

Schwap CPU Design Documentation

Charlie Fenoglio, Alexander Hirschfeld, mckeeaj, and Wesley Van Pelt

Winter 2015/2016

1 Registers

There are a total of 76 16-bit registers; 12 are fixed and 64 (spilt into 16 groups of 4) "schwapable" registers.

1.1 Register Names and Descriptions

Name	Number	Description	Saved Across Call?
\$0	0	The Value 0	-
\$pc	1	Program Counter	Yes
\$sp	2	Stack Pointer	Yes
\$ra	3	Return Address	Yes
\$s0 - \$s3	4 - 7	User Temporary Saved	Yes
\$t0 - \$t3	8 - 11	User Temporaries	No
\$h0 - \$h3	12 - 15	Schwap	-

1.2 Schwap Registers

The "schwap" registers are registers that appear to be swapped using a command. There is no data movement when schwapping, it only changes which registers the \$h0 - \$h3 refer to. There are 8 groups the user can switch between and 8 reserved groups.

1.2.1 Schwap Group Numbers, Descriptions, and Uses

Group Number	ID	Uses	Saved Across Call?
0 - 7	0 - 3	User Temporaries	No
8	0 - 3	I/O for devices 0 - 3	-
9	0 - 3	Arguments 0 - 3	No
10	0 - 3	Return Values 0 - 3	No
11	0 - 3	System Call Values 0 - 3	No
12	0 - 3	Kernel Reserved	No
13	0 - 3	Temporary Restore	No
14	0	Exception Cause	No
	1	Exception Status	No
	2	EPC	No
	3	Exception Temporary	No
15	0 - 3	Assembler Temporaries	No

2 Instructions

All instructions are 16-bits. The destination register is also used as a source unless otherwise noted. All offsets are bit shifted left by 1 since all instructions are 2 bytes long.

2.1 Instruction Types and Bit Layouts

Instructions can be manually translated by putting the bits for each of the components of the instructions in the places listed by the diagrams for each type. The OP codes, function codes, and types can be found on the "Core Instructions" (2.2) table. The destination and source are register numbers, which can be found under the "Register Names and Descriptions" (1.1) table. Schwap group numbers can be found under the "Schwap Group Numbers, Descriptions, and Uses" (1.2.1) table.

2.1.1 A-Type

Used for all ALU operations. It consists of a 4-bit OP code, 4-bit destination, 4-bit source, and a 4-bit function code.

OP Code		Destination		Source		Func. Code	
15	12	11	8	7	4	3	0

2.1.2 B-Type

Used for branching. It consists of a 4-bit OP code, 4-bit 1st source, 4-bit 2nd source, and a 4-bit offset.

OP Code		Source 0		Source 1		Offset	
15	12	11	8	7	4	3	0

2.1.3 H-Type

Used for schwapping. It consists of a 4-bit OP code, 8 unused bits, and a 4-bit schwap group number.

OP Code						Group	
15	12					3	0

2.1.4 J-Type

Used for jumping. It consists of a 4-bit OP code, 4-bit source, and an 8-bit offset.

OP Code		Source		Offset			
15	12	11	8	7			0

2.1.5 R-Type

Used for reading and writing memory. It consists of a 4-bit OP code, 4-bit destination (not used as a source), 4-bit source, and a 4-bit offset.

OP Code		Destination		Source		Offset	
15	12	11	8	7	4	3	0

2.1.6 S-Type

Used for sudo. Only has a 4-bit OP code.

OP Code							
15	12						

2.2 Core Instructions

OP Code	Function Code	Name	Type	Syntax	Meaning	Description
0x0 and 0x1 [†]	0x0	add	A	add [dest], [src] add [dest], [immediate]	dest += src dest += immediate	Adds 2 Integers
	0x1	adu	A	adu [dest], [src] adu [dest], [immediate]	dest += src dest += immediate	Adds 2 Unsigned Integers
	0x2	sub	A	sub [dest], [src] sub [dest], [immediate]	dest -= src dest -= immediate	Subtracts 2 Integers
	0x3	sbu	A	sbu [dest], [src] sbu [dest], [immediate]	dest -= src dest -= immediate	Subtracts 2 Unsigned Integers
	0x4	sll	A	sll [dest], [immediate]	dest <<= immediate	Left Logical Bit shift
	0x5	srl	A	srl [dest], [immediate]	dest >>= immediate	Right Logical Bit shift
	0x6	sra	A	sra [dest], [immediate]	dest >>>= immediate	Right Arithmetic Bit shift
	0x7	and	A	and [dest], [src] and [dest], [immediate]	dest &= src dest &= immediate	Ands 2 Values
	0x8	orr	A	orr [dest], [src] orr [dest], [immediate]	dest = src dest = immediate	Ors 2 Values
	0x9	xor	A	xor [dest], [src] xor [dest], [immediate]	dest ^ = src dest ^ = immediate	Xors 2 Values
	0xA	not	A	not [dest], [src] not [dest], [immediate]	dest = ~src dest = ~immediate	Bitwise Nots 2 Values
	0xB	tsc	A	tsc [dest], [src] tsc [dest], [immediate]	dest = ~src + 1 dest = ~immediate + 1	Converts a number to 2's compliment
	0xF	cpy	A	cpy [dest], [src] cpy [dest], [immediate]	dest = src dest = immediate	Copies a value
0x2	-	jr	J	jr [regNumber]	PC = regNumber	Sets the PC to the value in the register
0x3	-	rsh	H	rsh [groupNumber]	-	Sets the schwappable registers to use
0x4	-	beq	B	beq [src0], [src1], label	-	Branches to the label if src0 == src1
0x5	-	bnq	B	bnq [src0], [src1], label	-	Branches to the label if src0 != src1
0x6	-	bgt	B	bgt [src0], [src1], label	-	Branches to the label if src0 > src1
0x7	-	r	R	r [dest], [src]	-	Moves the value in memory at the address in src into dest
0x8	-	w	R	w [dest], [src]	-	Moves the value in dest to the address in src in memory
0xF	-	sudo	S	sudo	-	Same as "syscall" in MIPS

[†]0x0 is used for instructions which do not use an immediate, 0x1 is used if the instruction does use an immediate

2.3 Psuedo Instructions

Name	Syntax	Actual Code	Description
li	li [dest], [immediate]		
j	j label		
jal	jal label		

3 Examples

3.1 Basic Use Examples

3.1.1 Loading an address into a register

```
code
```

3.1.2 Iteration

```
code
```

3.1.3 Conditional Statements

```
code
```

3.1.4 I/O

```
code
```

3.2 relPrime and gcd Implementation

3.2.1 relPrime

```
RelPrime:
    rsh    9                #set schwap
    cpy    $s2, $ra         #save $ra
    cpy    $s0, $h0         #copy n out of schwap
    li     $s1, 0x2         #load 2 to m
    rsh    9                #set schwap to args
While:
    cpy    $h0, $s0         #set a0 to n
    cpy    $h1, $s1         #set a1 to m
    jl     GCD              #call GCD
    rsh    10               #set schwap
    li     $t0, 0x1         #load immediate 0x1 to t0
    bne    $h0, $t0, Done   #branch to done if r0 != 1
    add    $s1, 0x1         #add 1 to m
    j      While            #jump to the start of the loop
Done:
    rsh    10               #load return registers
    cpy    $h0, $s1         #set r0 to m
    jr     $s2, 0           #return to the previous function
```

3.2.2 gcd

```
GCD:
    rsh    9                #schwap to argument register
Base:
    bne    $h0, $z0, GMain  #a!=0 go to GMain
```

	cpy	\$t0, \$h1	<i>#copy h1 to t0 for RSH</i>
	rsh	10	<i>#schwap to return registers</i>
	cpy	\$h0, \$t0	<i>#load t0 to r1</i>
	jr	\$ra, 0	<i>#return</i>
GMain:	beq	\$h1, \$z0, Exit	<i>#jump to exit if b is zero</i>
	bgt	\$h0, \$h1, If	<i>#jump to If if a>b</i>
Else:	sub	\$h1, \$h0	<i>#else: b=b-a</i>
	j	GMain	<i>#loop</i>
If:	sub	\$h0, \$h1	<i>#if: a=a-b</i>
	j	GMain	<i>#loop</i>
Exit:	cpy	\$t0, \$h0	<i>#copy h0 to t0 for rsh schwap</i>
	rsh	10	<i>#make sure we're in the right spot</i>
	cpy	\$h0, \$t0	<i>#copy t0 to h0</i>
	jr	\$ra	<i>#return</i>