TPU Instruction Set Architecture v1

Revision 2

Tuesday, 14 July 2015

Domipheus Labs

Contents

Known Issues

Missing instructions

Instructions are currently missing from this version of the ISA and will be added in time. Currently missing instructions include the following table, but it is not complete.

Save PC
Branch Conditional to relative offset
Branch to relative offset
Save Carry/Overflow

Latencies

Latencies are not fully known until CPU design is finalized, however, all instructions apart from memory read and write take a fixed amount of cycles. Memory read and write add additional latencies.

The pipelining is not operational on v1 of this CPU.

Glossary

Op	Op Code
Rd	Destination Register
F	Function flags
Ra	Source Register A
Rb	Source Register B
Imm	Immediate Value
Fs	Signed Flag
R	Reserved
X	Undefined

Instruction Forms

Definitions

- Form RRR
  - Destination and two source registers
- Form RRs
  - Two source registers
- Form RRd
  - One destination and one source register
- Form R
  - A single source register
- Form RImm

  - Destination register with 8-bit immediate value
- Form Imm
  - 8-bit immediate value

All forms have various flag and unused bit spaces that instruction operations may use.

Layout

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RRR	opcode				rD			F	rA			rB			Unused	
RRs	opcode				Unused			F	rA			rB			Unused	
RRd	opcode				rD			F	rA			Unused				
R	opcode				rD			F	Unused							
RImm	opcode				rD			F	8-bit Immediate Value							
Imm	opcode				Unused			F	8-bit Immediate Value							

Instruction Listing

add.s, add.u

Add signed or unsigned

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	**0**	**0**	**0**	Rd	Rd	Rd	**S**	Ra	Ra	Ra	Rb	Rb	Rb	R	R

Regs[Rd] = Regs[Ra] + Regs[Rb]

The bit S indicates whether the addition operation is signed. If S is 1, the operands are taken as signed integer values, unsigned otherwise.

sub.s, sub.u

Subtract signed or unsigned

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	**0**	**0**	**1**	Rd	Rd	Rd	**0**	Ra	Ra	Ra	Rb	Rb	Rb	R	R

Regs[Rd] = Regs[Ra] - Regs[Rb]

The bit S indicates whether the addition operation is signed. If S is 1, the operands are taken as signed integer values, unsigned otherwise.

or

Bitwise OR

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	**0**	**1**	**0**	Rd	Rd	Rd	**0**	Ra	Ra	Ra	Rb	Rb	Rb	R	R

Regs[Rd] = Regs[Ra] or Regs[Rb]

xor

Bitwise XOR

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	**0**	**1**	**1**	Rd	Rd	Rd	**0**	Ra	Ra	Ra	Rb	Rb	Rb	R	R

Regs[Rd] = Regs[Ra] xor Regs[Rb]

and

Bitwise AND

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	**1**	**0**	**0**	Rd	Rd	Rd	**0**	Ra	Ra	Ra	Rb	Rb	Rb	R	R

Regs[Rd] = Regs[Ra] and Regs[Rb]

not

Bitwise NOT

Instruction Form RRd

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	**1**	**0**	**1**	Rd	Rd	Rd	**0**	Ra	Ra	Ra	**0**	**0**	**0**	R	R

Regs[Rd] = not Regs[Ra]

read

Memory Read

Instruction Form RRd

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	**1**	**1**	**0**	Rd	Rd	Rd	**0**	Ra	Ra	Ra	0	0	0	R	R

Regs[Rd] = Memory[Regs[Ra]]

Reads a 16-bit word from memory at the specified location into the destination register.

write

Memory Write

Instruction Form RRs

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
0	**1**	**1**	**1**	0	0	0	**0**	Ra	Ra	Ra	Rb	Rb	Rb	R	R

Memory[Regs[Ra]] = Regs[Rb]

Writes the 16-bit word in Rb into memory at the specified location.

load.h, load.l

Load Immediate High, Load Immediate Low

Instruction Form RImm

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
1	**0**	**0**	**0**	Rd	Rd	Rd	**LF**	Im	Im	Im	Im	Im	Im	Im	Im

If LF = 1 then

Regs[Rd] = 0x00FF & Im

else

Regs[Rd] = 0xFF00 & Im<<8

End if

cmp.s, cmp.u

Compare Integers

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
1	**0**	**0**	**1**	*Rd*			*Fs*	*Ra*			*Rb*			R	R

Regs[***Rd***] = *compare* (Regs[***Ra***] , Regs[***Rb***], Fs)

Compares integer values within registers ***Ra*** and ***Rb***, placing a result bit field in ***Rd***.

If ***Fs*** is set, comparisons are treated as signed integers. The result bit field written to ***Rd*** is defined as follows. The result of the comparison goes into the associated bit of the register, a set bit indicating true.

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
X	**Ra = Rb**	**Ra > Rb**	**Ra < Rb**	**Ra = 0**	**Rb = 0**	X	X	X	X	X	X	X	X	X	X

shl

Shift Left Logical from Register

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
1	**0**	**1**	**0**	Rd	Rd	Rd	**0**	Ra	Ra	Ra	Rb	Rb	Rb	R	R

Regs[Rd] = Regs[Ra] << Regs[Rb]

shr

Shift Right Logical from Register

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
1	**0**	**1**	**1**	Rd	Rd	Rd	**0**	Ra	Ra	Ra	Rb	Rb	Rb	R	R

Regs[Rd] = Regs[Ra] >> Regs[Rb]

jump, br

branch to register location

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
1	**1**	**0**	**0**	0	0	0	**0**	Ra	Ra	Ra	0	0	0	R	R

PC = Regs[Ra]

jump, bi

Branch to immediate location

Instruction Form Imm

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
1	**1**	**0**	**0**	0	0	0	**1**	Im	Im	Im	Im	Im	Im	Im	Im

PC = 0x00FF & Im

jump.eq, jump.aez, jump.bz, jump.anz, jump.bnz, jump.gt, jump.lt

Jump if Conditional

Instruction Form RRR

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
1	**1**	**0**	**1**	0	0	0	C_2	Ra			Rb			C_1	C_0

If **C** (bits $C_2C_1C_0$ concatenated) matches with condition bits written by a CMP instruction stored in Reg[**Ra**] then

PC = Reg[**Rb**]

End if

Table of **C** bits to condition mappings.

C_2 , C_1 , C_0	Condition
0, 0, 0	EQ
0, 0, 1	Ra = 0
0, 1, 0	Rb = 0
0, 1, 1	Ra != 0
1, 0, 0	Rb != 0
1, 0, 1	Ra > Rb
1, 1, 0	Ra < Rb