Team 1-3

TABLE OF CONTENTS

(Ctrl-click to jump directly to a section)

Milestone 1: Assembly Language and Machine Language Specifications
Part 1: Assembly Language Specifications

The Assembly Language has sixteen registers.

0. ($zero) Zero
1. ($G0) General Purpose
2. "
3. "
4. "
5. "
6. "
7. "
8. ($G7) "
9. ($SP) Stack Pointer
10. ($RA) Return Address
11. ($DP) Data Pointer
12. ($D0) Destination
13. "
14. "
15. ($D3) "

The last four registers are special "destination registers." Most computations can only store their results in one of these four. When used as destination registers (RD's), registers $12-$15 are referred to as $d0-$d3.

These four, by convention, are saved to the stack by a called function.

There are four types of commands:

The R3 Type:
15-11: Opcode
10: Operation Flag
9-8: RD
7-4: RS1
3-0: RS2 or a 4-bit immediate value
(RD is a value from 0b00-0b11 corresponding to one of $D0-$D3. Each RS can be any register. The Operation Flag modifies the action of an operation. See the list of operations.)

For example:
add 0, 2, $G0, $G1 adds $G0 and $G1 and places the results in $D2.

The Immediate Type:
15-11: Opcode
10: Operation Flag
9-8: RD
7-0: 8-bit immediate value

For example:
li *0, 1, 5* sets the upper 8 bits of $D1 to 00001001.

The L/S Type:
15-11: Opcode
10-7: RS1
6-3: RS2
2-0: 3-bit immediate value

For example:
lw *$G2*, *$G3*, *2* loads the value at 2($G3) into $G2.

The Exceptional Type:
15-11: Opcode
10-3: 5-bit immediate value or *unused*
2-0: *unused*

For example:
lach *0b00001010* flushes the output buffer at 0b00001010 to the output device.

There first eight words in memory are actually the memory mapped registers:
0x0 Interrupt Vector – Holds the address of the interrupt handler
0x2 System Condition Register
- Bit 15: 1 if the last operation resulted in overflow
- Bit 14: 1 if the last operation resulted in zero (for arithmetic operations)
- Bit 13: 1 if the last operation resulted in a negative number
- Bit 12: 1 if the last operation resulted in a positive number

0x4 Interrupt Condition Register
- Bit 15: 1 if the reset button has been pressed
- Bit 14: 1 if the Peripheral Interrupt requires attention
- Bit 13: 1 if the Internal Interrupt 1 requires attention
- Bit 12: Internal Interrupt 2…

0x6 Interrupt Control Register
- Bit 15: Set to 0 to Globally Ignore all interrupts
- Bit 14: Set to 0 to ignore the Peripheral Interrupt
- Bit 13: Set to 0 to ignore Internal Interrupt 1
- Bit 12: Set to 0 to ignore Internal Interrupt 2…

0x8 Input – Holds each finalized input
0xA Output high – Upper 16 bits of the output register
0xC Output low – Lower 16 bit of the output register
0xE Program Counter

Following is a list of the instructions with their opcodes:

<u>R3 Instructions</u>

0x00 add	x, rd, rs, rt :	Adds rs and rt and puts the result in rd.
0x01 and	x, rd, rs, rt :	Bitwise ands rs and rt and puts the result in rd.
0x02 or	x, rd, rs, rt :	Bitwise ors rs and rt and puts the result in rd.
0x03 xor	x, rd, rs, rt :	Bitwise xors rs and rt and puts the result in rd.
0x04 not	0, rd, rs, 0:	Bitwise nots rs and puts the result in rd.
0x04 not	1, rd, rs, 0:	2s complement nots rs and puts the result in rd.
0x05 btj	0, rd, rs, i4:	Tests the *i*th bit of rs; if it is clear, jump to rd.
0x05 btj	1, rd, rs, i4:	Tests the *i*th bit of rs; if it is set, jump to rd.
0x06 btl	0, rd, rs, i4:	Tests the *i*th bit of rs; if it is clear, jump and link to rd.
0x06 btl	1, rd, rs, i4:	Tests the *i*th bit of rs; if it is set, jump and link to rd.
0x07 bs	0, rd, rs, i4:	Sets the *i*th bit of rs to 0 and puts the result in rd.
0x07 bs	1, rd, rs, i4:	Sets the *i*th bit of rs to 1 and puts the result in rd.
0x08 s	0, rd, rs, i4:	Shifts rs left *i* bits and puts the result in rd.
0x08 s	1, rd, rs, i4:	Shifts rs right *i* bits and puts the result in rd.

Immediate Instructions

0x09 li	0, rd, i8 :	Sets the upper 8 bits of rd to *i*.
0x09 li	1, rd, i8 :	Sets the lower 8 bits of rd to *i* and sets the upper 8 to 0.

LS Instructions

0x0A lw	rs, rt, i3:	Loads *i*(rt) into rs. (register is byte addressing, immediate is word-addressing)
0x0B sw	rs, rt, i3:	Stores rs into *i*(rt). (register is byte addressing, immediate is word-addressing)
0x0C copy	rs, rt, 0 :	Copies rt into rs.
0x0D sex	rs, rt, 0 :	Sign extends rt and puts the result in rs.

Exceptional Instructions

0x0E latch	i5, 0:	Flushes the given 32-bit output buffer to the output device. (*i5 is the address of the lower of the two output buffer registers*)
0x0F rfi	0:	Returns from an interrupt.

Pseudo (Compound) Instructions

ubj	rd	Shorthand for btj 0, rd, $zero, x
ubl	rd	Shorthand for btl 0, rd, $zero, x
clr	rs	Shorthand for copy rs, $zero, 0
sub	rd, rs, rt	Shorthand for: not 1, rd, rs add rd, rd, rt

NOTE: To jump, one must test a bit of a register. Usually, this is the System Condition Register. It must be loaded into a register with lw rs, $zero, 0b001.

For example, to jump if $g0 and $g1 are equal:

```
not     1, $d0, $g1             # $d0 = -1 times $g1.
add     x, $d0, $g0, $d0        # Effectively, we've subtracted $g1 from $g0.
lw      $g2, $zero, 0b001       # Load System Condition Register
li      1, $d0, 0x00            # Load jump target into $d0
li      0, $d0, 0x23
btj     1, $d0, $g2, 0xE        # Jump if the 14th bit of $g2 is 1.
```

If $g0 and $g1 are equal, their subtraction results in a zero, which sets the fourteenth bit of the system condition register to 1 (see list of mem-mapped registers). We then load the system condition register and test this bit for a jump. Similarly, one can test for greater-than and less-than using the corresponding bits of the system condition register.

Part 2: Sample Programs

Following is a program to calculate the relatively prime value to an input number using Euclid's algorithm.

```
# Group 01-03
#  Gregory Weir
#  Adrian Rutledge
#  Angela Smiley
#  Joe Wegehaupt
#  Nathan Carlson
#
# GCD program in our assembly language
#  - gets input from an external device via an interrupt
#  - result is sent to output

main:

0x40    01001 1 00 00000000    #li     1, $d0, a(i_handler)
0x42    01001 0 00 10111100    #li     0, $d0, a(i_handler)
0x44     01011 1100 0000 000   #sw     $d0, $0, 0              # interrupt handler

spin:

0x46     01001 1 00 00000000   #li     1, $d0, a(spin)         # until we get a go
                                                               # interrupt, do nothing
0x48    01001 0 00 01000110    #li     0, $d0, a(spin)
0x4a    00101 0 00 0000 0000   #btj    0, $d0, $0, 0

output:

0x4c     01011 0001 0000 110   #sw     $g0, $0, 6
0x4e     01011 0000 0000 101   #sw     $0, $0, 5
0x50     01110 00001010 000    #latch  0xA                     # send output
0x52    01001 1 00 00000000    #li     1, $d0, a(spin)
0x54    01001 0 00 01000110    #li     0, $d0, a(spin)
0x56    00101 0 00 0000 0000   #btj    0, $d0, $0, 0           # spin again

loop:

0x58    01001 1 00 00000010    #li     1, $d0, 2               # store 2 in $d0
0x5a    01100 0001 1100 000    #copy   $g0, $d0                # set $g0 (m) to 2
0x5c    01001 0 00 00000000    #li     1, $d0, a(gcd)
0x5e    01001 0 00 01110110    #li     0, $d0, a(gcd)
0x60    00110 0 00 0000 0000   #btl    0, $d0, $0, 0           # load + jump/link to gcd
0x62    01001 1 01 11111111    #li     1, $d1, -1
0x64    01101 1101 1101 000    #sex    $d1, $d1        # load + sign-extend 8-bit -1 to $d1
0x66    00000 0 11 1111 1101   #add    0, $d3, $d3, $d1        # add -1 to ret from gcd($d3)
0x68    01010 1101 0000 001    #lw     $g3, $0, 1              # load system cond. res
0x6a    01001 1 00 00000000    #li     1, $d0, a(output)
0x6c    01001 0 00 01001100    #li     0, $d0, a(output)       # load address of output, we
                                                               # are done
0x6e    00101 0 00 1101 1110   #btj    0, $d0, $g3, 14         # if zero flag is cleared, go
                                                               # to output(if ret != 1)
0x70    01001 1 00 00000001    #li     1, $d0, 1
0x72    00000 0 00 1100 0001   #add    0, $d0, $d0, $g0        # add 1 to m
0x74    01100 0001 1100 000    #copy   $g0, $d0
0x76    01001 1 00 00000000    #li     1, $d0, a(loop)
0x78    01001 0 00 01010000    #li     0, $d0, a(loop)
0x7a    00101 0 00 0000 000    #btj    0, $d0, $0, 0       # load & go back to start of loop

gcd:

0x7c    01100 0011 0000 000    #copy   $g2, $0                 #clear g2

loopgcd:

0x7e    01001 1 10 00000000    #li     1, $d2, a(exitgcd)      # load the address of exitgcd
0x80    01001 0 10 10110110    #li     0, $d2, a(exitgcd)
0x82    00000 0 01 0001 0000   #add    0, $d1, $g0, $0         # add b to 0
0x84    00100 0100 0000 001    #lw     $g3, $0, 1              # get condition register
0x86    00101 1 10 0100 1110   #btj    1, $d2, $g3, 14         # jump if zero
0x88    00101 1 00 0100 1101   #btj    1, $d2, $g3, 13         # jump if negative
0x8a    00100 1 10 0010 0000   #not    1, $d2, $g1
0x8c    01001 0 00 00000001    #li     0, $d0, 1
0x8e    00000 0 00 1100 1110   #add    0, $d0, $d0, $d2
0x90    00000 0 00 0001 1100   #add    0, $d0, $g0, $d0        # subtract b - a
0x92    01010 0100 0000 001    #lw     $g3, $0, 1              # load the condition register
0x94    01001 1 10 00000000    #li     1, $d2, a(ifgcd)        # load the address of ifgcd
```

```
0x96    01001 0 10 10101010   #li     0, $d2, a(ifgcd)
0x98    00101 1 10 0100 1100  #btj    1, $d2, $g3, 12        # jump if subtraction result
                                                             # positive
0x9a    00100 1 10 0001 0000  #not    1, $d2, $g0
0x9c    01001 0 00 00000001   #li     0, $d0, 1
0x9e    00000 0 00 1100 1110  #add    0, $d0, $d0, $d2
0xa0    00000 0 00 0010 1100  #add    0, $d0, $g1, $d0       # subtract a - b
0xa2    01100 0010 1100 000   #copy   $g1, $d0               # a = a - b
0xa4    01001 1 10 00000000   #li     1, $d2, a(loopgcd)     # load the address of loopgcd
0xa6    01001 0 10 01111110   #li     0, $d2, a(loopgcd)
0xa8    00101 0 10 0000 0000  #btj    0, $d2, $0, 0

ifgcd:

0xaa    01100 0011 0010 000   #copy   $g2, $g1               # temp = a
0xac    01100 0010 0001 000   #copy   $g1, $g0               # a = b
0xae    01100 0001 0011 000   #copy   $g0, $g2               # b = temp
0xb0    01001 1 10 00000000   #li     1, $d2, a(loopgcd)     # load the address of loopgcd
0xb2    01001 0 10 01111110   #li     0, $d2, a(loopgcd)
0xb4    00101 0 10 0000 0000  #btj    0, $d2, $0, 0

exitgcd:

0xb6    01100 1111 0010 000   #copy $d3, $g1                 # $d3 = a
0xb8    01100 1110 1010 000   #copy $d2, $ra
0xba    00101 0 10 0000 0000  #btj    0, $d2, $0, 0          # return


################## ISR ###################################
i_handler:

0xbc    01010 1101 0000 011   #lw     $g3, $0, 3     # load interrupt control register
0xbe    00111 0 00 0100 1111  #bs     0, $d0, $g3, 15               # set global disable
0xc0    01011 1100 0000 011   #sw     $d0, $0, 3             # disable interrupts
0xc2    01010 0100 0000 010   #lw     $g3, $0, 2     # load interrupt condition register
0xc4    01001 1 10 00000000   #li     1, $d2, 0x00
0xc6    01001 0 10 11011100   #li     0, $d2, a(input)
0xc8    00101 1 10 0100 1110  #btj    1, $d2, $g3, 14        # if input interrupt
                                                             # happened, jump
0xca    01001 1 10 00000000   #li     1, $d2, 0x00
0xcc    01001 0 10 11110000   #li     0, $d2, a(go)
0xce    00101 1 10 0100 1101  #btj    1 ,$d2, $g3, 13     # if go interrupt happened, jump
0xd0    01010 1101 0000 011   #lw     $g3, $0, 3          # load interrupt control register
0xd2    00111 1 00 0100 1111  #bs     1, $d0, $g3, 15               # set global enable
0xd4    01011 1100 0000 011   #sw     $d0, $0, 3             # enable interrupts
0xd6    01001 1 10 00000000   #li     1, $d2, a(spin)
0xd8    01001 0 10 01000110   #li     0, $d2, a(spin)
0xda    00101 0 10 0000 0000  #btj    0, $d2, $0, 0          # start spinning again

input:

0xdc    01010 0010 0000 0100  #lw     $g1, $0, 4             # load the input into $g1
0xde    01010 0100 0000 0010  #lw     $g3, $0, 2             # load the interrupt
                                                             # condition register
0xe0    00111 1 00 0100 1110  #bs     0, $d0, $g3, 14        # we have serviced the
                                                             # interrupt, set the bit to 0
0xe2    01011 1100 0000 010   #sw     $d0, $0, 2
0xe4    01010 1101 0000 011   #lw     $g3, $0, 3          # load interrupt control register
0xe6    00111 1 00 0100 1111  #bs     1, $d0, $g3, 15               # set global enable
0xe8    01011 1100 0000 011   #sw     $d0, $0, 3             # enable interrupts
0xea    01001 1 10 00000000   #li     1, $d2, 0x00
0xec    01001 0 10 01000110   #li     0, $d2, a(spin)
0xee    00101 0 10 0000 0000  #btj    0, $d2, $0, 0          # start spinning again

go:

0xf0    01010 0100 0000 0010  #lw     $g3, $0, 2
0xf2    00111 1 00 0100 1101  #bs     0, $d0, $g3, 13
0xf4    01011 1100 0000 010   #sw     $d0, $0, 2
0xf6    01010 1101 0000 011   #lw     $g3, $0, 3          # load interrupt control register
0xf8    00111 1 00 0100 1111  #bs     1, $d0, $g3, 15               # set global enable
0xfa    01011 1100 0000 011   #sw     $d0, $0, 3             # enable interrupts
0xfc    01001 1 10 00000000   #li     1, $d2, 0x00
0xfe    01001 0 10 01000110   #li     0, $d2, 0xa(loop)
0x100   00101 0 10 0000 0000  #btj    0, $d2, $0, 0          # start executing program
```

Following is a program to test the remaining instructions.

```
# Group 01-03
#  Gregory Weir
#  Adrian Rutledge
#  Angela Smiley
#  Joe Wegehaupt
#  Nathan Carlson
# revision 1.1
# tests each instruction and stores 0xBEEF in memory locations corresponding
#  to the instuction
# starts at 0x01d2
#
# 0x01d2 - add
# 0x01d4 - subtract
# 0x01d6 - and
# 0x01d8 - or
# 0x01da - xor
# 0x01dc - not
# 0x01de - not (2's complement)
# 0x01e0 - bit clear
# 0x01e2 - bit set
# 0x01e4 - shift left
# 0x01e6 - shift right
# 0x01e8 - li
# 0x01ea - lw/sw
# 0x01ec - copy
# 0x01ee - sex


#should start at 0x0040, but i screwed up, so i needed to put an instruction
# at 0x003e to load the full address of the "data area"

#main:

0x003e  01001 1 00 11010010   #li     1, $d0, 0xd2
0x0040  01001 0 00 00000001   #li     0, $d0, 1      # address of test info
0x0042  01001 1 00 00000101   #li     1, $d1, 5      # number to be used in operations
0x0044  01100 0001 1101 000   #copy   $g0, $d1
0x0046  01001 1 00 00000110   #li     1, $d1, 6      # ditto
0x0048  01100 0010 1101 000   #copy   $g1, $d1
0x004a  01001 1 01 11101111   #li     1, $d1, 239    # 0xbeef
0x004c  01001 0 01 10111110   #li     0, $d1, 190
0x004e  01100 0110 1101 000   #copy   $g5, $d1
0x0050  01001 1 01 00000010   #li     1, $d1, 2      # increment for the mem address
0x0052  01100 0111 1101 000   #copy   $g6, $d1

#add1t:

0x0054  00000 0 01 0001 0010  #add    0, $d1, $g0, $g1       # add the two test numbers
0x0056  01001 1 11 00001011   #li     1, $d3, 11             # load the predicted result
0x0058  00000 1 01 1101 1111  #add    1, $d1, $d1, $d3   # subtract actual from predicted
0x005a  01010 0100 0000 001   #lw     $g3, $0, 1             # load the condition register
0x005c  01001 1 10 01100100   #li     1, $d2, a(addt2)       # load address of next test
0x005e  01001 1 10 00000000   #li     0, $d2, a(addt2)
0x0060  00101 0 10 0100 1110  #btj    0, $d2, $g3, 14        # test if not zero
0x0062  01011 0110 1100 000   #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#add2t:

0x0064  00000 0 00 1100 0111  #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x0066  00000 1 01 0010 0001  #add    1, $d1, $g1, $g0       # subtract the test numbers
0x0068  01001 1 11 00000001   #li     1, $d3, 1              # load the predicted result
0x006a  00000 1 01 1101 1111  #add    1, $d1, $d1, $d3 # subtract the predicted and actual
0x006c  01010 0100 0000 001   #lw     $g3, $0, 1             # load the condition register
0x006e  01001 1 10 01110110   #li     1, $d2, a(andt)
0x0070  01001 0 10 00000000   #li     0, $d2, a(andt)
0x0072  00101 0 10 0100 1110  #btj    0, $d2, $g3, 14               # test if not zero
0x0074  01011 0110 1100 000   #sw     $g5, $d0, 0    # store 0xBEEF in the mem location

#andt:

0x0076  00000 0 00 1100 0111  #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x0078  00001 0 01 0001 0010  #and    $d1, $g0, $g1          # and the two test numbers
0x007a  01001 1 11 00000100   #li     1, $d3, 4              # load the predicted
0x007c  00000 1 01 1101 1111  #add    1, $d1, $d1, $d3       # sub. actual and predicted
0x007e  01010 0100 0000 001   #lw     $g3, $0, 1             # load the condition register
0x0080  01001 1 10 10001000   #li     1, $d2, a(ort)
0x0082  01001 0 10 00000000   #li     0, $d2, a(ort)
0x0084  00101 0 10 0100 1110  #btj    0, $d2, $g3, 14        # test if not zero
```

```
0x0086  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#ort:

0x0088  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x008a  00010 0 01 0001 0010   #or     $d1, $g0, $g1
0x008c  01001 1 11 00000111    #li     1, $d3, 7
0x008e  00000 1 01 1101 1111   #add    1, $d1, $d1, $d3       # sub. actual and predicted
0x0090  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x0092  01001 1 10 10011010    #li     1, $d2, a(xort)
0x0094  01001 0 10 00000000    #li     0, $d2, a(xort)
0x0096  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x0098  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#xort:

0x009a  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x009c  00011 0 01 0001 0010   #xor    $d1, $g0, $g1
0x009e  01001 1 11 00000011    #li     1, $d3, 3
0x00a0  00000 1 01 1101 1111   #add    1, $d1, $d1, $d3       # sub. actual and predicted
0x00a2  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x00a4  01001 1 10 10101100    #li     1, $d2, a(not1t)
0x00a6  01001 0 10 00000000    #li     0, $d2, a(not1t)
0x00a8  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x00aa  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#not1t:
0x00ac  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x00ae  00100 0 01 0001 0000   #not    0, $d1, $g0
0x00b0  01001 1 11 11111010    #li     1, $d3, -6
0x00b2  01001 0 11 11111111    #li     0, $d3, 255
0x00b4  00000 1 01 1101 1111   #add    1, $d1, $d1, $d3       # sub. actual and predicted
0x00b6  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x00b8  01001 1 10 11000000    #li     1, $d2, a(not2t)
0x00ba  01001 0 10 00000000    #li     0, $d2, a(not2t)
0x00bc  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x00be  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#not2t:

0x00c0  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x00c2  00100 1 01 0001 0000   #not    1, $d1, $g0
0x00c4  01001 1 11 11111011    #li     1, $d3, -5
0x00c6  01001 0 11 11111111    #li     0, $d3, 255
0x00c8  00000 1 01 1101 1111   #add    1, $d1, $d1, $d3       # sub. actual and predicted
0x00ca  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x00cc  01001 1 10 11010100    #li     1, $d2, a(bs1t)
0x00ce  01001 0 10 00000000    #li     0, $d2, a(bs1t)
0x00d0  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x00d2  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#bs1t:

0x00d4  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x00d6  01001 1 10 00000100    #li     1, $d2, 4
0x00d8  00111 0 01 10 0011     #bs     0, $d1, $d2, 3
0x00da  01001 1 11 00000000    #li     1, $d3, 0
0x00dc  00000 1 01 1101 1111   #add    1, $d1, $d1, $d3       # sub. actual and predicted
0x00de  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x00e0  01001 1 10 11101000    #li     1, $d2, a(bs2t)
0x00e2  01001 0 10 00000000    #li     0, $d2, a(bs2t)
0x00e4  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x00e6  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#bs2t:

0x00e8  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x00ea  01001 1 11 00000000    #li     1, $d3, 0
0x00ec  01100 0100 1111 000    #copy   $g3, $d3
0x00ee  00111 1 01 0100 0011   #bs     1, $d1, $g3, 3
0x00f0  01001 1 11 00000100    #li     1, $d3, 4
0x00f2  00000 1 01 0100 1111   #add    1, $d1, $g3, $d3       # sub. actual and predicted
0x00f4  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x00f6  01001 1 10 11111110    #li     1, $d2, a(s1t)
0x00f8  01001 0 10 00000000    #li     0, $d2, a(s1t)
0x00fa  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x00fc  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#s1t:

0x00fe  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
```

```
0x0100  01000 0 01 0001 0001   #s      0, $d1, $g0, 1
0x0102  01001 1 11 00001010    #li     1, $d3, 10
0x0104  00000 1 01 1101 1111   #add    1, $d1, $d1, $d3       # sub. actual and predicted
0x0106  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x0108  01001 1 10 00010000    #li     1, $d2, a(s2t)
0x010a  01001 0 10 00000001    #li     0, $d2, a(s2t)
0x010c  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x010e  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#s2t:

0x0110  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x0112  01000 1 01 0001 0001   #s      1, $d1, $g0, 1
0x0114  01001 1 11 00000010    #li     1, $d3, 2
0x0116  00000 1 01 1101 1111   #add    1, $d1, $d1, $d3       # sub. actual and predicted
0x0118  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x011a  01001 1 10 00100010    #li     1, $d2, a(lit)
0x011c  01001 0 10 00000001    #li     0, $d2, a(lit)
0x011e  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x0120  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#lit:

0x0122  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x0124  01001 1 01 00000101    #li     1, $d1, 5
0x0126  01001 0 01 00000000    #li     0, $d1, 0
0x0128  00000 1 01 1101 0001   #add    1, $d1, $d1, $g0       # sub. actual and predicted
0x012a  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x012c  01001 1 10 00110100    #li     1, $d2, a(lwswt)
0x012e  01001 0 10 00000001    #li     0, $d2, a(lwswt)
0x0130  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x0132  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#lwswt:                                # let's just store 0xbeef and read it back out

0x0134  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6       # increment memory location
0x0136  01011 0110 1100 000    #sw     $g5, $d0, 0            # store
0x0138  01010 1101 1100 000    #lw     $d1, $d0, 0
0x013a  00000 1 01 1101 0110   #add    1, $d1, $d1, $g5
0x013c  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x013e  01001 1 10 00110100    #li     1, $d2, a(copyt)
0x0140  01001 1 10 00000001    #li     0, $d2, a(copyt)
0x0142  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14        # test if not zero
0x0144  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#copyt:

0x0146  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x0148  01100 0001 0010 000    #copy   $g0, $g1
0x014a  01001 1 11 00000110    #li     1, $d3, 6
0x014c  00000 1 01 0001 1111   #add    1, $d1, $g0, $d3       # sub. actual and predicted
0x014e  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x0150  01001 1 10 00110100    #li     1, $d2, a(sext)
0x0152  01001 1 10 00000001    #li     0, $d2, a(sext)
0x0154  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x0156  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location

#sext:

0x0158  00000 0 00 1100 0111   #add    0, $d0, $d0, $g6 #increment the memory location by 2
0x015a  01001 1 11 11111110    #li     1, $d3, -2
0x015c  01101 1101 1101 000    #sex    $d3, $d3
0x015e  01001 1 10 11111110    #li     1, $d2, -2
0x0160  01001 0 10 11111111    #li     0, $d2, 255
0x0162  00000 1 01 1110 1111   #add    1, $d1, $d2, $d3       # sub. actual and predicted
0x0164  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x0166  01001 1 10 01110110    #li     1, $d2, a(btj1t)
0x0168  01001 1 10 00000001    #li     0, $d2, a(btj1t)
0x016a  00101 0 10 0100 1110   #btj    0, $d2, $g3, 14               # test if not zero
0x016c  01011 0110 1100 000    #sw     $g5, $d0, 0       # store 0xBEEF in the mem location
0x016e  01001 1 10 00000101    #li     1, $d2, 5          # number to be used in operations
0x0170  01100 0001 1110 000    #copy   $g0, $d2
0x0172  01001 1 11 00000110    #li     1, $d3, 6              # ditto
0x0174  01100 0010 1111 000    #copy   $g1, $d3


#btj1t:                        # jump if positive set

0x0176  00000 0 01 0001 0010   #add    0, $d1, $g0, $g1
0x0178  01010 0100 0000 001    #lw     $g3, $0, 1             # load the condition register
0x017a  01001 1 10 10000110    #li     1, $d2, a(btj2t) # load the address of the next test
```

```
0x017c  01001 0 10 00000001   #li    0, $d2, a(btj2t)
0x017e  00101 1 10 0100 1100  #btj   1, $d2, $g3, 12       # jump to test if positive
                                                           # condition register set
0x0180  01001 1 10 11000100   #li    1, $d2, a(loop)      # load address of the spin loop
0x0182  01001 0 10 00000001   #li    0, $d2, a(loop)
0x0184  00101 0 10 0000 0000  #btj   0, $d2, $0, 0         # jump to the loop if the
                                                           # other jump failed

#btj2t:

0x0186  00000 1 01 0001 0010  #add   1, $d1, $g0, $g1 # subtract to get a negative result
0x0188  01010 0100 0000 001   #lw    $g3, $0, 1            # load the condition register
0x018a  01001 1 10 10010110   #li    1, $d2, a(btj3t)
0x018c  01001 0 10 00000001   #li    0, $d2, a(btj3t)
0x018e  00101 1 10 0100 1101  #btj   1, $d2, $g3, 13       # jump to next test if
                                                           #negative result bit set
0x0190  01001 1 10 11000100   #li    1, $d2, a(loop)
0x0192  01001 0 10 00000001   #li    0, $d2, a(loop)
0x0194  00101 0 10 0000 0000  #btj   0, $d2, $0, 0         # else jump to spin loop

#btj3t:

0x0196  00000 1 01 0001 0001  #add   1, $d1, $g0, $g0     # subtract to get a zero result
0x0198  01010 0100 0000 001   #lw    $g3, $0, 1            # load the condition register
0x019a  01001 1 10 10100110   #li    1, $d2, a(btl1t)
0x019c  01001 0 10 00000001   #li    0, $d2, a(btl1t)
0x019e  00101 1 10 0100 1110  #btj   1, $d2, $g3, 14  # jump to next test if zero bit set
0x01a0  01001 1 10 11000100   #li    1, $d2, a(loop)
0x01a2  01001 0 10 00000001   #li    0, $d2, a(loop)
0x01a4  00101 0 10 0000 0000  #btj   0, $d2, $0, 0         # else jump to spin loop

#btl1t:

0x01a6  00000 0 01 0001 0010  #add   0, $d1, $g0, $g1      # add to get a postive result
0x01a8  01010 0100 0000 001   #lw    $g3, $0, 1            # load the condition register
0x01aa  01001 1 10 11001010   #li    1, $d2, a(jump_target)
0x01ac  01001 0 10 00000001   #li    0, $d2, a(jump_target)
0x01ae  00110 1 10 0100 1100  #btl   1, $d2, $g3, 12       # jump to the jump_target if
                                                           # positive bit set

#btl2t:
10110000
0x01b0  00000 1 10 0001 0010  #add   1, $d1, $g0, $g1 # subtract to get a negative result
0x01b2  01010 0100 0000 001   #lw    $g3, $0, 1            # load the condition register
0x01b4  01001 1 10 11001010   #li    1, $d2, a(jump_target)
0x01b6  01001 0 10 00000001   #li    0, $d2, a(jump_target)
0x01b8  00110 1 10 0100 1101  #btl   1, $d2, $g3, 13       # jump to jump_target if
                                                           # negative bit set

#btl3t:

0x01ba  00000 1 01 0001 0001  #add   1, $d1, $g0, $g0     # subtract to get a zero result
0x01bc  01010 0100 0000 001   #lw    $g3, $0, 1            # load the condition register
0x01be  01001 1 10 11001010   #li    1, $d2, a(jump_target)
0x01c0  01001 0 10 00000001   #li    0, $d2, a(jump_target)
0x01c2  00110 1 10 0100 1110  #btl   1, $d2, $g3, 14 #jump to jump_target if zero bit set

#loop:

0x01c4  01001 1 10 11000100   #li    1, $d2, a(loop)
0x01c6  01001 0 10 00000001   #li    0, $d2, a(loop)
0x01c8  00101 0 10 0000 0000  #btj   0, $d2, $0, 0

#jump_target:

0x01ca  00000 0 00 1100 0111  #add   0, $d0, $d0, $g6     # increment memory address by 2
0x01cc  01011 0110 1100 000   #sw    $g5, $d0, 0           # store 0xbeef in memory
0x01ce  01100 1110 1010 000   #copy $d2, $ra
0x01d0  00101 0 10 0000 0000  # btj   0, $d2, $0, 0         # go back to the test function
```