TPU Instruction Set Architecture v1.2

Revision 3

August 1st 2015

Domipheus Labs

Contents

Version History

Version	Description	Date
1.0	Initial	
1.1	Added Save PC, Save Status, add immediate unsigned, sub immediate unsigned, Jump conditional to relative offset	July 31 2015
1.2	References to jumps replaced with branches. Branch conditionals with new instruction forms. Read and Write now have a signed offset in the encoding.	August 3 2015

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.

Branch to relative offset

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
C	Conditiona flags

Instruction Forms

Definitions

- Form RRR
    - Destination and two source registers
- Form RRs
    - Two source registers, optional immediate
- Form RRd
    - One destination and one source register, optional immediate
- Form CRsI
    - Ass RRd, however, rD is used as a constant flag section.
- Form CRR
    - Condition flags and two source registers
- Form R
    - A single source register
- Form RRImm
    - Destination, Source and 4-bit immediate.
- Form RImm
    - Destination register with 8-bit immediate value
- Form Imm
    - 8-bit immediate value
- Form RRImm
    - Destination reg, source reg and 4-bit immediate

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				Imm[4:2]			F	rA			rB			Imm[1:0]	
RRd	opcode				rD			F	rA			5-bit immediate				
CRsI	opcode				C			F	rA			5-bit immediate				
CRR	opcode				C			F	rA			rB			Unused	
RRImm	opcode				rD			F	rA			4-bit Immediate				?
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	**0**

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.

addi.u

Add unsigned immediate

Instruction Form RRImm

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

Regs[Rd] = Regs[Ra] + Imm

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.

subi.u

Add unsigned immediate

Instruction Form RRImm

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	Imm	Imm	Imm	Imm	**1**

```
Regs[Rd] = Regs[Ra] - Imm
```

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	Imm				

Regs[Rd] = Memory[Regs[Ra] + Imm]

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

write

Memory Write

Instruction Form RRsW

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

Memory[Regs[Ra] + Imm] = 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]

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]

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

br.eq, br.az, br.bz, br.anz, br.bnz, br.gt, br.lt

Branch Conditional

Instruction Form CRR

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

If **C** matches with condition bits written by a CMP instruction stored in Reg[**Ra**] then

PC = Reg[**Rb**]

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

bro.eq, bro.az, bro.bz, bro.anz, bro.bnz, bro.gt, bro.lt

Branch conditional to relative offset

Instruction Form CRsI

15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
1	**1**	**0**	**1**	C_2	C_1	C_0	**1**	Ra			Imm_4	Imm_3	Imm_2	Imm_1	Imm_0

Imm is signed.

If **C** matches with condition bits written by a CMP instruction stored in Reg[**Ra**] then

PC = PC + Imm

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

spc

Save PC

Instruction Form RRR

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

Regs[Rd] = PC

sstatus

Save Status

Instruction Form RRR

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

```
Regs[Rd] = Status
```