, label

branch if rs equals zero, offset = (label addr - instruction addr - 4) / 4
number clocks: 2

[illegible]**bne rs, label**

branch if rs not equal zero, offset = (label addr - instruction addr - 4) / 4
number clocks: 2

[illegible]**bge rs, label**

branch if rs is greater than or equal zero, offset = (label addr - instruction addr - 4) / 4
number clocks: 2

[illegible]**bgt rs, label**

branch if rs is greater than zero, offset = (label addr - instruction addr - 4) / 4
number clocks: 2

[illegible]

ble rs, label

branch if rs is less than or equal zero, offset = (label addr - instruction addr - 4) / 4
number clocks: 2

[illegible]

blt rs, label

branch if rs is less than zero, offset = (label addr - instruction addr - 4) / 4

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
bltz rs,offset	0	0	0	0	0	1	rs					0	0	0	0	0	offset															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
nop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

bra label

unconditional branch, offset = (label addr - instruction addr - 4) / 4

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
beq r0,r0,offset	0	0	0	1	0	0	r0					r0					offset															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
nop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

bclr n, rd

clear bit n of rd (n must range from 0 to 31)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ins rd,r0,n,1	0	1	1	1	1	1	r0					rd					n					n					0	0	0	1	0	0

bset n, rd

set bit n of rd (n must range from 0 to 31)

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori r29,r0,0x01	0	0	1	1	0	1	r0					r29					immediate															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ins rd,r29,n,1	0	1	1	1	1	1	r29					rd					n					n					0	0	0	1	0	0

btst n, rd

load bit n of rd to r29 (n must range from 0 to 31)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ext r29,rd,n,1	0	1	1	1	1	1	rd					r29					0	0	0	0	0	n					0	0	0	0	0	0

cmp rs, rd

r29 = rd - rs
number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
subu r29,rs,rd	0	0	0	0	0	0	rd				rs				r29				0	0	0	0	0	1	0	0	0	1	1			

db <data>

assembler directive which inserts byte data in the program

```
db 0xFF, 5, "Hello", 23      ;ASCII strings are allowed
db 5, MyLabel                 ;Error: labels not permitted
db 5, MyData                  ;equates are allowed
db 5, -23                     ;Error: byte range = 0 to 255
```

dh <data>

assembler directive which inserts "half integer" data in the program (data is stored in "little endian" format)

```
dh 5, MyLabel                 ;Error: labels not permitted
dh 5, MyData                  ;equates are allowed
dh 5, -23                     ;Error: range = 0 to 0xFFFF
```

divs rs, rd

rd = rd / rs (both dividend rd and divisor rs are 32 bit signed integers, result rd is not rounded)
r29 = remainder
number clocks: 4-36

div rd,rs

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0 0 0 0 0 0

rd

rs

0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0

mflo rd

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

rd

0 0 0 0 0 0 1 0 0 1 0

mfhi r29

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

r29

0 0 0 0 0 0 1 0 0 0 0

divu rs, rd

rd = rd / rs (both dividend rd and divisor rs are 32 bit unsigned integers, result rd is not rounded)
r29 = remainder
number clocks: 4-36

divu rd,rs

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0 0 0 0 0 0

rd

rs

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1

mflo rd

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

rd

0 0 0 0 0 0 1 0 0 1 0

mfhi r29

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

r29

0 0 0 0 0 0 1 0 0 0 0

di

disable interrupts
number clocks: n

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
di	0	1	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

dl <data>

assembler directive which inserts long data in the program (data will be stored in "little endian" format)

```
dl  0xFF, "Hello"           ;Error: ASCII strings not allowed
dl  0xFF80FA55, 0xFFE0, MyLabel ;labels and equates are allowed
```

end

assembler directive which ends the program

eor data, rd

rd = rd eor data (data must range from 0 to 0xFFFF)
number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
xori rd,rd,data	0	0	1	1	1	0	rd					rd					immediate															

eor rs, rd

rd = rd eor rs
number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
xor rd,rs,rd	0	0	0	0	0	0	rs					rd					rd					0	0	0	0	0	1	0	0	1	1	0

equ <name>, <data>

assembler directive used to assign equates

```
equ  MyData1, 0xFF           ;MyData1 = 255
equ  MyData2, -15            ;MyData2 = -15
equ  MyData3, MyAddr         ;Error: labels not permitted
```

glz rs, rd

rd = number of “leading zeros” in rs (rd will range from 0 to 32)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
clz rd,rs	0	1	1	1	0	0	rs					rd					rd					0	0	0	0	0	1	0	0	0	0	0

goto label

goto label (addr = (label addr & 0x0FFF FFFF) / 4)

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
j addr	0	0	0	0	1	0	addr																									
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
nop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

goto rs

goto addr in rs (addr must be modulo 4)

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
j r rs	0	0	0	0	0	0	rs					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
nop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ior data, rd

rd = rd or data (data must range from 0 to 0xFFFF)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori rd,rd,data	0	0	1	1	0	1	rd					rd					immediate															

ior rs, rd

rd = rd or rs

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
or rd,rs,rd	0	0	0	0	0	0	rs					rd					rd					0	0	0	0	0	1	0	0	1	0	1

lsl n, rd

rd = rd lsl n times (n must range from 0 to 31)
number clocks: 1

[illegible]**lsl** **rs, rd**

rd = rd lsl "rs" times (lower 5 bits in "rs" are used for the shift data)
number clocks: 1

[illegible]

lsr n, rd

rd = rd lsr n times (n must range from 0 to 31)
number clocks: 1

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
srl	rd,rd,data	0	0	0	0	0	0	0	0	0	0	0	rd				rd				data				0				0	0	0	1	0

lsrc rs, rd

```
rd = rd lsr "rs" times (lower 5 bits in "rs" are used for the shift data)
number clocks: 1
```

[illegible]

mfc0 <cp0 reg>, rd

rd = CP0 register (reg name must be: Status, Count, Compare, IntCtl, SrsCtl, Cause, EPC, or EBase)
number clocks: n

[illegible]

mfpr rd

rd = general register from previous shadow register set
number clocks: n

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00			
rdpgpr	rd,rd	0	1	0	0	0	0	0	1	0	1	0	rd				rd				0								0							

movb rs, (rd)

number clocks: 1

[illegible]**movb rs, (rd)+**

number clocks: 2

sb rs,0(rd)	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00	1 0 1 0 0 0	rd	rs	offset
addiu rd,rd,1	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00	0 0 1 0 0 1	rd	rd	immediate

movb rs, n(rd)

n is offset in bytes (offset must range from -32,768 to +32,767)

number clocks: 1

[illegible]**movb** (rs), rd

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lbu rd,0(rs)	1	0	0	1	0	0	rs			rd			offset																			

movb (rs)+, rd

number clocks: 2

lbu rd,0(rs)	31 30 29 28 27 26 1 0 0 1 0 0	25 24 23 22 21 rs	20 19 18 17 16 rd	15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 offset
addiu rs,rs,1	31 30 29 28 27 26 0 0 1 0 0 1	25 24 23 22 21 rs	20 19 18 17 16 rs	15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 immediate

movb n(rs), rd

n is offset in bytes (offset must range from -32,768 to +32,767)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lbu rd,n(rs)	1	0	0	1	0	0	rs					rd					offset															

movb (addr), rd

"addr" must be a constant
number clocks: 3

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lui r29,uaddr	0	0	1	1	1	1	0	0	0	0	0	r29				immediate																
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori r29,r29,laddr	0	0	1	1	0	1	r29				r29				immediate																	
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lbu rd,0(r29)	1	0	0	1	0	0	r29				rd				offset																	

movb rs, (addr)

"addr" must be a constant
number clocks: 3

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lui r29,uaddr	0	0	1	1	1	1	0	0	0	0	0	r29				immediate																
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori r29,r29,laddr	0	0	1	1	0	1	r29				r29				immediate																	
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
sb rs,0(r29)	1	0	1	0	0	0	r29				rs				offset																	

movh data, rd

data must range from 0 to 0xFFFF

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori rd,r0,data	0	0	1	1	0	1	r0					rd					immediate															

movh rs, (rd)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
sh rs,0(rd)	1	0	1	0	0	1	rd			rs			offset																			

movh rs, (rd)+

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
sh rs,0(rd)	1	0	1	0	0	1	rd					rs					offset															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
addiu rd,rd,2	0	0	1	0	0	1	rd					rd					immediate															

movh rs, n(rd)

n is offset in half words (offset = $n * 2$ and must range from -32,768 to +32,767)

number clocks: 1

[illegible]

movh (rs), rd

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1hu rd,0(rs)	1	0	0	1	0	1	rs					rd					offset															

movh (rs)+, rd

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1hu rd,0(rs)	1	0	0	1	0	1	rs					rd					offset															

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
addiu rs,rs,2	0	0	1	0	0	1	rs					rs					immediate															

movh n(rs), rd

n is offset in half words (offset = $n * 2$ and must range from -32,768 to +32,767)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1hu rd,n(rs)	1	0	0	1	0	1	rs					rd					offset															

movh (addr), rd

"addr" must be a constant

number clocks: 3

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lui r29,uaddr	0	0	1	1	1	1	0	0	0	0	0	r29					immediate															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori r29,r29,laddr	0	0	1	1	0	1	r29					r29					immediate															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lhu rd,0(r29)	1	0	0	1	0	1	r29					rd					offset															

movh rs, (addr)

"addr" must be a constant

number clocks: 3

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lui r29,uaddr	0	0	1	1	1	1	0	0	0	0	0	r29				immediate																
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori r29,r29,laddr	0	0	1	1	0	1	r29				r29				immediate																	
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
sh rs,0(r29)	1	0	1	0	0	1	r29				rs				offset																	

movl data, rd

data can be a constant or an address label (range is -2,147,483,648 to +2,147,483,647)

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lui rd,udata	0	0	1	1	1	1	0	0	0	0	0	rd				immediate																

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori rd,rd,ldata	0	0	1	1	0	1	rd				rd				immediate																	

movl rs, rd

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
addiu rd,rs,0	0	0	1	0	0	1	rs					rd					immediate															

movl rs, (rd)

number clocks: 1

[illegible]**movl** rs, (rd)+

number clocks: 2

sw rs,0(rd)	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
	1 0 1 0 1 1 rd rs offset

addiu rd,rd,4	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
	0 0 1 0 0 1 rd rd immediate

movl rs, n(rd)

n is offset in words (offset = $n * 4$ and must range from -32,768 to +32,767)

number clocks: 1

[illegible]**movl (rs), rd**

number clocks: 1

[illegible]

movl (rs)+, rd

number clocks: 2

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lw rd,0(rs)	1	0	0	0	1	1	rs					rd					offset															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
addiu rs,rs,4	0	0	1	0	0	1	rs					rs					immediate															

movl n(rs), rd

n is offset in words (offset = n * 4 and must range from -32,768 to +32,767)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lw rd,n(rs)	1	0	0	0	1	1	rs					rd					offset															

movl (addr), rd

"addr" must be a constant

number clocks: 3

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lui r29,uaddr	0	0	1	1	1	1	0	0	0	0	0	r29					immediate															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori r29,r29,laddr	0	0	1	1	0	1	r29					r29					immediate															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lw rd,0(r29)	1	0	0	0	1	1	r29					rd					offset															

movl rs, (addr)

"addr" must be a constant

number clocks: 3

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
lui r29,uaddr	0	0	1	1	1	1	0	0	0	0	0	r29					immediate															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
ori r29,r29,laddr	0	0	1	1	0	1	r29					r29					immediate															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
sw rs,0(r29)	1	0	1	0	1	1	r29					rs					offset															

mtc0 rs, <cp0 reg>

CP0 register = rs (CP0 reg names: Status, Config, Count, Compare, IntCtl, SrsCtl, Cause, EPC, or EBase)

number clocks: 1

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
mtc0	rs,rd,sel	0	1	0	0	0	0	0	0	1	0	0	rs					rd					0 0 0 0 0 0 0 0					sel					

move general register rs to previous shadow register set
number clocks: n

[illegible]

mul data, rd

rd = data * rd (both data and rd can be positive or negative, data must range from -32,768 to +32,767)
number clocks: 3

addiu r29,r0,data	31 30 29 28 27 26 25 24 23 22 21	20 19 18 17 16 15 14 13 12 11	10 09 08 07 06 05 04 03 02 01 00
	0 0 1 0 0 1	r0	r29
	immediate		

mul rd,rd,r29	31 30 29 28 27 26 25 24 23 22 21	20 19 18 17 16 15 14 13 12 11	10 09 08 07 06 05 04 03 02 01 00
	0 1 1 1 0 0	rd	r29
		rd	0 0 0 0 0 0 0 0 0 0 1 0

mul rs, rd

rd = rd * rs (both rd and rs can be positive or negative)
number clocks: 2

[illegible]

mulx rs, rd

$r1/r2 = rd * rs$ (both rd and rs are 32 bit unsigned integers, result = 64 bits, $r1 = b63$ to $b32$, $r2 = b31$ to $b0$)
number clocks: 4

[illegible]

neg rs, rd

```
rd = 0 - rs
number clocks: 1
```

[illegible]

nop

no operation
number clocks: 1

[illegible]

lw r2,0(r30)	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
	1 0 0 0 1 1 r30 r2 offset
lw r9,4(r30)	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
	1 0 0 0 1 1 r30 r9 offset
lw r5,8(r30)	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
	1 0 0 0 1 1 r30 r5 offset
addiu r30,r30,12	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
	0 0 1 0 0 1 r30 r30 immediate

retn

return from subroutine

number clocks: 5

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
lw r31,0(r30)	1	0	0	0	1	1	r30					r31					offset																
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
addiu r30,r30,4	0	0	1	0	0	1	r30					r30					immediate																
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
jrr r31	0	0	0	0	0	0	r31					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	
nop	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

reti

return from interrupt

number clocks: n

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
eret	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0

sub data, rd

rd = rd - data (data must range from +32,768 to -32,767)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
addiu rd,rd,-data	0	0	1	0	0	1	rd					rd					immediate															

sub rs, rd

rd = rd - rs

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
subu rd,rd,rs	0	0	0	0	0	0	rd					rs					rd					0	0	0	0	0	1	0	0	0	1	1

wait

start low power mode (sleep or idle)

number clocks: 1

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
wait	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0