TSC Instruction Set Architecture

1 Instruction Format

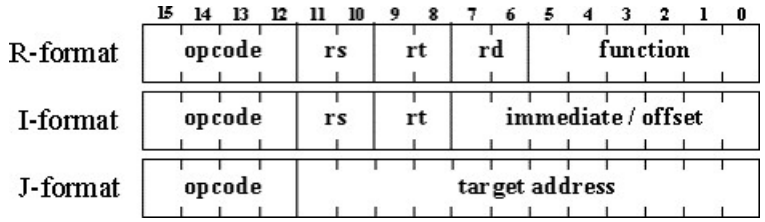

Figure 1: Instruction Format

2 Instruction Opcodes

Instruction	Opcode	Function Code	Format
ADD	15	0	R
SUB	15	1	R
AND	15	2	R
ORR	15	3	R
NOT	15	4	R
TCP	15	5	R
SHL	15	6	R
SHR	15	7	R
ADI	4	-	I
ORI	5	-	I
LHI	6	-	I
WWD	15	28	R
LWD	7	-	I
SWD	8	-	I
BNE	0	-	I
BEQ	1	-	I
BGZ	2	-	I
BLZ	3	-	I
JMP	9	-	J
JAL	10	-	J
JPR	15	25	R
JRL	15	26	R
HLT	15	29	R

3 Instruction Description

3.1 ADD

3.1.1 Assembler Format

ADD $rd, $rs, $rt

3.1.2 Description

The contents of $rs and the contents of $rt are added to form the result. The result is placed into $rd.

3.1.3 Operation

$rd ← $rs + $rt

3.1.4 Examples

```
ADD $1, $2, $0    ; $1 ← $2 + $0
ADD $2, $2, $2    ; $2 ← $2 + $2
```

3.2 SUB

3.2.1 Assembler Format

SUB $rd, $rs, $rt

3.2.2 Description

The contents of $rt are subtracted from the contents of $rs to form the result. The result is placed into $rd.

3.2.3 Operation

$rd ← $rs + !$rt + 1

3.2.4 Examples

```
SUB  $3, $0, $1    ; $3 ← $0 - $1
SUB $0, $0, $0    ; $0 ← $0 - $0 ; This clears $0
```

3.3 AND

3.3.1 Assembler Format

AND $rd, $rs, $rt

3.3.2 Description

The contents of $rs are combined with the contents of $rt in a bit-wise AND operation. The result is placed into $rd.

3.3.3 Operation

$rd ← $rs & $rt

3.3.4 Examples

```
AND  $0, $1, $2    ; $0 ← $1 & $2
AND  $3, $3, $3    ; $3 ← $3 & $3 ; Contents of $3 are unchanged
```

3.4 ORR

3.4.1 Assembler Format

ORR $rd, $rs, $rt

3.4.2 Description

The contents of $rs are combined with the contents of $rt in a bit-wise OR operation. The result is placed into $rd.

3.4.3 Operation

$rd ← $rs | $rt

3.4.4 Examples

```
ORR  $1, $2, $1    ; $1 ← $2 | $1
ORR  $3, $3, $3    ; $3 ← $3 | $3 ; Contents of $3 are unchanged
```

3.5 NOT

3.5.1 Assembler Format

NOT $rd, $rs

3.5.2 Description

The bit-wise inverse of the contents of $rs are placed into $rd.

3.5.3 Operation

$rd ← !$rs

3.5.4 Examples

```
NOT  $0, $1    ; $0 ← !$1
NOT  $3, $3    ; $3 ← !$3
```

3.6 TCP

3.6.1 Assembler Format

TCP $rd, $rs

3.6.2 Description

The two's complement of the contents of $rs is placed into $rd.

3.6.3 Operation

$rd ← !$rs + 1

3.6.4 Examples

```
TCP $0, $2    ; $0 ← !$2 + 1
TCP $1, $1    ; $1 ← !$1 + 1
```

3.7 SHL

3.7.1 Assembler Format

SHL $rd, $rs

3.7.2 Description

The contents of $rs are shifted left one bit, inserting zero into the least significant bit. The result is placed into $rd.

3.7.3 Operation

$rd ← { $rs14..0 ## 0 }

3.7.4 Examples

```
SHL  $0, $1    ; $0 ← ($1 << 1)
```

3.8 SHR

3.8.1 Assembler Format

SHR $rd, $rs

3.8.2 Description

The contents of $rs are shifted right one bit, sign-extending the most significant bit i.e. the value of the most significant bit is preserved. The result is placed into $rd.

3.8.3 Operation

$rd ← { $rs15 ## $rs15..1 }

3.8.4 Examples

```
SHR $2, $1    ; $2 ← ($1 >> 1)
SHR $1, $1    ; $1 ← ($1 >> 1)
```

3.9 ADI

3.9.1 Assembler Format

ADI $rt, $rs, imm

3.9.2 Description

The 8-bit immediate value is sign-extended to 16 bits and added to the contents of $rs to form the result. The result is placed into $rt.

3.9.3 Operation

$rt ← $rs + { (imm7)8 ## imm7..0 }

3.9.4 Examples

```
ADI  $0, $1, -17    ; $0 ← $1 - 17
```

3.10 ORI

3.10.1 Assembler Format

ORI $rt, $rs, imm

3.10.2 Description

The 8-bit immediate value is zero-extended to 16 bits and combined with the contents of $rs in a bit-wise OR operation. The result is placed into $rt.

3.10.3 Operation

$rt ← $rs | { (0)8 ## imm7..0 }

3.10.4 Examples

```
ORI  $0, $1, 0xff     ; $0 ← $1 |0x00ff
ORI $1, $1, 1000      ; Is this valid? Why or why not?
ORI $2, $3, 0         ; $2 ← $3 ; Contents of $3 are copied to $2
```

3.11 LHI

3.11.1 Assembler Format

LHI $rt, imm

3.11.2 Description

The 8-bit immediate value is concatenated to 8 bits of zeros, the immediate value being the most significant halfword. The result is placed into $rt.

3.11.3 Operation

$rt ← { imm7..0 ## (0)8 }

3.11.4 Examples

```
LHI $3, 15    ; $3 ← 15 <<  8
LHI $2, 0     ; $2 ← 0 ; This clears $2
```

3.12 WWD

3.12.1 Assembler Format

WWD $rs

3.12.2 Description

The contents of $rs are written to the output port and the output signal is asserted. Execution stops until an acknowledge is received from the output device.

3.12.3 Operation

output port ← $rs

3.12.4 Examples

```
WWD $0    ; output port ← $0
```

3.13 LWD

3.13.1 Assembler Format

LWD $rt, $rs, offset

3.13.2 Description

The 8-bit address offset is sign-extended and added to the contents of $rs to form a memory address. The word at the specified memory location is loaded into $rt.

3.13.3 Operation

$rt ← Mem[$rs + { (offset7)8 ## offset7..0 }]

3.13.4 Examples

```
LWD $0, $0, 16    ; $0 ← Mem[$0 + 16]
LWD $0, $3, -4    ; $0 ← Mem[$3 - 4]
LWD $0, $3, 0     ; $0 ← Mem[$3]
```

3.14 SWD

3.14.1 Assembler Format

SWD $rt, $rs, offset

3.14.2 Description

The 8-bit address offset is sign-extended and added to the contents of $rs to form a memory address. The contents of $rt are stored at the specified memory location.

3.14.3 Operation

Mem[$rs + { (offset7)8 ## offset7..0 }] ← $rt

3.14.4 Examples

```
SWD $1, $1, 120    ; Mem[$1 + 120] ← $1
SWD $1, $3, -2     ; Mem[$3 - 2] ← $1
SWD $1, $0, 0      ; Mem[$0] ← $1
```

3.15 BNE

3.15.1 Assembler Format

BNE $rs, $rt, offset

3.15.2 Description

A branch target address is computed from the sum of the address of the instruction after the branch instruction and the 8-bit, sign-extended offset. The contents of $rs and $rt are compared. If they are not equal, then the target address is written into the PC and program execution continues with the instruction at the target address. Otherwise, program execution continues with the instruction following the branch.

3.15.3 Operation

If $rs != $rt then $pc ← $inst after the branch inst + { (offset7)8 ## offset7..0 }

3.15.4 Examples

```
BNE $0, $2, 6 ; If $0 != $2 then $pc ← $inst after the branch inst + 6
BNE $1, $2, LOOP1 ; If $1 != $2 then $pc ← LOOP1
```

The last example is the common usage.

Note that if a label is in the offset field in the assembly instruction, the assembler will compute the offset and insert it into the binary instruction. If a number is in the offset field, the number itself will be inserted into the binary instruction.

3.16 BEQ

3.16.1 Assembler Format

BEQ $rs, $rt, offset

3.16.2 Description

A branch target address is computed from the sum of the address of the instruction after the branch instruction and the 8-bit, sign-extended offset. The contents of $rs and $rt are compared. If they are equal, then the target address is written into the PC and program execution continues with the instruction at the target address. Otherwise, program execution continues with the instruction following the branch.

3.16.3 Operation

If $rs == $rt then $pc ← $$inst after the branch inst + { (offset7)8 ## offset7..0 }

3.16.4 Examples

```
BEQ $0, $3, 6        ; If $0 == $3 then $pc ← $inst after the branch inst + 6
BEQ $2, $0, -36  ; If $2 == $0 then $pc ← $inst after the branch inst - 36
BEQ $0, $2, BYE ; If $0 == $2 then $pc ← BYE
```

The last example is the common usage.

Note that if a label is in the offset field in the assembly instruction, the assembler will compute the offset and insert it into the binary instruction. If a number is in the offset field, the number itself will be inserted into the binary instruction.

3.17 BGZ

3.17.1 Assembler Format

BGZ $rs, offset

3.17.2 Description

A branch target address is computed from the sum of the address of the instruction after the branch instruction and the 8-bit, sign-extended offset. The contents of $rs and zero are compared. If the contents of $rs are greater than zero, then the target address is written into the PC and program execution continues with the instruction at the target address. Otherwise, program execution continues with the instruction following the branch.

3.17.3 Operation

If $rs > 0 then $pc ← $inst after the branch inst + { (offset7)8 ## offset7..0 }

3.17.4 Examples

```
BGZ $3, 42 ; If $3 > 0 then $pc ← $inst after the branch inst + 42
BGZ $2, -3 ; If $2 > 0 then $pc ← $inst after the branch inst - 3
BGZ $0, N2 ; If $0 > 0 then $pc ← N2
```

The last example is the common usage.

Note that if a label is in the offset field in the assembly instruction, the assembler will compute the offset and insert it into the binary instruction. If a number is in the offset field, the number itself will be inserted into the binary instruction.

3.18 BLZ

3.18.1 Assembler Format

BLZ $rs, offset

3.18.2 Description

A branch target address is computed from the sum of the address of the instruction after the branch instruction and the 8-bit, sign-extended offset. The contents of $rs and zero are compared. If the contents of $rs are less than zero, then the target address is written into the PC and program execution continues with the instruction at the target address. Otherwise, program execution continues with the instruction following the branch.

3.18.3 Operation

If $rs < 0 then $pc ← $inst after the branch inst+ { (offset7)8 ## offset7..0 }

3.18.4 Examples

```
BLZ $3, 200     ; If $3 < 0 then $pc ← $inst after the branch inst+ 200
BLZ $2, -17     ; If $2 < 0 then $pc ← $inst after the branch inst - 17
BLZ $0, JEFF    ; If $0 < 0 then $pc ← JEFF
```

The last example is the common usage.

Note that if a label is in the offset field in the assembly instruction, the assembler will compute the offset and insert it into the binary instruction. If a number is in the offset field, the number itself will be inserted into the binary instruction.

3.19 JMP

3.19.1 Assembler Format

JMP target

3.19.2 Description

A target address is computed by concatenating the four high-order bits of the instruction's address with the 12-bit, unsigned target offset. The target address is written into the PC and program execution continues with the instruction at the target address.

3.19.3 Operation

$pc ← $pc15..12 ## target11..0

3.19.4 Examples

```
JMP LOOP   ;  $pc ← LOOP    If four high-order bits of $pc are 0
JMP 0x0100  ; $pc ← 0xc100  If $pc previously contained 0xc68f
JMP 0xeeff ; Is this valid? No, it's not. Why not?
```

3.20 JAL

3.20.1 Assembler Format

JAL target

3.20.2 Description

The address of the next instruction is placed in $2. A target address is computed by concatenating the four high-order bits of the instruction's address with the 12-bit, unsigned target offset. The target address is written into the PC and program execution continues with the instruction at the target address.

3.20.3 Operation

$2 ← $next inst
$pc ← $pc15..12 ## target11..0

3.20.4 Examples

JAL LOOP ; $2 ← $next inst ; $pc ← LOOP
JAL 0x05b0 ; $2 ← $next inst ; $pc ← 0x05b0 ; If four high-order bits of $pc are 0
JAL 0xff34 ; Is this valid? No, why not?

3.21 JPR

3.21.1 Assembler Format

JPR $rs

3.21.2 Description

The contents of $rs are written to the PC. Program execution continues with the instruction at that address.

3.21.3 Operation

$pc ← $rs

3.21.4 Examples

JPR $2 ; $pc ← $2

JPR is commonly used in conjunction with JAL to return from procedure calls. JPR and JRL are the only control instructions that allow branching to any address in physical memory.

3.22 JRL

3.22.1 Assembler Format

JRL $rs

3.22.2 Description

The address of the next instruction is placed in $2. The contents of $rs are written to the PC. Program execution continues with the instruction at that address.

3.22.3 Operation

$2 ← $next inst
$pc ← $rs

3.22.4 Examples

JRL $3 ; $2 ← $next inst ; $pc ← $3
JRL $2 ; What happens when this is executed?

JRL is commonly used to jump to procedures that are out of range for a JAL instruction. JPR and JRL are the only control instructions that allow branching to any address in physical memory.

3.23 HLT

3.23.1 Assembler Format

HLT

3.23.2 Description

Indicates the end of a program. When executed, the machine does not fetch.

3.23.3 Operation

is_halted $\leftarrow$ 1
The machine halts.

3.23.4 Examples

HLT