Contents

Test bench Output

Simple If Decode

```
# read at mem[0x80020000] = 0x27bdffe8
#
# ADDIU $29, $29,    -24
#
# read at mem[0x80020004] = 0xafbe0014
#
# SW $30,     20($29)
#
# read at mem[0x80020008] = 0x03a0f021
#
# ADDU $30, $29, $ 0
#
# read at mem[0x8002000c] = 0x24020003
#
# ADDIU $ 2, $ 0,      3
#
# read at mem[0x80020010] = 0xafc20000
#
# SW $ 2,      0($30)
#
# read at mem[0x80020014] = 0x24020002
#
# ADDIU $ 2, $ 0,      2
#
# read at mem[0x80020018] = 0xafc20004
#
# SW $ 2,      4($30)
#
# read at mem[0x8002001c] = 0xafc00008
#
# SW $ 0,      8($30)
#
# read at mem[0x80020020] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x80020024] = 0x28420005
#
# SLTI $ 2, $ 2,      5
#
# read at mem[0x80020028] = 0x10400007
#
# BEQ $ 2, $ 0, 0x80020048
#
# read at mem[0x8002002c] = 0x00000000
#
# NOP
#
# read at mem[0x80020030] = 0x8fc30000
#
# LW $ 3,      0($30)
#
# read at mem[0x80020034] = 0x8fc20004
```

```
#
# LW $ 2,      4($30)
#
# read at mem[0x80020038] = 0x00621021
#
# ADDU $ 2, $ 3, $ 2
#
# read at mem[0x8002003c] = 0xafc20000
#
# SW $ 2,      0($30)
#
# read at mem[0x80020040] = 0x08008016
#
# J 0x80020058
#
# read at mem[0x80020044] = 0x00000000
#
# NOP
#
# read at mem[0x80020048] = 0x8fc30000
#
# LW $ 3,      0($30)
#
# read at mem[0x8002004c] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x80020050] = 0x00621023
#
# SUBU $ 2, $ 3, $ 2
#
# read at mem[0x80020054] = 0xafc20000
#
# SW $ 2,      0($30)
#
# read at mem[0x80020058] = 0x8fc30000
#
# LW $ 3,      0($30)
#
# read at mem[0x8002005c] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x80020060] = 0x00621021
#
# ADDU $ 2, $ 3, $ 2
#
# read at mem[0x80020064] = 0xafc20008
#
# SW $ 2,      8($30)
#
# read at mem[0x80020068] = 0x8fc20008
#
# LW $ 2,      8($30)
#
# read at mem[0x8002006c] = 0x03c0e821
#
```

```
# ADDU $29, $30, $ 0
#
# read at mem[0x80020070] = 0x8fbe0014
#
# LW $30,     20($29)
#
# read at mem[0x80020074] = 0x27bd0018
#
# ADDIU $29, $29,     24
#
# read at mem[0x80020078] = 0x03e00008
#
# JR $31
#
# read at mem[0x8002007c] = 0x00000000
#
# NOP
```

SimpleAdd Decode

```
# read at mem[0x80020000] = 0x27bdffe8
#
# ADDIU $29, $29,    -24
#
# read at mem[0x80020004] = 0xafbe0014
#
# SW $30,     20($29)
#
# read at mem[0x80020008] = 0x03a0f021
#
# ADDU $30, $29, $ 0
#
# read at mem[0x8002000c] = 0x24020003
#
# ADDIU $ 2, $ 0,      3
#
# read at mem[0x80020010] = 0xafc20000
#
# SW $ 2,      0($30)
#
# read at mem[0x80020014] = 0x24020002
#
# ADDIU $ 2, $ 0,      2
#
# read at mem[0x80020018] = 0xafc20004
#
# SW $ 2,      4($30)
#
# read at mem[0x8002001c] = 0xafc00008
#
# SW $ 0,      8($30)
#
# read at mem[0x80020020] = 0x8fc30000
#
# LW $ 3,      0($30)
#
```

```
# read at mem[0x80020024] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x80020028] = 0x00621021
#
# ADDU $ 2, $ 3, $ 2
#
# read at mem[0x8002002c] = 0xafc20008
#
# SW $ 2,      8($30)
#
# read at mem[0x80020030] = 0x8fc20008
#
# LW $ 2,      8($30)
#
# read at mem[0x80020034] = 0x03c0e821
#
# ADDU $29, $30, $ 0
#
# read at mem[0x80020038] = 0x8fbe0014
#
# LW $30,     20($29)
#
# read at mem[0x8002003c] = 0x27bd0018
#
# ADDIU $29, $29,     24
#
# read at mem[0x80020040] = 0x03e00008
#
# JR $31
#
# read at mem[0x80020044] = 0x00000000
#
# NOP
```

BubbleSort Decode

```
# read at mem[0x80020000] = 0x27bdffc8
#
# ADDIU $29, $29,    -56
#
# read at mem[0x80020004] = 0xafbf0034
#
# SW $31,     52($29)
#
# read at mem[0x80020008] = 0xafbe0030
#
# SW $30,     48($29)
#
# read at mem[0x8002000c] = 0x03a0f021
#
# ADDU $30, $29, $ 0
#
# read at mem[0x80020010] = 0x2402000c
#
```

```
# ADDIU $ 2, $ 0,     12
#
# read at mem[0x80020014] = 0xafc20010
#
# SW $ 2,     16($30)
#
# read at mem[0x80020018] = 0x24020009
#
# ADDIU $ 2, $ 0,      9
#
# read at mem[0x8002001c] = 0xafc20014
#
# SW $ 2,     20($30)
#
# read at mem[0x80020020] = 0x24020004
#
# ADDIU $ 2, $ 0,      4
#
# read at mem[0x80020024] = 0xafc20018
#
# SW $ 2,     24($30)
#
# read at mem[0x80020028] = 0x24020063
#
# ADDIU $ 2, $ 0,     99
#
# read at mem[0x8002002c] = 0xafc2001c
#
# SW $ 2,     28($30)
#
# read at mem[0x80020030] = 0x24020078
#
# ADDIU $ 2, $ 0,    120
#
# read at mem[0x80020034] = 0xafc20020
#
# SW $ 2,     32($30)
#
# read at mem[0x80020038] = 0x24020001
#
# ADDIU $ 2, $ 0,      1
#
# read at mem[0x8002003c] = 0xafc20024
#
# SW $ 2,     36($30)
#
# read at mem[0x80020040] = 0x24020003
#
# ADDIU $ 2, $ 0,      3
#
# read at mem[0x80020044] = 0xafc20028
#
# SW $ 2,     40($30)
#
# read at mem[0x80020048] = 0x2402000a
#
# ADDIU $ 2, $ 0,     10
```

```
#
# read at mem[0x8002004c] = 0xafc2002c
#
# SW $ 2,     44($30)
#
# read at mem[0x80020050] = 0x27c20010
#
# ADDIU $ 2, $30,     16
#
# read at mem[0x80020054] = 0x00402021
#
# ADDU $ 4, $ 2, $ 0
#
# read at mem[0x80020058] = 0x24050008
#
# ADDIU $ 5, $ 0,      8
#
# read at mem[0x8002005c] = 0x0c008020
#
# JAL 0x80020080
#
# read at mem[0x80020060] = 0x00000000
#
# NOP
#
# read at mem[0x80020064] = 0x00001021
#
# ADDU $ 2, $ 0, $ 0
#
# read at mem[0x80020068] = 0x03c0e821
#
# ADDU $29, $30, $ 0
#
# read at mem[0x8002006c] = 0x8fbf0034
#
# LW $31,     52($29)
#
# read at mem[0x80020070] = 0x8fbe0030
#
# LW $30,     48($29)
#
# read at mem[0x80020074] = 0x27bd0038
#
# ADDIU $29, $29,     56
#
# read at mem[0x80020078] = 0x03e00008
#
# JR $31
#
# read at mem[0x8002007c] = 0x00000000
#
# NOP
#
# read at mem[0x80020080] = 0x27bdffe8
#
# ADDIU $29, $29,    -24
#
```

```
# read at mem[0x80020084] = 0xafbe0014
#
# SW $30,     20($29)
#
# read at mem[0x80020088] = 0x03a0f021
#
# ADDU $30, $29, $ 0
#
# read at mem[0x8002008c] = 0xafc40018
#
# SW $ 4,     24($30)
#
# read at mem[0x80020090] = 0xafc5001c
#
# SW $ 5,     28($30)
#
# read at mem[0x80020094] = 0xafc00000
#
# SW $ 0,      0($30)
#
# read at mem[0x80020098] = 0x0800805f
#
# J 0x8002017c
#
# read at mem[0x8002009c] = 0x00000000
#
# NOP
#
# read at mem[0x800200a0] = 0x24020001
#
# ADDIU $ 2, $ 0,      1
#
# read at mem[0x800200a4] = 0xafc20004
#
# SW $ 2,      4($30)
#
# read at mem[0x800200a8] = 0x08008055
#
# J 0x80020154
#
# read at mem[0x800200ac] = 0x00000000
#
# NOP
#
# read at mem[0x800200b0] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x800200b4] = 0x2442ffff
#
# ADDIU $ 2, $ 2,     -1
#
# read at mem[0x800200b8] = 0x00021080
#
# SLL $ 2, $ 2, $ 2
#
# read at mem[0x800200bc] = 0x8fc30018
```

```
#
# LW $ 3,     24($30)
#
# read at mem[0x800200c0] = 0x00621021
#
# ADDU $ 2, $ 3, $ 2
#
# read at mem[0x800200c4] = 0x8c430000
#
# LW $ 3,      0($ 2)
#
# read at mem[0x800200c8] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x800200cc] = 0x00021080
#
# SLL $ 2, $ 2, $ 2
#
# read at mem[0x800200d0] = 0x8fc40018
#
# LW $ 4,     24($30)
#
# read at mem[0x800200d4] = 0x00821021
#
# ADDU $ 2, $ 4, $ 2
#
# read at mem[0x800200d8] = 0x8c420000
#
# LW $ 2,      0($ 2)
#
# read at mem[0x800200dc] = 0x0043102a
#
# SLT $ 2, $ 2, $ 3
#
# read at mem[0x800200e0] = 0x10400019
#
# BEQ $ 2, $ 0, 0x8002014c
#
# read at mem[0x800200e4] = 0x00000000
#
# NOP
#
# read at mem[0x800200e8] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x800200ec] = 0x2442ffff
#
# ADDIU $ 2, $ 2,     -1
#
# read at mem[0x800200f0] = 0x00021080
#
# SLL $ 2, $ 2, $ 2
#
# read at mem[0x800200f4] = 0x8fc30018
#
```

```
# LW $ 3,     24($30)
#
# read at mem[0x800200f8] = 0x00621021
#
# ADDU $ 2, $ 3, $ 2
#
# read at mem[0x800200fc] = 0x8c420000
#
# LW $ 2,      0($ 2)
#
# read at mem[0x80020100] = 0xafc20008
#
# SW $ 2,      8($30)
#
# read at mem[0x80020104] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x80020108] = 0x2442ffff
#
# ADDIU $ 2, $ 2,     -1
#
# read at mem[0x8002010c] = 0x00021080
#
# SLL $ 2, $ 2, $ 2
#
# read at mem[0x80020110] = 0x8fc30018
#
# LW $ 3,     24($30)
#
# read at mem[0x80020114] = 0x00621021
#
# ADDU $ 2, $ 3, $ 2
#
# read at mem[0x80020118] = 0x8fc30004
#
# LW $ 3,      4($30)
#
# read at mem[0x8002011c] = 0x00031880
#
# SLL $ 3, $ 3, $ 2
#
# read at mem[0x80020120] = 0x8fc40018
#
# LW $ 4,     24($30)
#
# read at mem[0x80020124] = 0x00831821
#
# ADDU $ 3, $ 4, $ 3
#
# read at mem[0x80020128] = 0x8c630000
#
# LW $ 3,      0($ 3)
#
# read at mem[0x8002012c] = 0xac430000
#
# SW $ 3,      0($ 2)
```

```
#
# read at mem[0x80020130] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x80020134] = 0x00021080
#
# SLL $ 2, $ 2, $ 2
#
# read at mem[0x80020138] = 0x8fc30018
#
# LW $ 3,     24($30)
#
# read at mem[0x8002013c] = 0x00621021
#
# ADDU $ 2, $ 3, $ 2
#
# read at mem[0x80020140] = 0x8fc30008
#
# LW $ 3,      8($30)
#
# read at mem[0x80020144] = 0xac430000
#
# SW $ 3,      0($ 2)
#
# read at mem[0x80020148] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x8002014c] = 0x24420001
#
# ADDIU $ 2, $ 2,      1
#
# read at mem[0x80020150] = 0xafc20004
#
# SW $ 2,      4($30)
#
# read at mem[0x80020154] = 0x8fc3001c
#
# LW $ 3,     28($30)
#
# read at mem[0x80020158] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x8002015c] = 0x00621823
#
# SUBU $ 3, $ 3, $ 2
#
# read at mem[0x80020160] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x80020164] = 0x0043102a
#
# SLT $ 2, $ 2, $ 3
#
```

```
# read at mem[0x80020168] = 0x1440ffd1
#
# BNE $ 2, $ 0, 0x800600b4
#
# read at mem[0x8002016c] = 0x00000000
#
# NOP
#
# read at mem[0x80020170] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x80020174] = 0x24420001
#
# ADDIU $ 2, $ 2,      1
#
# read at mem[0x80020178] = 0xafc20000
#
# SW $ 2,      0($30)
#
# read at mem[0x8002017c] = 0x8fc30000
#
# LW $ 3,      0($30)
#
# read at mem[0x80020180] = 0x8fc2001c
#
# LW $ 2,     28($30)
#
# read at mem[0x80020184] = 0x0062102a
#
# SLT $ 2, $ 3, $ 2
#
# read at mem[0x80020188] = 0x1440ffc5
#
# BNE $ 2, $ 0, 0x800600a4
#
# read at mem[0x8002018c] = 0x00000000
#
# NOP
#
# read at mem[0x80020190] = 0x03c0e821
#
# ADDU $29, $30, $ 0
#
# read at mem[0x80020194] = 0x8fbe0014
#
# LW $30,     20($29)
#
# read at mem[0x80020198] = 0x27bd0018
#
# ADDIU $29, $29,     24
#
# read at mem[0x8002019c] = 0x03e00008
#
# JR $31
#
# read at mem[0x800201a0] = 0x00000000
```

```
#
# NOP
```

fact Decode

```
# read at mem[0x80020000] = 0x27bdffe8
#
# ADDIU $29, $29,    -24
#
# read at mem[0x80020004] = 0xafbf0014
#
# SW $31,     20($29)
#
# read at mem[0x80020008] = 0xafbe0010
#
# SW $30,     16($29)
#
# read at mem[0x8002000c] = 0x03a0f021
#
# ADDU $30, $29, $ 0
#
# read at mem[0x80020010] = 0xafc40018
#
# SW $ 4,     24($30)
#
# read at mem[0x80020014] = 0x8fc20018
#
# LW $ 2,     24($30)
#
# read at mem[0x80020018] = 0x14400004
#
# BNE $ 2, $ 0, 0x80020030
#
# read at mem[0x8002001c] = 0x00000000
#
# NOP
#
# read at mem[0x80020020] = 0x24020001
#
# ADDIU $ 2, $ 0,      1
#
# read at mem[0x80020024] = 0x08008013
#
# J 0x8002004c
#
# read at mem[0x80020028] = 0x00000000
#
# NOP
#
# read at mem[0x8002002c] = 0x8fc20018
#
# LW $ 2,     24($30)
#
# read at mem[0x80020030] = 0x2442ffff
#
# ADDIU $ 2, $ 2,     -1
#
```

```
# read at mem[0x80020034] = 0x00402021
#
# ADDU $ 4, $ 2, $ 0
#
# read at mem[0x80020038] = 0x0c008000
#
# JAL 0x80020000
#
# read at mem[0x8002003c] = 0x00000000
#
# NOP
#
# read at mem[0x80020040] = 0x00401821
#
# ADDU $ 3, $ 2, $ 0
#
# read at mem[0x80020044] = 0x8fc20018
#
# LW $ 2,     24($30)
#
# read at mem[0x80020048] = 0x70621002
#
# MUL $ 2, $ 3, $ 2
#
# read at mem[0x8002004c] = 0x03c0e821
#
# ADDU $29, $30, $ 0
#
# read at mem[0x80020050] = 0x8fbf0014
#
# LW $31,     20($29)
#
# read at mem[0x80020054] = 0x8fbe0010
#
# LW $30,     16($29)
#
# read at mem[0x80020058] = 0x27bd0018
#
# ADDIU $29, $29,     24
#
# read at mem[0x8002005c] = 0x03e00008
#
# JR $31
#
# read at mem[0x80020060] = 0x00000000
#
# NOP
#
# read at mem[0x80020064] = 0x27bdffe0
#
# ADDIU $29, $29,    -32
#
# read at mem[0x80020068] = 0xafbf001c
#
# SW $31,     28($29)
#
# read at mem[0x8002006c] = 0xafbe0018
```

```
#
# SW $30,     24($29)
#
# read at mem[0x80020070] = 0x03a0f021
#
# ADDU $30, $29, $ 0
#
# read at mem[0x80020074] = 0xafc00010
#
# SW $ 0,     16($30)
#
# read at mem[0x80020078] = 0x08008027
#
# J 0x8002009c
#
# read at mem[0x8002007c] = 0x00000000
#
# NOP
#
# read at mem[0x80020080] = 0x8fc40010
#
# LW $ 4,     16($30)
#
# read at mem[0x80020084] = 0x0c008000
#
# JAL 0x80020000
#
# read at mem[0x80020088] = 0x00000000
#
# NOP
#
# read at mem[0x8002008c] = 0xafc20014
#
# SW $ 2,     20($30)
#
# read at mem[0x80020090] = 0x8fc20010
#
# LW $ 2,     16($30)
#
# read at mem[0x80020094] = 0x24420001
#
# ADDIU $ 2, $ 2,      1
#
# read at mem[0x80020098] = 0xafc20010
#
# SW $ 2,     16($30)
#
# read at mem[0x8002009c] = 0x8fc20010
#
# LW $ 2,     16($30)
#
# read at mem[0x800200a0] = 0x2842000a
#
# SLTI $ 2, $ 2,     10
#
# read at mem[0x800200a4] = 0x1440fff6
#
```

```
# BNE $ 2, $ 0, 0x80060084
#
# read at mem[0x800200a8] = 0x00000000
#
# NOP
#
# read at mem[0x800200ac] = 0x00001021
#
# ADDU $ 2, $ 0, $ 0
#
# read at mem[0x800200b0] = 0x03c0e821
#
# ADDU $29, $30, $ 0
#
# read at mem[0x800200b4] = 0x8fbf001c
#
# LW $31,     28($29)
#
# read at mem[0x800200b8] = 0x8fbe0018
#
# LW $30,     24($29)
#
# read at mem[0x800200bc] = 0x27bd0020
#
# ADDIU $29, $29,     32
#
# read at mem[0x800200c0] = 0x03e00008
#
# JR $31
#
# read at mem[0x800200c4] = 0x00000000
#
# NOP
```

CheckVowel Decode

```
# read at mem[0x80020000] = 0x27bdffd8
#
# ADDIU $29, $29,    -40
#
# read at mem[0x80020004] = 0xafbe0024
#
# SW $30,     36($29)
#
# read at mem[0x80020008] = 0x03a0f021
#
# ADDU $30, $29, $ 0
#
# read at mem[0x8002000c] = 0xafc00000
#
# SW $ 0,      0($30)
#
# read at mem[0x80020010] = 0xafc00004
#
# SW $ 0,      4($30)
#
# read at mem[0x80020014] = 0x3c028002
```

```
#
# LUI $ 2, 0x8002
#
# read at mem[0x80020018] = 0x8c45019c
#
# LW $ 5,    412($ 2)
#
# read at mem[0x8002001c] = 0x2443019c
#
# ADDIU $ 3, $ 2,    412
#
# read at mem[0x80020020] = 0x8c640004
#
# LW $ 4,      4($ 3)
#
# read at mem[0x80020024] = 0x2443019c
#
# ADDIU $ 3, $ 2,    412
#
# read at mem[0x80020028] = 0x8c630008
#
# LW $ 3,      8($ 3)
#
# read at mem[0x8002002c] = 0xafc50008
#
# SW $ 5,      8($30)
#
# read at mem[0x80020030] = 0xafc4000c
#
# SW $ 4,     12($30)
#
# read at mem[0x80020034] = 0xafc30010
#
# SW $ 3,     16($30)
#
# read at mem[0x80020038] = 0x2442019c
#
# ADDIU $ 2, $ 2,    412
#
# read at mem[0x8002003c] = 0x9042000c
#
# LBU $ 2,     12($ 2)
#
# read at mem[0x80020040] = 0xa3c20014
#
# SB $ 2,     20($30)
#
# read at mem[0x80020044] = 0xa3c00015
#
# SB $ 0,     21($30)
#
# read at mem[0x80020048] = 0xa3c00016
#
# SB $ 0,     22($30)
#
# read at mem[0x8002004c] = 0xa3c00017
#
```

```
# SB $ 0,     23($30)
#
# read at mem[0x80020050] = 0xa3c00018
#
# SB $ 0,     24($30)
#
# read at mem[0x80020054] = 0xa3c00019
#
# SB $ 0,     25($30)
#
# read at mem[0x80020058] = 0xa3c0001a
#
# SB $ 0,     26($30)
#
# read at mem[0x8002005c] = 0xa3c0001b
#
# SB $ 0,     27($30)
#
# read at mem[0x80020060] = 0xafc00000
#
# SW $ 0,      0($30)
#
# read at mem[0x80020064] = 0x0800805d
#
# J 0x80020174
#
# read at mem[0x80020068] = 0x00000000
#
# NOP
#
# read at mem[0x8002006c] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x80020070] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x80020074] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x80020078] = 0x24020061
#
# ADDIU $ 2, $ 0,     97
#
# read at mem[0x8002007c] = 0x10620037
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x80020080] = 0x00000000
#
# NOP
#
# read at mem[0x80020084] = 0x8fc20000
#
# LW $ 2,      0($30)
```

```
#
# read at mem[0x80020088] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x8002008c] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x80020090] = 0x24020041
#
# ADDIU $ 2, $ 0,     65
#
# read at mem[0x80020094] = 0x10620031
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x80020098] = 0x00000000
#
# NOP
#
# read at mem[0x8002009c] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x800200a0] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x800200a4] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x800200a8] = 0x24020065
#
# ADDIU $ 2, $ 0,    101
#
# read at mem[0x800200ac] = 0x1062002b
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x800200b0] = 0x00000000
#
# NOP
#
# read at mem[0x800200b4] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x800200b8] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x800200bc] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
```

```
# read at mem[0x800200c0] = 0x24020045
#
# ADDIU $ 2, $ 0,     69
#
# read at mem[0x800200c4] = 0x10620025
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x800200c8] = 0x00000000
#
# NOP
#
# read at mem[0x800200cc] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x800200d0] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x800200d4] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x800200d8] = 0x24020069
#
# ADDIU $ 2, $ 0,    105
#
# read at mem[0x800200dc] = 0x1062001f
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x800200e0] = 0x00000000
#
# NOP
#
# read at mem[0x800200e4] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x800200e8] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x800200ec] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x800200f0] = 0x24020049
#
# ADDIU $ 2, $ 0,     73
#
# read at mem[0x800200f4] = 0x10620019
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x800200f8] = 0x00000000
```

```
#
# NOP
#
# read at mem[0x800200fc] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x80020100] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x80020104] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x80020108] = 0x2402006f
#
# ADDIU $ 2, $ 0,    111
#
# read at mem[0x8002010c] = 0x10620013
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x80020110] = 0x00000000
#
# NOP
#
# read at mem[0x80020114] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x80020118] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x8002011c] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x80020120] = 0x2402004f
#
# ADDIU $ 2, $ 0,     79
#
# read at mem[0x80020124] = 0x1062000d
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x80020128] = 0x00000000
#
# NOP
#
# read at mem[0x8002012c] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x80020130] = 0x03c21021
#
```

```
# ADDU $ 2, $30, $ 2
#
# read at mem[0x80020134] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x80020138] = 0x24020075
#
# ADDIU $ 2, $ 0,    117
#
# read at mem[0x8002013c] = 0x10620007
#
# BEQ $ 3, $ 2, 0x80020160
#
# read at mem[0x80020140] = 0x00000000
#
# NOP
#
# read at mem[0x80020144] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x80020148] = 0x03c21021
#
# ADDU $ 2, $30, $ 2
#
# read at mem[0x8002014c] = 0x90430008
#
# LBU $ 3,      8($ 2)
#
# read at mem[0x80020150] = 0x24020055
#
# ADDIU $ 2, $ 0,     85
#
# read at mem[0x80020154] = 0x14620004
#
# BNE $ 3, $ 2, 0x8002016c
#
# read at mem[0x80020158] = 0x00000000
#
# NOP
#
# read at mem[0x8002015c] = 0x8fc20004
#
# LW $ 2,      4($30)
#
# read at mem[0x80020160] = 0x24420001
#
# ADDIU $ 2, $ 2,      1
#
# read at mem[0x80020164] = 0xafc20004
#
# SW $ 2,      4($30)
#
# read at mem[0x80020168] = 0x8fc20000
#
# LW $ 2,      0($30)
```

```
#
# read at mem[0x8002016c] = 0x24420001
#
# ADDIU $ 2, $ 2,      1
#
# read at mem[0x80020170] = 0xafc20000
#
# SW $ 2,      0($30)
#
# read at mem[0x80020174] = 0x8fc20000
#
# LW $ 2,      0($30)
#
# read at mem[0x80020178] = 0x28420014
#
# SLTI $ 2, $ 2,     20
#
# read at mem[0x8002017c] = 0x1440ffbb
#
# BNE $ 2, $ 0, 0x80060070
#
# read at mem[0x80020180] = 0x00000000
#
# NOP
#
# read at mem[0x80020184] = 0x00001021
#
# ADDU $ 2, $ 0, $ 0
#
# read at mem[0x80020188] = 0x03c0e821
#
# ADDU $29, $30, $ 0
#
# read at mem[0x8002018c] = 0x8fbe0024
#
# LW $30,     36($29)
#
# read at mem[0x80020190] = 0x27bd0028
#
# ADDIU $29, $29,     40
#
# read at mem[0x80020194] = 0x03e00008
#
# JR $31
#
# read at mem[0x80020198] = 0x00000000
#
# NOP
#
# read at mem[0x8002019c] = 0x43686563
#
# ERROR: Opcode not recognized: 01000011011010000110010101100011
#
# read at mem[0x800201a0] = 0x6b566f77
#
# ERROR: Opcode not recognized: 01101011010101100110111101110111
#
```

```
# read at mem[0x800201a4] = 0x656c210a
#
# ERROR: Opcode not recognized: 01100101011011000010000100001010
#
# read at mem[0x800201a8] = 0x00000000
#
# NOP
```

Code

ECE429_Fetch.v: Fetch Module

```
module ECE429_Fetch(clk_in, pc_out, pc_decode_out, rw_out, stall_in,
access_size_out);

input clk_in;
input stall_in;

output [0:31] pc_out;
output [0:31] pc_decode_out;
output rw_out;    // 0 to read, 1 to write
output [0:1] access_size_out; // 11 for word, 10 for half-word, 01/00 byte

reg [0:31] program_counter;

assign pc_out = program_counter;
assign pc_decode_out = program_counter;
assign access_size_out = 2'b11;
assign rw_out = 1'b0;

initial begin
      program_counter = 32'h80020000;
end

always @ (negedge clk_in)
begin
      if (!stall_in) begin
            program_counter = program_counter + 4;
      end
end
endmodule
```

ECE429_Decode.v: Decoder Module

```
module ECE429_Decode( insn_in, pc_in );

input[0:31] insn_in;
input[0:31] pc_in;



/* R-Type
 *    A     B     C     D     E     F
 * |-----------------------------------|
 * |  6  |  5  |  5  |  5  |  5  |  6  |
 * |-----------------------------------|
```

```
 *
 * J-Type
 *    A                G
 * |-----------------------------------|
 * |  6  |             26              |
 * |-----------------------------------|
 *
 * I-Type
 *    A     B     C          H
 * |-----------------------------------|
 * |  6  |  5  |  5  |       16        |
 * |-----------------------------------|
 *
 *
 *          Letter            Reg Name
 *            A               op_code
 *            B               reg_RS
 *            C               reg_RT
 *            D               reg_RD
 *            E               reg_SHAMT
 *            F               reg_FUNCT
 *            G               jump_ADDR
 *            H               immediate_value
 *
 */
reg[0:5] op_code;
reg[0:4] reg_RS;
reg[0:4] reg_RT;
reg[0:4] reg_RD;
reg[0:4] reg_SHAMT;
reg[0:5] reg_FUNCT;
reg[0:25] jump_ADDR;
reg signed[0:15] immediate_value;

reg[0:31] nextPC;

/*****************************************************************
 *                  EXTRACT DATA FROM INSTRUCTION
 *****************************************************************/
/*
always @(insn_in) begin
      op_code = insn_in[0:5];
end

always @(insn_in) begin
      reg_RS = insn_in[6:10];
end

always @(insn_in) begin
      reg_RT = insn_in[11:15];
end

always @(insn_in) begin
      reg_RD = insn_in[16:20];
end

always @(insn_in) begin
```

```
      reg_SHAMT = insn_in[21:25];
end

always @(insn_in) begin
      reg_FUNCT = insn_in[26:31];
end

always @(insn_in) begin
      jump_ADDR = insn_in[6:31];
end

always @(insn_in) begin
      immediate_value = insn_in[16:31];
end


// Calculate the next P (for jumps)
always @(pc_in) begin
      nextPC = pc_in + 4;
end
*/


/*****************************************************************
 *                 PARSE INSTRUCTION OPCODE
 *----------------------------------------------------------------
 * Update whenever the instruction changes
 *****************************************************************/
always @( insn_in or pc_in )
begin

      op_code = insn_in[0:5];
      reg_RS = insn_in[6:10];
      reg_RT = insn_in[11:15];
      reg_RD = insn_in[16:20];
      reg_SHAMT = insn_in[21:25];
      reg_FUNCT = insn_in[26:31];
      jump_ADDR = insn_in[6:31];
      immediate_value = insn_in[16:31];
      nextPC = pc_in + 4;


      // Check whether a NOP or a valid instruction
      if( insn_in == 32'h00000000 ) begin
            $display("NOP\n");
      end else begin

            case( op_code )

                  // SPECIAL opcode
                  6'b000000:
                  begin
                        case( reg_FUNCT )

                              6'b000000:                    // SLL
                              begin
```

```
                                    $display("SLL $%d, $%d, $%d\n", reg_RD,
reg_RT, reg_SHAMT);
                              end
                              6'b000010:                    // SRL
                              begin
                                    $display("SRL $%d, $%d, $%d\n", reg_RD,
reg_RT, reg_SHAMT);
                              end
                              6'b000011:                    // SRA
                              begin
                                    $display("SRA $%d, $%d, $%d\n", reg_RD,
reg_RT, reg_SHAMT);
                              end
                              6'b001000:                    // JR
                              begin
                                    $display("JR $%d\n", reg_RS);
      // TODO: What do with other values in insn?  What is 10-bit zeroes and
"hint"?
                              end
                              6'b100000:                    // ADD
                              begin
                                    $display("ADD $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b100001:                    // ADDU
                              begin
                                    $display("ADDU $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b100010:                    // SUB
                              begin
                                    $display("SUB $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b100011:                    // SUBU
                              begin
                                    $display("SUBU $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b100100:                    // AND
                              begin
                                    $display("AND $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b100101:                    // OR
                              begin
                                    $display("OR $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b100110:                    // XOR
                              begin
                                    $display("XOR $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b100111:                    // NOR
                              begin
```

```
                                    $display("NOR $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b101010:                    // SLT
                              begin
                                    $display("SLT $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              6'b101011:                    // SLTU
                              begin
                                    $display("SLTU $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                              end
                              default:                      // Error
                              begin
                                    $display("Error decoding SPECIAL opcode:
%b\n", reg_FUNCT);
                              end

                        endcase
                  end

                  // REGIMM opcode
                  6'b000001:
                  begin
                        case( reg_RT )

                              5'b00000:                     // BLTZ
                              begin
                                    $display("BLTZ $%d, %d\n", reg_RS,
{immediate_value, 2'b00});
                              end
                              5'b00001:                     // BGEZ
                              begin
                                    $display("BGEZ $%d, %d\n", reg_RS,
{immediate_value, 2'b00});
                              end
                              default:                      // Error
                              begin
                                    $display("Error decoding REGIMM
opcode\n");
                              end

                        endcase
                  end

                  // J
                  6'b000010:
                  begin
                        $display("J 0x%h\n", {nextPC[0:3], jump_ADDR,
2'b00});          // TODO: Calculate jump address
                  end

                  // JAL
                  6'b000011:
                  begin
```

```
                        $display("JAL 0x%h\n", {nextPC[0:3], jump_ADDR,
2'b00});          // TODO: Calculate jump address
                  end

                  // BEQ
                  6'b000100:
                  begin
                        $display("BEQ $%d, $%d, 0x%h\n", reg_RS, reg_RT,
nextPC + $signed({immediate_value, 2'b00}));
                  end

                  // BNE
                  6'b000101:
                  begin
                        $display("BNE $%d, $%d, 0x%h\n", reg_RS, reg_RT,
nextPC + $signed({immediate_value, 2'b00}));
                  end

                  // BLEZ
                  6'b000110:
                  begin
                        $display("BLEZ $%d, 0x%h\n", reg_RS, nextPC +
$signed({immediate_value, 2'b00}));
                  end

                  // BGTZ
                  6'b000110:
                  begin
                        $display("BGTZ $%d, 0x%h\n", reg_RS, nextPC +
$signed({immediate_value, 2'b00}));
                  end

                  // ADDIU
                  6'b001001:
                  begin
                        $display("ADDIU $%d, $%d, %d\n", reg_RT, reg_RS,
immediate_value);
                  end

                  // SLTI
                  6'b001010:
                  begin
                        $display("SLTI $%d, $%d, %d\n", reg_RT, reg_RS,
immediate_value);
                  end

                  // ORI
                  6'b001101:
                  begin
                        $display("ORI $%d, $%d, 0x%h\n", reg_RT, reg_RS,
immediate_value);
                  end

                  // LUI
                  6'b001111:
                  begin
                        $display("LUI $%d, 0x%h\n", reg_RT, immediate_value);
```

```
                end

                // SPECIAL2
                6'b011100:
                begin
                      case( reg_FUNCT )
                            6'b000010:                    // MUL
                            begin
                                  $display("MUL $%d, $%d, $%d\n", reg_RD,
reg_RS, reg_RT);
                            end
                            default:                      // Error
                            begin
                                  $display("Error decoding SPECIAL2
opcode\n");
                            end
                      endcase
                end

                // LB
                6'b100000:
                begin
                      $display("LB $%d, %d($%d)\n", reg_RT,
immediate_value, reg_RS);
                end

                // LW
                6'b100011:
                begin
                      $display("LW $%d, %d($%d)\n", reg_RT,
immediate_value, reg_RS);
                end

                // LBU
                6'b100100:
                begin
                      $display("LBU $%d, %d($%d)\n", reg_RT,
immediate_value, reg_RS);
                end

                // SB
                6'b101000:
                begin
                      $display("SB $%d, %d($%d)\n", reg_RT,
immediate_value, reg_RS);
                end

                // SW
                6'b101011:
                begin
                      $display("SW $%d, %d($%d)\n", reg_RT,
immediate_value, reg_RS);
                end


                // Error: Could not parse instruction
                default :
```

```
                  begin
                        $display("ERROR: Opcode not recognized: %b\n",
insn_in);
                  end

            endcase

      end

end



endmodule
```

ECE429_Decode_tb.v: Decoder Test Bench

```
module ECE429_Decode_tb();

reg clock;
reg parseEnable;
wire[0:31] parseAddr;         // A 32-bit address to put into the memory
wire[0:31] memAddr;
wire[0:31] memData;                  // A 32-bit piece of data to write to
memory
wire[0:1] parseAccessSize;           // The access size to write to the memory
using
wire[0:1] memAccessSize;
wire memR_W;                               // Whether to read or write to
memory
wire parseDone;                            // Set to 1 once parser is
parseDone
wire parseError;                           // Set to 1 on parseError

wire fetchR_W;
wire[0:31] fetchAddr;
wire[0:31] decAddr;
wire[0:1] fetchAccessSize;

reg[0:31] maxfetchAddr;

reg[0:31] tmp_insn_in;

wire fetStall;

wire [0:31] dataout;          // To see output of the memory

initial begin
      clock = 0;
      maxfetchAddr = 32'h00000000;
      tmp_insn_in = 32'h00000000;
      // Toggle parse enable
      parseEnable = 0;
      #1
      parseEnable = 1;
      #1
      parseEnable = 0;
```

```
      @(posedge clock);

end

assign memR_W = (parseDone) ? fetchR_W : ~parseDone;
assign memAddr = (parseDone) ? fetchAddr : parseAddr;
assign memAccessSize = (parseDone) ? fetchAccessSize : parseAccessSize;
assign decAddr = fetchAddr;
assign fetStall = ~parseDone;
assign insn_in = (fetStall) ? 32'h00000000 : tmp_insn_in;

always begin
  #10 clock = ~clock;
end


always @(parseAddr) begin
      if(parseAddr > maxfetchAddr) begin
            maxfetchAddr = parseAddr;
      end
end


always @(parseError) begin
  if(parseError == 1) begin
    $finish;
  end
end

always @(negedge clock) begin
      if (parseDone == 1) begin
            tmp_insn_in = dataout;

            if(fetchAddr >= maxfetchAddr) begin
                  $finish;
            end

            $display("read at mem[0x%h] = 0x%h\n", fetchAddr, dataout);
      end
end

ECE429_Fetch f(
      .clk_in(clock),
      .pc_out(fetchAddr),
      .pc_decode_out(decAddr),
      .rw_out(fetchR_W),
      .stall_in(fetStall),
      .access_size_out(fetchAccessSize)
);

ECE429_SRECParser #("fact.srec") s(
      .clock(clock),
      .parseEnable(parseEnable),
      .parseAddr(parseAddr),
      .memData(memData),
      .parseAccessSize(parseAccessSize),
```

```
      .parseDone(parseDone),
      .parseError(parseError)
);

ECE429_Memory m(
  .clock(clock),
  .address(memAddr),
  .datain(memData),
  .access_size(memAccessSize),
  .r_w(memR_W),
  .dataout(dataout)
);

ECE429_Decode d(
  .insn_in(tmp_insn_in),
  .pc_in(decAddr)
);

endmodule
```