Appendix

A Instruction Set

Each MIPS microprocessor instruction is 32 bits long. The instructions come in 3 structures. The first is R-type instructions (or sometimes known as “Special”):

A.1 R-type

The R-type instructions take 2 register inputs (RS and RT) and one output register (RD). An operation is applied to the input values and the result is placed in the destination register. The operations that are available for this instruction type are:

ADD	Simple arithmetic Addition	RD = RS - RT
ADDU	As ADD but does not cause an interrupt when on an Overflow.	
SUB	Simple arithmetic Subtract	RD = RS - RT
SUBU	As SUB but does not cause an interrupt when on an Overflow.	
AND	Bitwise And	RD = RS & RT
OR	Bitwise Or	RD = RS | RT
XOR	Bitwise Xor	RD = RS ^ RT
NOR	Bitwise Nor	RD =~RS |~RT
SLT	Set RD to 1 if RS is less than RT else set RD to 0	
		If (RS<RT) RD=1 else RD=0
SLTU	!!!!!!!!!! Shaft didn’t implement fool!!! (did I?)	
JR	Move RS to the program counter	
MUL	Multiply	HI:LO = RS * RT
MULU	Multiply	UnsignedHI:LO = RS * RT
DIV	Divide	LO = RS / RT Remainder HI
DIVU	Divide Unsigned	LO = RS / RT Remainder HI
SYSCALL	Cause an exception	
BREAK	Cause an exception (The kernel knows which one was called)	

The R-type instruction is also capable of doing shifts. The Shift operation takes an input Register to be sifted (RT) a destination register (RD) and either a 5 bit immediate value (SA) or another register (RS). Register RT is shifted by the number specified in either the SA value or the bottom 5 bits of register RS.

The operations that are available for this instruction type are:

SLL	Shift logical left filling in 0's
SRL	Shift logical right filling in 0's
SRA	Shift arithmetic right filling in RT_{31}

The encoded instruction looks like so:

```
33222222222211111111110000000000
10987654321098765432109876543210  Bit number
000000|RS ||RT ||RD ||SA ||OPP |  Field
  6     5    5    5    5    6     Field Size
```

A.2 I-type

The I-type instructions take 1 input register (RS) a 16 bit immediate value (IMM) and a destination register (RT). The IMM is sign extended on all arithmetic operations. An operation is applied to the input values and the result is placed in the destination register. The operations that are available for this instruction type are:

ADDI	Simple arithmetic Add	RT = RS + IMM
ADDIU	As ADD but does not cause an interrupt when on an Overflow.	
ANDI	Bitwise And	RT = RS & IMM
ORI	Bitwise Or	RT = RS | IMM
XORI	Bitwise Xor	RT = RS ^ IMM
LUI	Load to the top of the register	RT = IMM<<16
SLTI	Set RT to 1 if RS is less than IMM else set RT to 0	
		If (RS<IMM) RD=1 else RD=0
SLTIU	!!!!!!!!!! Shaft didn't implement! fool!!! (Or did I?) LOOK UP!	

The I-type instructions are also used to do branches. The branch instructions take a 16bit immediate (IMM) and two registers (RS and RT). RS and RT are tested and if they match the condition then the branch may proceed. To execute the branch the current Immediate after being shifted by two and sign extended is added to the current PC. This allows branches of up to 32K in either direction. The operations that are available for this instruction type are:

BGEZ	Branch if Greater or Equal to Zero	Br if RS >= 0

BLTZ	Branch if Less Than Zero	Br if RS < 0
BGTZ	Branch if Greater Than Zero	Br if RS > 0
BLEZ	Branch if Less or Equal to Zero	Br if RS <= 0
BEQ	Branch if EQual	Br if RS == RT
BNE	Branch if Not Equal	Br if RS != RT

The memory access instructions are also encoded in I-type instructions. The immediate is sign extended and added to RS to create an effective address.

LW	Load Word	RT = $[RS+IMM]^{32}$
LH	Load Half	RT = $[RS+IMM]^{16}$
LHU	Load Half Unsigned	RT = $0^{16}\|\|[RS+IMM]^{16}$
LB	Load Byte	RT = $[RS+IMM]^{8}$
LHU	Load Half Unsigned	RT = $0^{24}\|\|[RS+IMM]^{8}$
LWL	Load Word Left	See below
LWR	Load Word Right	See below
SW	Store Word	$[RS+IMM]^{32}$ = RT
SH	Store Half	$[RS+IMM]^{16}$ = RT
LB	Store Byte	$[RS+IMM]^{8}$ = RT
LWL	Store Word Left	See below
LWR	Store Word Right	See below

Load/Store Word Left/Right instructions actions depend upon the bottom two bits of the address. The following four tables show the behaviour of the four instructions. The tables show each of the four bytes in the words in memory and register.

TABLE 1. Load Word Left

Register	**A**	**B**	**C**	**D**
Memory	**W**	**X**	**Y**	**Z**
$Addr_{1..0}$=0	Z	B	C	D
$Addr_{1..0}$=1	Y	X	C	D
$Addr_{1..0}$=2	X	Y	Z	D
$Addr_{1..0}$=3	W	X	Y	Z

TABLE 2. Load Word Right

Register	A	B	C	D
Memory	W	X	Y	Z
$Addr_{1..0}$=0	W	X	Y	Z
$Addr_{1..0}$=1	A	W	X	Y
$Addr_{1..0}$=2	A	B	W	X
$Addr_{1..0}$=3	A	B	C	W

TABLE 3. Store Word Left

Register	A	B	C	D
Memory	W	X	Y	Z
$Addr_{1..0}$=0	W	X	Y	A
$Addr_{1..0}$=1	W	X	A	B
$Addr_{1..0}$=2	W	A	B	C
$Addr_{1..0}$=3	A	B	C	D

TABLE 4. Store Word Right

Register	A	B	C	D
Memory	W	X	Y	Z
$Addr_{1..0}$=0	A	B	C	D
$Addr_{1..0}$=1	B	C	D	Z
$Addr_{1..0}$=2	C	D	Y	Z
$Addr_{1..0}$=3	D	X	Y	Z

The encoded instruction looks like so:

```
33222222222211111111110000000000
10987654321098765432109876543210  Bitnumber
|Ityp||RS ||RT ||   Immediate  |  Fields
   6    5    5         16         Field Size
```

A.3 J-type

The last instruction type is J-type. There is only one J type instruction and it takes a 26 bit immediate and places it in the bottom 26 bits of the PC.

```
33222222222211111111110000000000
10987654321098765432109876543210  Bitnumber
|Jtyp||          Target          |  Fields
   6                26             Field Size
```

Some branch instructions (BGEZ and BLTZ) as well as jumps (J and JR) can record the program counter that would have been the next instruction to be executed. This address is stored in the return address register ($31). These instructions simply have ‘AL’ added to them to make JAL, BGEZAL, etc.

The full instruction encoding is as follows:

```
33222222222211111111110000000000
10987654321098765432109876543210   Bitnumber

R-type(Special)
000000|RS ||RT ||RD |0000010000U  ADD(U)
000000|RS ||RT ||RD |0000010001U  SUB(U)
000000|RS ||RT ||RD |00000100100  AND
000000|RS ||RT ||RD |00000100101  OR
000000|RS ||RT ||RD |00000100110  XOR
000000|RS ||RT ||RD |00000100111  NOR

000000|RS ||RT ||RD ||SA |000V00  SLL(V)
000000|RS ||RT ||RD ||SA |000V10  SRL(V)
000000|RS ||RT ||RD ||SA |000V11  SRA(V)

000000|RS |00000|RD |0000000100A  J(AL)R

000000|RS ||RT |000000000001100U  MUL(U)
000000|RS ||RT |000000000001101U  DIV(U)
0000000000000000|RD |00000010000  MFHI
000000|RS |00000000000000010001  MTHI
0000000000000000|RD |00000010010  MFLO
000000|RS |00000000000000010011  MTLO

000000|     Code          |001100  SYSCALL
000000|     Code          |001101  BREAK

I-type
00100U|RS ||RT ||   Immediate  |  ADDI(U)
00101U|RS ||RT ||   Immediate  |  SUBI(U)
001100|RS ||RT ||   Immediate  |  ANDI
001101|RS ||RT ||   Immediate  |  ORI
001110|RS ||RT ||   Immediate  |  XORI
001111|RS ||RT ||   Immediate  |  LUI

Branch
000001|RS |A0000|   Immediate  |  BGEZ(AL)
000001|RS |A0001|   Immediate  |  BLTZ(AL)
000100|RS ||RT ||   Immediate  |  BEQ
000101|RS ||RT ||   Immediate  |  BNE
000110|RS |00000|   Immediate  |  BLEZ
000111|RS |00000|   Immediate  |  BGTZ
00001A|           Target       |  J(AL)

MemoryI/O
100U00|RS ||RT ||   Immediate  |  LB(U)
100U01|RS ||RT ||   Immediate  |  LH(U)
100011|RS ||RT ||   Immediate  |  LW
100010|RS ||RT ||   Immediate  |  LWL
100110|RS ||RT ||   Immediate  |  LWR

101000|RS ||RT ||   Immediate  |  SB
```

```
101001|RS ||RT ||   Immediate  |  SH
101011|RS ||RT ||   Immediate  |  SW
101010|RS ||RT ||   Immediate  |  SWL
101110|RS ||RT ||   Immediate  |  SWR

Coprocessor
0100zz00000|RT ||RD |00000000000  MFCz
0100zz00100|RT ||RD |00000000000  MTCz
0100zz00010|RT ||RD |00000000000  CFCz
0100zz00110|RT ||RD |00000000000  CTCz
0100zz0100000000|   Immediate  |  BCzF
0100zz0100000001|   Immediate  |  BCzT
0100zz1|      Operation         |  COPz
1100zz|RS ||RT ||   Immediate  |  LWCz
1110zz|RS ||RT ||   Immediate  |  SWCz
```