Tree: 1f06f3751d

Find file Copy path

rars / PseudoOps.txt

TheThirdOne Fix auipc and lui argument checks

1f06f37 on 7 Jun

1 contributor

Raw Blame History

199 lines (169 sloc) 15.2 KB

```
# Copyright (c) 2003-2010,  Pete Sanderson and Kenneth Vollmar
#
# Developed by Pete Sanderson (psanderson@otterbein.edu)
# and Kenneth Vollmar (kenvollmar@missouristate.edu)
#
# Permission is hereby granted, free of charge, to any person obtaining
# a copy of this software and associated documentation files (the
# "Software"), to deal in the Software without restriction, including
# without limitation the rights to use, copy, modify, merge, publish,
# distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so, subject
# to the following conditions:
#
# The above copyright notice and this permission notice shall be
# included in all copies or substantial portions of the Software.


# File containing definitions of MIPS pseudo-ops

# File format:
#   Each line contains specification for one pseudo-op, including optional
#   First item is source statement syntax, specified in same "example" par
#   Source statement specification ends with a tab.  It is followed by a t
#   templates to complete and substitute for the pseudo-op.
#   Format for specifying syntax of templates is different from specifying
#      (n=0,1,2,3,...) is token position in source statement (operator is 
#      RGn means substitute register found in n'th token of source stateme
#      LLn means substitute low order 16-bits from label address in source
#      LHn means substitute high order 16-bits from label address in sourc
#      PCLn is similar to LLn except the value substituted will be relativ
#      PCHn is similar to LHn except the value substituted will be relativ
#      VLn means substitute low order 16-bits from 32-bit value in source 
#      VHn means substitute high order 16-bits from 32-bit value in source
#      LAB means substitute textual label from last token of source statem
#   Everything else is copied as is into the generated statement (you must
#   The list of basic instruction templates is optionally followed a descr
#   To add optional description, append a tab then the '#' character follo
#
#  See documentation for ExtendedInstruction.makeTemplateSubstitutions() f
#
#  Matching for a given instruction mnemonic is first-fit not best-fit.  I
#  immediate operand options, they should be listed in that order (16-bit 
#  version will never be matched since the 32-bit version fits small immed
#
#  The pseudo-op specification must start in the first column.  If first c
#
#  When specifying the example instruction (first item on line), the conve
#  - for a register operand, specify a numbered register (e.g. $t1 or $f1)
#    The numerical value is not significant.  This is NOT the case when wr
#    In the templates, numbered registers are parsed as is (use only $0 and
#  - for an immediate operand, specify a positive value indicative of the 
#    a 5 bit value, 100 to represent a 16-bit value, and 100000 to represe
#  - for a label operand, I use the string "label" (without the quotes).
#  The idea is to give the parser an example that will be parsed into the 
#  is done by comparing the source token sequence to list of token sequence
#  IMPORTANT NOTE:  The use of $t1,$t2, etc in the instruction sample mean
#                   can be used in that position.  It is simply a placehol
#                   $1 is used in the template specification, $1 ($at) is 
#                   instruction!  If you want the generated code to echo t

#######################  arithmetic and branch pseudo-ops ################

nop ;addi x0, x0, 0 ;#NO OPeration

not t1,t2 ;xori RG1, RG2, -1 ;#Bitwise NOT (bit inversion)
mv  t1,t2 ;add RG1, x0, RG2  ;#MoVe : Set t1 to contents of t2
neg t1,t2 ;sub RG1, x0, RG2  ;#NEGate : Set t1 to negation of t2

# non-(load and store) pseudo-instructions for floating point (coprocessor
fmv.s  f1, f2 ;fsgnj.s  RG1, RG2, RG2;# Move the value of f2 to f1
fabs.s f1, f2 ;fsgnjx.s RG1, RG2, RG2;# Set f1 to the absolute value of f2
fneg.s f1, f2 ;fsgnjn.s RG1, RG2, RG2;# Set f1 to the negation of f2

sgt  t1,t2,t3 ;slt  RG1, RG3, RG2 ;#Set Greater Than : if t2 greater than 
sgtu t1,t2,t3 ;sltu RG1, RG3, RG2 ;#Set Greater Than Unsigned : if t2 grea
seqz t1,t2    ;sltiu RG1, RG2, 1  ;#Set EQual to Zero :     if t2 == 0 the
snez t1,t2    ;sltu RG1, x0, RG2  ;#Set Not Equal to Zero : if t2 != 0 the
sgtz t1,t2    ;slt RG1, x0, RG2   ;#Set Greater Than Zero : if t2 >  0 the
sltz t1,t2    ;slt RG1, RG2, x0   ;#Set Less Than Zero :    if t2 <  0 the

b label       ;jal x0, LAB      ;#Branch : Branch to statement at label un
beqz t1,label ;beq RG1, x0, LAB ;#Branch if EQual Zero : Branch to stateme
bnez t1,label ;bne RG1, x0, LAB ;#Branch if Not Equal Zero : Branch to sta
bgez t1,label ;bge RG1, x0, LAB ;#Branch if Greater than or Equal to Zero 
bltz t1,label ;blt RG1, x0, LAB ;#Branch if Less Than Zero : Branch to sta
bgtz t1,label ;blt x0, RG1, LAB ;#Branch if Greater Than: Branch to statem
blez t1,label ;bge x0, RG1, LAB ;#Branch if Less than or Equal to Zero : B
bgt  t1,t2,label ;blt  RG2, RG1, LAB ;#Branch if Greater Than : Branch to 
bgtu t1,t2,label ;bltu RG2, RG1, LAB ;#Branch if Greater Than Unsigned: Br
```

```
ble  t1,t2,label ;bge  RG2, RG1, LAB ;#Branch if Less or Equal : Branch to
bleu t1,t2,label ;bgeu RG2, RG1, LAB ;#Branch if Less or Equal Unsigned :

j label        ;jal  x0, LAB     ;#Jump : Jump to statement at label
jal label      ;jal  x1, LAB     ;#Jump And Link: Jump to statement at lab
jr t0, -100    ;jalr x0, RG1, VL2;#Jump Register: Jump to address in t0
jalr t0, -100  ;jalr x1, RG1, VL2;#Jump And Link Register: Jump to address
ret            ;jalr x0, x1, 0   ;#Return: return from a subroutine
call label     ;auipc x6,PCH1    ;jalr x1, x6, PCL1;#CALL: call a far-away
tail label     ;auipc x6,PCH1    ;jalr x0, x6, PCL1;#TAIL call: tail call

#########################  load/store pseudo-ops start here  ##############
#
#  Most of these simply provide a variety of convenient memory addressing
#  specifying load/store address.
#

li t1,-100     ;addi RG1, x0, VL2                ;#Load Immediate : Set t1
li t1,10000000 ;lui RG1, VH2 ;addi RG1, RG1, VL2 ;#Load Immediate : Set t1

la t1,label  ;auipc RG1, PCH2 ; addi RG1, RG1, PCL2;#Load Address : Set t1

lw t1,(t2)     ;lw RG1,0(RG3)   ;#Load Word : Set t1 to contents of effect
lw t1,-100     ;lw RG1, VL2(x0) ;#Load Word : Set t1 to contents of effect
lw t1,10000000 ;lui   RG1, VH2  ;lw RG1, VL2(RG1)  ;#Load Word : Set t1 to
lw t1,label        ;auipc RG1, PCH2 ;lw RG1, PCL2(RG1) ;#Load Word : Set t

sw t1,(t2)        ;sw RG1,0(RG3)   ;#Store Word : Store t1 contents into e
sw t1,-100        ;sw RG1, VL2(x0) ;#Store Word : Store $t1 contents into
sw t1,10000000,t2 ;lui   RG3, VH2  ;sw RG1, VL2(RG3)  ;#Store Word : Store
sw t1,label,t2    ;auipc RG3, PCH2 ;sw RG1, PCL2(RG3) ;#Store Word : Store

lh t1,(t2)     ;lh RG1,  0(RG3) ;#Load Halfword : Set t1 to sign-extended
lh t1,-100     ;lh RG1, VL2(x0) ;#Load Halfword : Set t1 to sign-extended
lh t1,10000000 ;lui RG1, VH2    ;lh RG1, VL2(RG1)  ;#Load Halfword : Set t
lh t1,label        ;auipc RG1, PCH2 ;lh RG1, PCL2(RG1) ;#Load Halfword : S

sh t1,(t2)        ;sh RG1,0(RG3)   ;#Store Halfword : Store the low-order
sh t1,-100        ;sh RG1, VL2(x0) ;#Store Halfword : Store the low-order
sh t1,10000000,t2 ;lui   RG3, VH2  ;sh RG1, VL2(RG3)  ;#Store Halfword : S
sh t1,label,t2    ;auipc RG3, PCH2 ;sh RG1, PCL2(RG3) ;#Store Halfword : S

```

```
lb t1,(t2)     ;lb RG1,0(RG3)   ;#Load Byte : Set t1 to sign-extended 8-bi
lb t1,-100     ;lb RG1, VL2(x0) ;#Load Byte : Set $1 to sign-extended 8-bi
lb t1,10000000 ;lui RG1, VH2    ;lb RG1, VL2(RG1)  ;#Load Byte : Set $t1 t
lb t1,label        ;auipc RG1, PCH2 ;lb RG1, PCL2(RG1) ;#Load Byte : Set $

sb t1,(t2)        ;sb RG1,0(RG3)   ;#Store Byte : Store the low-order 8 bi
sb t1,-100        ;sb RG1, VL2(x0) ;#Store Byte : Store the low-order 8 bi
sb t1,10000000,t2 ;lui   RG3, VH2  ;sb RG1, VL2(RG3)  ;#Store Byte : Store
sb t1,label,t2    ;auipc RG3, PCH2 ;sb RG1, PCL2(RG3) ;#Store Byte : Store

lhu t1,(t2)     ;lhu RG1,0(RG3)   ;#Load Halfword Unsigned : Set t1 to zer
lhu t1,-100     ;lhu RG1, VL2(x0) ;#Load Halfword Unsigned : Set t1 to zer
lhu t1,10000000 ;lui RG1, VH2     ;lhu RG1, VL2(RG1)  ;#Load Halfword Unsi
lhu t1,label    ;auipc RG1, PCH2  ;lhu RG1, PCL2(RG1) ;#Load Halfword Unsi

lbu t1,(t2)     ;lbu RG1,0(RG3)   ;#Load Byte Unsigned : Set $t1 to zero-e
lbu t1,-100     ;lbu RG1, VL2(x0) ;#Load Byte Unsigned : Set $t1 to zero-e
lbu t1,10000000 ;lui RG1, VH2    ;lbu RG1, VL2(RG1)  ;#Load Byte Unsigned 
lbu t1,label    ;auipc RG1, PCH2 ;lbu RG1, PCL2(RG1) ;#Load Byte Unsigned 

# load and store pseudo-instructions for floating point (coprocessor 1) re
flw f1,(t2)     ;flw RG1,0(RG3)  ;#Load Word Coprocessor 1 : Set f1 to 32-
flw f1,-100     ;flw RG1, VL2(x0);#Load Word Coprocessor 1 : Set f1 to 32-
flw f1,10000000,t3;lui RG3, VH2    ;flw RG1, VL2(RG3) ;#Load Word Coproces
flw f1,label, t3;auipc RG3, PCH2 ;flw RG1, PCL2(RG3);#Load Word Coprocesso

fsw f1,(t2)     ;fsw RG1,0(RG3)  #Store Word Coprocessor 1 : Store 32-bit 
fsw f1,-100     ;fsw RG1, VL2(x0);#Store Word Coprocessor 1 : Store 32-bit
fsw f1,10000000,t3;lui RG3, VH2    ;fsw RG1, VL2(RG3) ;#Store Word Coproce
fsw f1,label, t3;auipc RG3, PCH2 ;fsw RG1, PCL2(RG3);#Store Word Coprocess


######################  CSR pseudo-ops ####################

csrr t1, 100 ;csrrs RG1, RG2, x0 ;#Read control and status register
csrw t1, 100 ;csrrw x0, RG2, RG1 ;#Write control and status register
csrs t1, 100 ;csrrs x0, RG2, RG1 ;#Set bits in control and status register
csrc t1, 100 ;csrrc x0, RG2, RG1 ;#Clear bits in control and status regist

csrwi 100, 100 ;csrrwi x0, RG1, RG2 ;#Write control and status register
csrsi 100, 100 ;csrrsi x0, RG1, RG2 ;#Set bits in control and status regis
csrci 100, 100 ;csrrci x0, RG1, RG2 ;#Clear bits in control and status reg
```

```

frcsr t1     ; csrrs RG1, 0x003, x0  ;#Read FP control/status register
fscsr t1, t2 ; csrrw RG1, 0x003, RG2 ;#Swap FP control/status register
fscsr t1     ; csrrs  x0, 0x003, RG1 ;#Write FP control/status register

frrm t1      ; csrrs RG1, 0x003, x0  ;#Read FP rounding mode
fsrm t1, t2  ; csrrw RG1, 0x003, RG2 ;#Swap FP rounding mode
fsrm t1      ; csrrs  x0, 0x003, RG1 ;#Write FP rounding mode
fsrmi t1, 100 ; csrrwi RG1, 0x003, RG2 ;#Swap FP rounding mode, immediate
fsrmi 100     ; csrrsi  x0, 0x003, RG1 ;#Write FP rounding mode, immediate

frflags t1      ; csrrs RG1, 0x003, x0  ;#Read FP exception flags
fsflags t1, t2  ; csrrw RG1, 0x003, RG2 ;#Swap FP exception flags
fsflags t1      ; csrrs  x0, 0x003, RG1 ;#Write FP exception flags
fsflagsi t1, 100 ; csrrwi RG1, 0x003, RG2 ;#Swap FP exception flags, immed
fsflagsi 100     ; csrrsi  x0, 0x003, RG1 ;#Write FP exception flags, imme
```