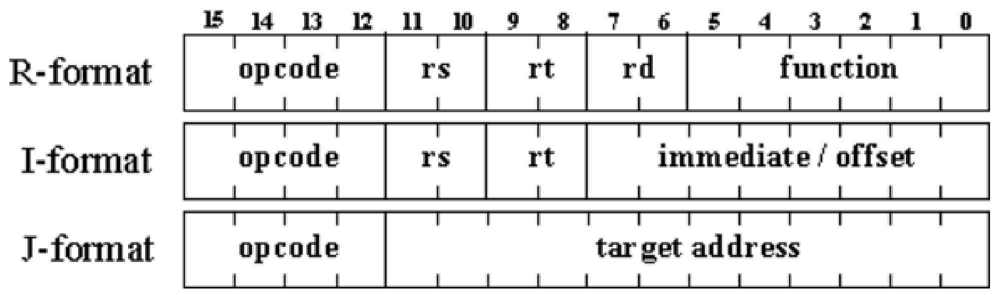
Instruction format



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
[RWD]	15	27	R
[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
[ENI]	15	30	R
[DSI]	15	31	R

Assembler Directive

- [BSC]
- [BSS]
- [END]
- [EQU]
- [ORG]

Instruction Description

ADD

1. Assembler Format
   - ADD $rd, $rs, $rt
2. Description
   - The contents of $rs and the contents of $rt are added to form the result. The result is placed into $rd.
3. Operation
   - $rd <-- $rs + $rt
4. Examples

```
o   ADD $1, $2, $0        ;$1 <-- $2 + $0
o   ADD $2, $2, $2        ;$2 <-- $2 + $2
```

SUB

1. Assembler Format
   - SUB $rd, $rs, $rt
2. Description
   - The contents of $rt are subtracted from the contents of $rs to form the result. The result is placed into $rd.
3. Operation
   - $rd <-- $rs + !$rt + 1

4. Examples

```
o   SUB $3, $0, $1       ;$3 <-- $0 - $1
o   SUB $0, $0, $0       ;$0 <-- $0 - $0  ;This clears
  $0.
```

AND

1. Assembler Format
   - AND $rd, $rs, $rt
2. Description
   - The contents of $rs are combined with the contents of $rt in a bit-wise logical AND operation. The result is placed into $rd.
3. Operation
   - $rd <-- $rs & $rt
4. Examples

```
o   AND $0, $1, $2       ;$0 <-- $1 & $2
o   AND $3, $3, $3       ;$3 <-- $3 & $3
o                      ;Contents of $3 are unchanged
```

ORR

1. Assembler Format
   - ORR $rd, $rs, $rt
2. Description
   - The contents of $rs are combined with the contents of $rt in a bit-wise logical OR operation. The result is placed into $rd.
3. Operation
   - $rd <-- $rs | $rt
4. Examples

```
o   ORR $1, $2, $1       ;$1 <-- $2 | $1
o   ORR $3, $3, $3       ;$3 <-- $3 | $3
o                      ;Contents of $3 are unchanged
```

NOT

1. Assembler Format
   - NOT $rd, $rs
2. Description
   - The bit-wise inverse of the contents of $rs are placed into $rd.
3. Operation
   - $rd <-- !$rs
4. Examples

```
o   NOT $0, $1       ;$0 <-- !$1
```

```
o   NOT $3, $3       ;$3 <-- !$3
```

TCP

1. Assembler Format
   - TCP $rd, $rs
2. Description
   - The two's complement of the contents of $rs is placed into $rd.
3. Operation
   - $rd <-- !$rs + 1
4. Examples

```
o   TCP $0, $2       ;$0 <-- !$2 + 1
o   TCP $1, $1       ;$1 <-- !$1 + 1
```

SHL

1. Assembler Format
   - SHL $rd, $rs
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. Operation
   - $rd <-- $rs14..0 ## 0
4. Examples

```
o   SHL $0, $1       ;$0 <-- $1 << 1
```

SHR

1. Assembler Format
   - SHR $rd, $rs
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. Operation
   - $rd <-- $rs15 ## $rs15..1
4. Examples

```
o   SHR $2, $1       ;$2 <-- $1 >> 1
o   SHR $1, $1       ;$1 <-- $1 >> 1
```

ADI

1. Assembler Format
   - ADI $rt, $rs, imm
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. Operation
   - $rt <-- $rs + { (imm7)8 ## imm7..0 }
4. Examples

```
o   ADI $0, $1, -17       ;$0 <-- $1 - 17
```

ORI

1. Assembler Format
   - ORI $rt, $rs, imm
2. Description
   - The 8-bit immediate value is zero-extended to 16 bits and combined with the contents of $rs in a bit-wise logical OR operation. The result is placed into $rt.
3. Operation
   - $rt <-- $rs | (08 ## imm7..0)
4. Examples

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

LHI

1. Assembler Format
   - LHI $rt, imm
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. Operation
   - $rt <-- imm7..0 ## 08
4. Examples

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

RWD

1. Assembler Format
    - RWD $rd
2. Description
    - Program execution is halted until an input signal is received on the input port. A 16-bit word is read from the input port and placed into $rd.
3. Operation
    - $rd <-- inputport
4. Examples

```
o   RWD $2       ;$2 <-- inputport
```

WWD

1. Assembler Format
    - WWD $rs
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. Operation
    - outputport <-- $rs
4. Examples

```
o   WWD $0       ;outputport <-- $0
```

LWD

1. Assembler Format
    - LWD $rt, $rs, offset
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. Operation
    - $rt <-- M[$rs + { (offset7)8 ## offset7..0 }]
4. Examples

```
o   LWD $0, $0, 16       ;$0 <-- M[$0 + 16]
o   LWD $0, $3, -4       ;$0 <-- M[$3 - 4]
o   LWD $0, $3, 0        ;$0 <-- M[$3]
```

SWD

1. Assembler Format
    - SWD $rt, $rs, offset
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. Operation
   - M[$rs + { (offset7)8 ## offset7..0 }] <-- $rt
4. Examples

```
o   SWD $1, $1, 120      ;M[$1 + 120] <-- $1
o   SWD $1, $3, -2       ;M[$3 - 2] <-- $1
o   SWD $1, $3, 0        ;M[$3] <-- $1
```

BNE

1. Assembler Format
   - BNE $rs, $rt, offset
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. Operation
   - If $rs != $rt then $pc <-- $pc + { (offset7)8 ## offset7..0 }
4. Examples

```
o   BNE $0, $2, 6     ;If $0 != $2 then $pc <-- $pc + 7
o   BNE $1, $2, LOOP1  ;If $1 != $2 then $pc <-- LOOP1
o   ;The last example is the common usage.
o   ;Note that if a label is in the offset field in
o   ;the assembly instruction, the assembler
o   ;will compute the offset and insert it into
o   ;the binary instruction.  If a number is in
o   ;the offset field, the number itself will be
o   ;inserted into the binary instruction.
```

BEQ

1. Assembler Format
   - BEQ $rs, $rt, offset
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. Operation

- If $rs == $rt then $pc <-- $pc + { (offset7)8 ## offset7..0 }

4. Examples

```
o   BEQ $0, $3, 6   ;If $0 == $3 then $pc <-- $pc + 7
o   BEQ $2, $0, -36  ;If $2 == $0 then $pc <-- $pc - 35
o   BEQ $0, $2, BYE  ;If $0 == $2 then $pc <-- BYE
o   ;The last example is the common usage.
o   ;Note that if a label is in the offset field in
o   ;the assembly instruction, the assembler
o   ;will compute the offset and insert it into
o   ;the binary instruction.  If a number is in
o   ;the offset field, the number itself will be
o   ;inserted into the binary instruction.
```

BGZ

1. Assembler Format
   - BGZ $rs, offset
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. Operation
   - If $rs > 0 then $pc <-- $pc + { (offset7)8 ## offset7..0 }
4. Examples

```
o   BGZ $3, 42    ;If $3 > 0 then $pc <-- $pc + 43
o   BGZ $2, -3    ;If $2 > 0 then $pc <-- $pc - 2
o   BGZ $0, N2    ;If $0 > 0 then $pc <-- N2
o   ;The last example is the common usage.
o   ;Note that if a label is in the offset field in
o   ;the assembly instruction, the assembler
o   ;will compute the offset and insert it into
o   ;the binary instruction.  If a number is in
o   ;the offset field, the number itself will be
o   ;inserted into the binary instruction.
```

BLZ

1. Assembler Format
   - BLZ $rs, offset
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. Operation
   - If $rs < 0 then $pc <-- $pc + { (offset7)8 ## offset7..0 }
4. Examples

```
o   BLZ $3, 200      ;Is this valid?  Why or why not?
o   BLZ $0, -17   ;If $0 < 0 then $pc <-- $pc - 16
o   BLZ $3, JEFF   ;If $3 < 0 then $pc <-- JEFF
o   ;The last example is the common usage.
o   ;Note that if a label is in the offset field in
o   ;the assembly instruction, the assembler
o   ;will compute the offset and insert it into
o   ;the binary instruction.  If a number is in
o   ;the offset field, the number itself will be
o   ;inserted into the binary instruction.
```

JMP

1. Assembler Format
   - JMP target
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. Operation
   - $pc <-- $pc15..12 ## target11..0
4. Examples

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

JAL

1. Assembler Format
   - JAL target
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. Operation
   - $2 <-- $pc $pc <-- $pc15..12 ## target11..0
4. Examples

```
o   JAL LOOP       ;$2 <-- $pc     ;$pc <-- LOOP
o                 ;if four high-order bits of $pc are 0
o   JAL 0x05b0     ;$2 <-- $pc     ;$pc <-- 0x95b0
o                 ;if $pc previously contained 0x91ae
o   JAL 0xff34     ;Is this valid?  No, why not?
o                 ;JAL is normally used to jump to a
   procedure.
o                 ;Why isn't JMP used instead?
```

JPR

1. Assembler Format
   - JPR $rs
2. Description
   - The contents of $rs are written to the PC. Program execution continues with the instruction at that address.
3. Operation
   - $pc <-- $rs
4. Examples

```
o   JPR $2     ;$pc <-- $2
o   ;JPR is commonly used in conjunction with JAL to
o   ;return from procedure calls.
o   ;JPR and JRL are the only control instructions that
o   ;allow branching to any address in physical memory.
```

JRL

1. Assembler Format
   - JRL $rs
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. Operation
   - $2 <-- $pc $pc <-- $rs
4. Examples

```
o   JRL $3     ;$2 <-- $pc     ;$pc <-- $3
o   JRL $2     ;What happens when this is executed?
o   ;JRL is commonly used to jump to procedures that
o   ;are out of range for a JAL instruction.
```

```
o    ;JPR and JRL are the only control instructions that
o    ;allow branching to any address in physical memory.
```

HLT

1. Assembler Format
   - HLT
2. Description
   - Indicates the end of a program. When executed, the machine does not fetch the next instruction.
3. Operation
   - The machine halts.
4. Examples

```
o    HLT
```

ENI

1. Assembler Format
   - ENI
2. Description
   - If interrupts are disabled, interrupts are re-enabled. Otherwise nothing occurs.
3. Operation
   - Unknown at this time.
4. Examples

```
o    ENI       ;Enables interrupts
```

DSI

1. Assembler Format
   - DSI
2. Description
   - If interrupts are enabled, interrupts are disabled. Otherwise nothing occurs.
3. Operation
   - Unknown at this time.
4. Examples

```
o    DSI       ;Disables interrupts
```

Assembler Directive Descirption

BSC

1. Assembler Format
    - BSC operand list
2. Description
    - The BSC directive reserves blocks of memory for data storage and initializes the locations in the block to the values in the comma-delimited operand list (comma-delimited means the elements in a list are separated by commas.) The number of elements in the operand list determines the number of memory locations to be reserved. A label is optional but BSC is generally useless without one. The elements in the operand list can be literal numbers (in hex or decimal), symbolic names, or mathematical expressions. If the operand is a symbolic name (label), it must have been previously defined. The elements can be positive or negative.
3. Examples

```
o   NEG2 .BSC -2   ;-2 stored at NEG2
o   X    .BSC 0x0005, 10, 0x000f, -5
o           ;The X block has 4 words initialized
o           ;to the values in the list
o   Y    .BSC X+1, X+2   ;The value X+1 is stored at Y
o                       ;The value X+2 is stored at Y+1
```

See the sample code for an example of how the values defined by BSS are accessed by other instructions.

BSS

1. Assembler Format
    - BSS operand
2. Description
    - The BSS directive reserves blocks of memory for data storage. The operand indicates the number of memory locations to be reserved. The TSC Assembler initializes all locations in the block to 0x0000. A label is optional but BSS is generally useless without one. The operand can be a literal number (in hex or decimal), a symbolic name, or a mathematical expression although expressions and symbolic names usually are not useful in the context of BSS. If the operand is a symbolic name (label), it must have been previously defined. The operand must be positive.
3. Examples

```
o   A    .BSS 10      ;The A block has 10 words
o   B    .BSS 0x0011  ;The B block has 17 words
```

See the sample code for an example of how the locations reserved by BSS are accessed by other instructions.

END

1. Assembler Format

   - END operand
2. Description
   - The END directive marks the physical end of the program. All code after END is discarded. All TSC programs must have an END directive. END cannot have a label because it does not translate into an instruction or reserved memory location. The operand is optional and is ignored by the assembler.
3. Examples

```
o   .END        ;Pretty simple
o   .END BEGIN   ;The operand indicates where the
   program began
```

EQU

1. Assembler Format
   - label .EQU operand
2. Description
   - The EQU directive assigns the operand value to the label. The value can represent a memory location or a data constant. The operand can be a literal number (in hex or decimal), a symbolic name, or a mathematical expression. If the operand is a symbolic name (label), it must have been previously defined. The value of the operand can be positive or negative. EQU is not needed to write TSC programs but is provided as a convenience to the advanced assembly programmer.
3. Examples

```
o   ;EQU used to duplicate label values
o   X    OR $3, $2, $0
o   Y    .EQU X         ;Y=X
o   ;EQU used to create constants
o   A    .EQU 56        ;A=56
o   B    .EQU A+4       ;B=60
o        ADI $2, $2, B-A  ;Add 4 to $2
```

ORG

1. Assembler Format
   - ORG address
2. Description
   - The ORG directive provides the assembler with the memory address where the next instruction is to be placed. ORG is often on the first line of the program but is not required there (the TSC Assembler defaults to address zero if ORG is not found.) ORG is the means by which separate program segments are created. The address operand can be a literal number (in hex or decimal), a symbolic name, or a mathematical expression. If the operand is a symbolic name (label), it

must have been previously defined. ORG cannot have a label because it does not translate into an instruction or reserved memory location.

3. Examples

```
o   .ORG 0x56ff    ;Next instruction placed at 0x56ff
o   .ORG START     ;Next instruction placed at the
   value
o                 ;of the label START
o   .ORG START+3   ;Next instruction placed at START+3
```

See the sample code for an example of how ORG is used to create segments.