University Of Waterloo

ECE 429

Task 2
Fetching and Decoding Instructions

Stephan Van Den Heuval	sjvanden
Ravil Baizhiyenov	rbaizhiy
David Janssen	dajjanss
Ben Ridder	brridder

May 28, 2012

1 Output

Listing 1: Bubble Sort Output

```
#
# PC: 80020000, Instruction: 00100111101111011111111111001000
# I-Type instruction.
# ADDIU rs: 29 rt: 29 immediate: 65480
#
# PC: 80020004, Instruction: 10101111101111110000000000110100
# I-Type instruction.
# SW base: 29 rt: 31 offset: 52
#
# PC: 80020008, Instruction: 10101111101111100000000000110000
# I-Type instruction.
# SW base: 29 rt: 30 offset: 48
#
# PC: 8002000c, Instruction: 00000011101000001111000000100001
# R-Type instruction.
# ADDU rs: 29 rt: 0 rd: 30
#
# PC: 80020010, Instruction: 00100100000000100000000000001100
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 12
#
# PC: 80020014, Instruction: 10101111110000100000000000010000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 16
#
# PC: 80020018, Instruction: 00100100000000100000000000001001
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 9
#
# PC: 8002001c, Instruction: 10101111110000100000000000010100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 20
#
# PC: 80020020, Instruction: 00100100000000100000000000000100
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 4
#
# PC: 80020024, Instruction: 10101111110000100000000000011000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 24
#
# PC: 80020028, Instruction: 00100100000000100000000001100011
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 99
#
# PC: 8002002c, Instruction: 10101111110000100000000000011100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 28
```

```
#
# PC: 80020030, Instruction: 00100100000000100000000001111000
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 120
#
# PC: 80020034, Instruction: 10101111110000100000000000100000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 32
#
# PC: 80020038, Instruction: 00100100000000100000000000000001
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 1
#
# PC: 8002003c, Instruction: 10101111110000100000000000100100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 36
#
# PC: 80020040, Instruction: 00100100000000100000000000000011
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 3
#
# PC: 80020044, Instruction: 10101111110000100000000000101000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 40
#
# PC: 80020048, Instruction: 00100100000000100000000000001010
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 10
#
# PC: 8002004c, Instruction: 10101111110000100000000000101100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 44
#
# PC: 80020050, Instruction: 00100111110000100000000000010000
# I-Type instruction.
# ADDIU rs: 30 rt: 2 immediate: 16
#
# PC: 80020054, Instruction: 00000000010000000010000000100001
# R-Type instruction.
# ADDU rs: 2 rt: 0 rd: 4
#
# PC: 80020058, Instruction: 00100100000001010000000000001000
# I-Type instruction.
# ADDIU rs: 0 rt: 5 immediate: 8
#
# PC: 8002005c, Instruction: 00001100000000001000000000100000
# unimplemented/incorrect intruction
#
# PC: 80020060, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
```

```
#
# PC: 80020064, Instruction: 00000000000000000001000000100001
# R-Type instruction.
# ADDU rs: 0 rt: 0 rd: 2
#
# PC: 80020068, Instruction: 00000011110000001110100000100001
# R-Type instruction.
# ADDU rs: 30 rt: 0 rd: 29
#
# PC: 8002006c, Instruction: 10001111101111110000000000110100
# I-Type instruction.
# LW base: 29 rt: 31 offset: 52
#
# PC: 80020070, Instruction: 10001111101111100000000000110000
# I-Type instruction.
# LW base: 29 rt: 30 offset: 48
#
# PC: 80020074, Instruction: 00100111101111010000000000111000
# I-Type instruction.
# ADDIU rs: 29 rt: 29 immediate: 56
#
# PC: 80020078, Instruction: 00000011111000000000000000001000
# R-Type instruction.
# unimplemented ADD type intruction
#
# PC: 8002007c, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 80020080, Instruction: 00100111101111011111111111101000
# I-Type instruction.
# ADDIU rs: 29 rt: 29 immediate: 65512
#
# PC: 80020084, Instruction: 10101111101111100000000000010100
# I-Type instruction.
# SW base: 29 rt: 30 offset: 20
#
# PC: 80020088, Instruction: 00000011101000001111000000100001
# R-Type instruction.
# ADDU rs: 29 rt: 0 rd: 30
#
# PC: 8002008c, Instruction: 10101111110001000000000000011000
# I-Type instruction.
# SW base: 30 rt: 4 offset: 24
#
# PC: 80020090, Instruction: 10101111110001010000000000011100
# I-Type instruction.
# SW base: 30 rt: 5 offset: 28
#
# PC: 80020094, Instruction: 10101111110000000000000000000000
# I-Type instruction.
```

```
# SW base: 30 rt: 0 offset: 0
#
# PC: 80020098, Instruction: 00001000000000001000000001011111
# J-Type instruction.
# J instr_index: 32863
#
# PC: 8002009c, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 800200a0, Instruction: 00100100000000100000000000000001
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 1
#
# PC: 800200a4, Instruction: 10101111110000100000000000000100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 4
#
# PC: 800200a8, Instruction: 00001000000000001000000001010101
# J-Type instruction.
# J instr_index: 32853
#
# PC: 800200ac, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 800200b0, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 800200b4, Instruction: 00100100010000101111111111111111
# I-Type instruction.
# ADDIU rs: 2 rt: 2 immediate: 65535
#
# PC: 800200b8, Instruction: 00000000000000100001000010000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 2 rt: 2 rd: 2
#
# PC: 800200bc, Instruction: 10001111110000110000000000011000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 24
#
# PC: 800200c0, Instruction: 00000000011000100001000000100001
# R-Type instruction.
# ADDU rs: 3 rt: 2 rd: 2
#
# PC: 800200c4, Instruction: 10001100010000110000000000000000
# I-Type instruction.
# LW base: 2 rt: 3 offset: 0
#
# PC: 800200c8, Instruction: 10001111110000100000000000000100
```

```
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 800200cc, Instruction: 00000000000000100001000010000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 2 rt: 2 rd: 2
#
# PC: 800200d0, Instruction: 10001111110001000000000000011000
# I-Type instruction.
# LW base: 30 rt: 4 offset: 24
#
# PC: 800200d4, Instruction: 00000000100000100001000000100001
# R-Type instruction.
# ADDU rs: 4 rt: 2 rd: 2
#
# PC: 800200d8, Instruction: 10001100010000100000000000000000
# I-Type instruction.
# LW base: 2 rt: 2 offset: 0
#
# PC: 800200dc, Instruction: 00000000010000110001000000101010
# R-Type instruction.
# SLT rs: 2 rt: 3 rd: 2
#
# PC: 800200e0, Instruction: 00010000010000000000000000011001
# J-Type instruction.
# BEQ rs: 2 rt: 0 offset: 25
#
# PC: 800200e4, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 800200e8, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 800200ec, Instruction: 00100100010000101111111111111111
# I-Type instruction.
# ADDIU rs: 2 rt: 2 immediate: 65535
#
# PC: 800200f0, Instruction: 00000000000000100001000010000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 2 rt: 2 rd: 2
#
# PC: 800200f4, Instruction: 10001111110000110000000000011000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 24
#
# PC: 800200f8, Instruction: 00000000011000100001000000100001
# R-Type instruction.
# ADDU rs: 3 rt: 2 rd: 2
#
```

```
# PC: 800200fc, Instruction: 10001100010000100000000000000000
# I-Type instruction.
# LW base: 2 rt: 2 offset: 0
#
# PC: 80020100, Instruction: 10101111110000100000000000001000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 8
#
# PC: 80020104, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 80020108, Instruction: 00100100010000101111111111111111
# I-Type instruction.
# ADDIU rs: 2 rt: 2 immediate: 65535
#
# PC: 8002010c, Instruction: 00000000000000100001000010000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 2 rt: 2 rd: 2
#
# PC: 80020110, Instruction: 10001111110000110000000000011000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 24
#
# PC: 80020114, Instruction: 00000000011000100001000000100001
# R-Type instruction.
# ADDU rs: 3 rt: 2 rd: 2
#
# PC: 80020118, Instruction: 10001111110000110000000000000100
# I-Type instruction.
# LW base: 30 rt: 3 offset: 4
#
# PC: 8002011c, Instruction: 00000000000000110001100010000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 2 rt: 3 rd: 3
#
# PC: 80020120, Instruction: 10001111110001000000000000011000
# I-Type instruction.
# LW base: 30 rt: 4 offset: 24
#
# PC: 80020124, Instruction: 00000000100000110001100000100001
# R-Type instruction.
# ADDU rs: 4 rt: 3 rd: 3
#
# PC: 80020128, Instruction: 10001100011000110000000000000000
# I-Type instruction.
# LW base: 3 rt: 3 offset: 0
#
# PC: 8002012c, Instruction: 10101100010000110000000000000000
# I-Type instruction.
# SW base: 2 rt: 3 offset: 0
```

```
#
# PC: 80020130, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 80020134, Instruction: 00000000000000100001000010000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 2 rt: 2 rd: 2
#
# PC: 80020138, Instruction: 10001111110000110000000000011000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 24
#
# PC: 8002013c, Instruction: 00000000011000100001000000100001
# R-Type instruction.
# ADDU rs: 3 rt: 2 rd: 2
#
# PC: 80020140, Instruction: 10001111110000110000000000001000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 8
#
# PC: 80020144, Instruction: 10101100010000110000000000000000
# I-Type instruction.
# SW base: 2 rt: 3 offset: 0
#
# PC: 80020148, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 8002014c, Instruction: 00100100010000100000000000000001
# I-Type instruction.
# ADDIU rs: 2 rt: 2 immediate: 1
#
# PC: 80020150, Instruction: 10101111110000100000000000000100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 4
#
# PC: 80020154, Instruction: 10001111110000110000000000011100
# I-Type instruction.
# LW base: 30 rt: 3 offset: 28
#
# PC: 80020158, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 8002015c, Instruction: 00000000011000100001100000100011
# R-Type instruction.
# SUBU rs: 3 rt: 2 rd: 3
#
# PC: 80020160, Instruction: 10001111110000100000000000000100
# I-Type instruction.
```

```
# LW base: 30 rt: 2 offset: 4
#
# PC: 80020164, Instruction: 00000000010000110001000000101010
# R-Type instruction.
# SLT rs: 2 rt: 3 rd: 2
#
# PC: 80020168, Instruction: 00010100010000001111111111010001
# J-Type instruction.
# BNE rs: 2 rt: 0 offset: 65489
#
# PC: 8002016c, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 80020170, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 80020174, Instruction: 00100100010000100000000000000001
# I-Type instruction.
# ADDIU rs: 2 rt: 2 immediate: 1
#
# PC: 80020178, Instruction: 10101111110000100000000000000000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 0
#
# PC: 8002017c, Instruction: 10001111110000110000000000000000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 0
#
# PC: 80020180, Instruction: 10001111110000100000000000011100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 28
#
# PC: 80020184, Instruction: 00000000011000100001000000101010
# R-Type instruction.
# SLT rs: 3 rt: 2 rd: 2
#
# PC: 80020188, Instruction: 00010100010000001111111111000101
# J-Type instruction.
# BNE rs: 2 rt: 0 offset: 65477
#
# PC: 8002018c, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 80020190, Instruction: 00000011110000001110100000100001
# R-Type instruction.
# ADDU rs: 30 rt: 0 rd: 29
#
# PC: 80020194, Instruction: 10001111101111100000000000010100
```

```
# I-Type instruction.
# LW base: 29 rt: 30 offset: 20
#
# PC: 80020198, Instruction: 00100111101111010000000000011000
# ** Note: $finish : fetch_tb.v(88)
# Time: 2380 ns Iteration: 0 Instance: /fetch_tb
```

Listing 2: Simple Add Output

```
#
# PC: 80020000, Instruction: 00100111101111011111111111101000
# I-Type instruction.
# ADDIU rs: 29 rt: 29 immediate: 65512
#
# PC: 80020004, Instruction: 10101111101111100000000000010100
# I-Type instruction.
# SW base: 29 rt: 30 offset: 20
#
# PC: 80020008, Instruction: 00000011101000001111000000100001
# R-Type instruction.
# ADDU rs: 29 rt: 0 rd: 30
#
# PC: 8002000c, Instruction: 00100100000000100000000000000011
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 3
#
# PC: 80020010, Instruction: 10101111110000100000000000000000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 0
#
# PC: 80020014, Instruction: 00100100000000100000000000000010
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 2
#
# PC: 80020018, Instruction: 10101111110000100000000000000100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 4
#
# PC: 8002001c, Instruction: 10101111110000000000000000001000
# I-Type instruction.
# SW base: 30 rt: 0 offset: 8
#
# PC: 80020020, Instruction: 10001111110000110000000000000000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 0
#
# PC: 80020024, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 80020028, Instruction: 00000000011000100001000000100001
# R-Type instruction.
```

```
# ADDU rs: 3 rt: 2 rd: 2
#
# PC: 8002002c, Instruction: 10101111110000100000000000001000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 8
#
# PC: 80020030, Instruction: 10001111110000100000000000001000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 8
#
# PC: 80020034, Instruction: 00000011110000001110100000100001
# R-Type instruction.
# ADDU rs: 30 rt: 0 rd: 29
#
# PC: 80020038, Instruction: 10001111101111100000000000010100
# I-Type instruction.
# LW base: 29 rt: 30 offset: 20
#
# PC: 8002003c, Instruction: 00100111101111010000000000011000
# ** Note: $finish : fetch_tb.v(88)
# Time: 420 ns Iteration: 0 Instance: /fetch_tb
```

Listing 3: Simple If Output

```
#
# PC: 80020000, Instruction: 00100111101111011111111111101000
# I-Type instruction.
# ADDIU rs: 29 rt: 29 immediate: 65512
#
# PC: 80020004, Instruction: 10101111101111100000000000010100
# I-Type instruction.
# SW base: 29 rt: 30 offset: 20
#
# PC: 80020008, Instruction: 00000011101000001111000000100001
# R-Type instruction.
# ADDU rs: 29 rt: 0 rd: 30
#
# PC: 8002000c, Instruction: 00100100000000100000000000000011
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 3
#
# PC: 80020010, Instruction: 10101111110000100000000000000000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 0
#
# PC: 80020014, Instruction: 00100100000000100000000000000010
# I-Type instruction.
# ADDIU rs: 0 rt: 2 immediate: 2
#
# PC: 80020018, Instruction: 10101111110000100000000000000100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 4
```

```
#
# PC: 8002001c, Instruction: 10101111110000000000000000001000
# I-Type instruction.
# SW base: 30 rt: 0 offset: 8
#
# PC: 80020020, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 80020024, Instruction: 00101000010000100000000000000101
# I-Type instruction.
# SLTI rs: 2 rt: 2 immediate: 5
#
# PC: 80020028, Instruction: 00010000010000000000000000000111
# J-Type instruction.
# BEQ rs: 2 rt: 0 offset: 7
#
# PC: 8002002c, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 80020030, Instruction: 10001111110000110000000000000000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 0
#
# PC: 80020034, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 80020038, Instruction: 00000000011000100001000000100001
# R-Type instruction.
# ADDU rs: 3 rt: 2 rd: 2
#
# PC: 8002003c, Instruction: 10101111110000100000000000000000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 0
#
# PC: 80020040, Instruction: 00001000000000001000000000010110
# J-Type instruction.
# J instr_index: 32790
#
# PC: 80020044, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 80020048, Instruction: 10001111110000110000000000000000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 0
#
# PC: 8002004c, Instruction: 10001111110000100000000000000100
# I-Type instruction.
```

```
# LW base: 30 rt: 2 offset: 4
#
# PC: 80020050, Instruction: 00000000011000100001000000100011
# R-Type instruction.
# SUBU rs: 3 rt: 2 rd: 2
#
# PC: 80020054, Instruction: 10101111110000100000000000000000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 0
#
# PC: 80020058, Instruction: 10001111110000110000000000000000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 0
#
# PC: 8002005c, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 80020060, Instruction: 00000000011000100001000000100001
# R-Type instruction.
# ADDU rs: 3 rt: 2 rd: 2
#
# PC: 80020064, Instruction: 10101111110000100000000000001000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 8
#
# PC: 80020068, Instruction: 10001111110000100000000000001000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 8
#
# PC: 8002006c, Instruction: 00000011110000001110100000100001
# R-Type instruction.
# ADDU rs: 30 rt: 0 rd: 29
#
# PC: 80020070, Instruction: 10001111101111100000000000010100
# I-Type instruction.
# LW base: 29 rt: 30 offset: 20
#
# PC: 80020074, Instruction: 00100111101111010000000000011000
# ** Note: $finish : fetch_tb.v(88)
# Time: 730 ns Iteration: 0 Instance: /fetch_tb
```

Listing 4: Sum Array Output

```
# PC: 80020000, Instruction: 00100111101111011111111111001000
# I-Type instruction.
# ADDIU rs: 29 rt: 29 immediate: 65480
#
# PC: 80020004, Instruction: 10101111101111100000000000110100
# I-Type instruction.
# SW base: 29 rt: 30 offset: 52
#
```

```
# PC: 80020008, Instruction: 00000011101000001111000000100001
# R-Type instruction.
# ADDU rs: 29 rt: 0 rd: 30
#
# PC: 8002000c, Instruction: 10101111110000000000000000000000
# I-Type instruction.
# SW base: 30 rt: 0 offset: 0
#
# PC: 80020010, Instruction: 10101111110000000000000000000100
# I-Type instruction.
# SW base: 30 rt: 0 offset: 4
#
# PC: 80020014, Instruction: 10101111110000000000000000000000
# I-Type instruction.
# SW base: 30 rt: 0 offset: 0
#
# PC: 80020018, Instruction: 00001000000000001000000000010000
# J-Type instruction.
# J instr_index: 32784
#
# PC: 8002001c, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 80020020, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 80020024, Instruction: 00000000000000100001000010000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 2 rt: 2 rd: 2
#
# PC: 80020028, Instruction: 00000011110000100001000000100001
# R-Type instruction.
# ADDU rs: 30 rt: 2 rd: 2
#
# PC: 8002002c, Instruction: 10001111110000110000000000000000
# I-Type instruction.
# LW base: 30 rt: 3 offset: 0
#
# PC: 80020030, Instruction: 10101100010000110000000000001000
# I-Type instruction.
# SW base: 2 rt: 3 offset: 8
#
# PC: 80020034, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 80020038, Instruction: 00100100010000100000000000000001
# I-Type instruction.
# ADDIU rs: 2 rt: 2 immediate: 1
```

```
#
# PC: 8002003c, Instruction: 10101111110000100000000000000000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 0
#
# PC: 80020040, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 80020044, Instruction: 00101000010000100000000000001010
# I-Type instruction.
# SLTI rs: 2 rt: 2 immediate: 10
#
# PC: 80020048, Instruction: 00010100010000001111111111110101
# J-Type instruction.
# BNE rs: 2 rt: 0 offset: 65525
#
# PC: 8002004c, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 80020050, Instruction: 10101111110000000000000000000000
# I-Type instruction.
# SW base: 30 rt: 0 offset: 0
#
# PC: 80020054, Instruction: 00001000000000001000000000100001
# J-Type instruction.
# J instr_index: 32801
#
# PC: 80020058, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 8002005c, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 80020060, Instruction: 00000000000000100001000010000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 2 rt: 2 rd: 2
#
# PC: 80020064, Instruction: 00000011110000100001000000100001
# R-Type instruction.
# ADDU rs: 30 rt: 2 rd: 2
#
# PC: 80020068, Instruction: 10001100010000100000000000001000
# I-Type instruction.
# LW base: 2 rt: 2 offset: 8
#
# PC: 8002006c, Instruction: 10001111110000110000000000000100
# I-Type instruction.
```

```
# LW base: 30 rt: 3 offset: 4
#
# PC: 80020070, Instruction: 00000000011000100001000000100001
# R-Type instruction.
# ADDU rs: 3 rt: 2 rd: 2
#
# PC: 80020074, Instruction: 10101111110000100000000000000100
# I-Type instruction.
# SW base: 30 rt: 2 offset: 4
#
# PC: 80020078, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 8002007c, Instruction: 00100100010000100000000000000001
# I-Type instruction.
# ADDIU rs: 2 rt: 2 immediate: 1
#
# PC: 80020080, Instruction: 10101111110000100000000000000000
# I-Type instruction.
# SW base: 30 rt: 2 offset: 0
#
# PC: 80020084, Instruction: 10001111110000100000000000000000
# I-Type instruction.
# LW base: 30 rt: 2 offset: 0
#
# PC: 80020088, Instruction: 00101000010000100000000000001010
# I-Type instruction.
# SLTI rs: 2 rt: 2 immediate: 10
#
# PC: 8002008c, Instruction: 00010100010000001111111111110011
# J-Type instruction.
# BNE rs: 2 rt: 0 offset: 65523
#
# PC: 80020090, Instruction: 00000000000000000000000000000000
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 0 rt: 0 rd: 0
#
# PC: 80020094, Instruction: 10001111110000100000000000000100
# I-Type instruction.
# LW base: 30 rt: 2 offset: 4
#
# PC: 80020098, Instruction: 00000011110000001110100000100001
# R-Type instruction.
# ADDU rs: 30 rt: 0 rd: 29
#
# PC: 8002009c, Instruction: 10001111101111100000000000110100
# I-Type instruction.
# LW base: 29 rt: 30 offset: 52
#
# PC: 800200a0, Instruction: 00100111101111010000000000111000
```

```
# ** Note: $finish : fetch_tb.v(88)
# Time: 980 ns Iteration: 0 Instance: /fetch_tb
```

Listing 5: Decode Testbench Output

```
# R-Type instruction.
# ADD rs: 5 rt: 24 rd: 6
#
# R-Type instruction.
# ADDU rs: 1 rt: 3 rd: 2
#
# R-Type instruction.
# SUB rs: 0 rt: 2 rd: 3
#
# R-Type instruction.
# SUBU rs: 0 rt: 2 rd: 3
#
# R-Type instruction.
# SLT rs: 0 rt: 2 rd: 3
#
# R-Type instruction.
# SLTU rs: 0 rt: 2 rd: 3
#
# R-Type instruction.
# SLL/NOP(sa = 0) sa: 1 rt: 2 rd: 3
#
# R-Type instruction.
# SRL sa: 1 rt: 2 rd: 3
#
# R-Type instruction.
# SRA sa: 1 rt: 2 rd: 3
#
# R-Type instruction.
# AND rs: 0 rt: 2 rd: 3
#
# R-Type instruction.
# OR rs: 0 rt: 2 rd: 3
#
# R-Type instruction.
# XOR rs: 0 rt: 2 rd: 3
#
# R-Type instruction.
# NOR rs: 0 rt: 2 rd: 3
#
# I-Type instruction.
# ADDIU rs: 29 rt: 29 immediate: 65481
#
# I-Type instruction.
# SLTI rs: 0 rt: 2 immediate: 6217
#
# I-Type instruction.
# LW base: 0 rt: 2 offset: 6217
```

```
#
# I-Type instruction.
# SW base: 0 rt: 2 offset: 6217
#
# I-Type instruction.
# LUI rt: 2 immediate: 6223
#
# I-Type instruction.
# ORI rs: 0 rt: 2 immediate: 6221
#
# J-Type instruction.
# J instr_index: 137282
#
# J-Type instruction.
# BEQ rs: 0 rt: 2 offset: 6212
#
# J-Type instruction.
# BNE rs: 0 rt: 2 offset: 6213
#
# J-Type instruction.
# BGTZ rs: 0 offset: 6215
#
# J-Type instruction.
# BLEZ rs: 0 offset: 6214
#
# REGIMM-Type instruction.
# BLTZ rs: 0 offset: 65
#
# REGIMM-Type instruction.
# BGEZ rs: 0 offset: 18497
#
# ** Note: $finish : decode_tb.v(44)
# Time: 280 ns Iteration: 0 Instance: /decode_tb
```

2 Source Code

Listing 6: Fetch Module

```
//
// fetch.v
//
//

module fetch (
    clock,
    address,
    insn,
    insn_decode,
    pc,
    wren,
```

```
    stall
);

    input wire clock; // Clock signal for module. Make module sensitive to the positive edge
        of the clock
    input wire[0:31] insn; // this receives the instruction word from the main memory
        associated with the supplied PC
    input wire stall; // When asserted, the fetch effectively does a NOP. Data supplied to
        any of the outputs do not change, and the PC is not incremented by 4.

    output reg[0:31] address; // Output address supplied to the address input of the main
        memory. This signal transmits the PC for the instruction we are going to fetch
    output reg[0:31] insn_decode; // This transmits the instruction received from the main
        memory from insn
    output reg[0:31] pc; // transmits PC to the decode stage
    output reg wren; // indicates whether the fetch stage is performing a read or write to
        the main memory. Should always be asserted to a read for the fetch stage.

    reg[2:0] stage;

    initial
    begin
        $display("Initializing Fetch module");
        pc = 32'h8002_0000;
        wren = 1'b0;
    end

    always @(posedge clock)
    begin
        if (stall != 1'b1) begin
            address = pc;
            insn_decode = insn;
            pc = pc +4;
        end
    end

endmodule
```

Listing 7: Fetch Module Test Bench

```
//
// fetch_tb.v
//
// Fetch test bench
//

module fetch_tb;
    reg clock;

    wire[0:31] address;
    wire wren;
    wire[0:31] data_in;
```

```
wire[0:31] data_out;

wire[0:31] fetch_address;
wire fetch_wren;
wire[0:31] fetch_data_in;
wire[0:31] fetch_data_out;

wire[0:31] fetch_insn_decode;
wire[0:31] fetch_pc;
reg fetch_stall;

wire[0:31] srec_address;
wire srec_wren;
wire[0:31] srec_data_in;
wire[0:31] srec_data_out;

wire srec_done;

reg[0:31] tb_address;
reg tb_wren;
reg[0:31] tb_data_in;
wire[0:31] tb_data_out;


wire[0:31] bytes_read;
integer byte_count;
integer read_word;
reg[0:31] fetch_word;

reg instruction_valid;

mem_controller mcu(
    .clock (clock),
    .address (address),
    .wren (wren),
    .data_in (data_in),
    .data_out (data_out)
);

fetch DUT(
    .clock (clock),
    .address (fetch_address),
    .insn (fetch_data_in),
    .insn_decode (fetch_data_out),
    .pc (fetch_pc),
    .wren (fetch_wren),
    .stall (fetch_stall)
);

srec_parser #("srec_files/SumArray.srec") U0(
    .clock (clock),
```

```
    .mem_address (srec_address),
    .mem_wren (srec_wren),
    .mem_data_in (srec_data_in),
    .mem_data_out (srec_data_out),
    .done (srec_done),
    .bytes_read(bytes_read)
);

decode U1(
    .clock (clock),
    .insn (fetch_word),
    .insn_valid (instruction_valid)
);

assign address = srec_done ? (fetch_stall ? tb_address : fetch_address) : srec_address;
assign wren = srec_done ? (fetch_stall ? tb_wren : fetch_wren) : srec_wren;
assign tb_data_out = data_out;
assign fetch_data_in = data_out;
assign data_in = srec_done ? (fetch_stall ? tb_data_in : fetch_data_in) : srec_data_in;

// Specify when to stop the simulation
event terminate_sim;
initial begin
    @ (terminate_sim);
    #10 $finish;
end

initial begin
    clock = 1;
    fetch_stall = 1;
    instruction_valid = 1'b0;
end

always begin
    #5 clock = !clock;
end

initial begin
    $dumpfile("fetch_tb.vcd");
    $dumpvars;
end

initial begin
    @(posedge srec_done);
    @(posedge clock);
    byte_count = bytes_read;
    tb_address = 32'h8002_0000;
    tb_wren = 1'b0;
    instruction_valid = 1'b0;
    fetch_stall = 0;
    while (byte_count  0) begin
```

```
            @(posedge clock);
            if ((fetch_address - 4)  32'h8002_000) begin
                instruction_valid = 1'b0;
            end else begin
                instruction_valid = 1'b1;
            end

                read_word = tb_data_out;
            fetch_word = fetch_data_out;

            if (fetch_address-4 = 32'h8002_0000) begin

                $display("PC: %X, Instruction: %b", fetch_address - 4, fetch_word);
            end
            tb_address = tb_address + 4;
            byte_count = byte_count - 4; // 27 = 0010 011
        end
        instruction_valid = 1'b0;
        -terminate_sim;
    end

endmodule
```

Listing 8: Decode Module

```
//
// decode.v
//
//

module decode (
    clock,
    insn,
    insn_valid,
    pc
);

    input wire[0:31] insn;
    input wire[0:31] pc;
    input wire clock;
    input wire insn_valid;


    reg[0:1] insn_type;

    wire[0:5] opcode;
    wire[4:0] rs;
    wire[4:0] rt;
    wire[4:0] rd;
    wire[4:0] shift_amount;
    wire[5:0] funct;
    wire[15:0] immediate;
```

```
wire[25:0] j_address;
wire[4:0] base;
wire[15:0] offset;

// Instruction types
parameter I_TYPE = 0;
parameter J_TYPE = 1;
parameter R_TYPE = 2;
parameter INVALID_INS = 3;

assign opcode = insn[0:5];
assign rs = insn[6:10];
assign base = insn[6:10];
assign rt = insn[11:15];
assign rd = insn[16:20];
assign shift_amount = insn[20:25];
assign funct = insn[26:31];
assign immediate = insn[16:31];
assign offset = insn[16:31];
assign j_address = insn[6:31];

always @(posedge clock)
begin
        if(insn_valid) begin
        case(opcode)
                // R-TYPE
            6'b000000: begin
                $display("R-Type instruction.");
                    case(funct)
                    6'b100000: //ADD
                          $display("ADD rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b100001: //ADDU
                          $display("ADDU rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b100010: //SUB
                          $display("SUB rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b100011: //SUBU
                          $display("SUBU rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b101010: //SLT
                          $display("SLT rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b101011: //SLTU
                          $display("SLTU rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b000000: //SLL
                          $display("SLL/NOP(sa = 0) sa: %d rt: %d rd: %d", shift_amount, rt,
                               rd);
                        6'b000010: //SRL
                          $display("SRL sa: %d rt: %d rd: %d", shift_amount, rt, rd);
                        6'b000011: //SRA
                          $display("SRA sa: %d rt: %d rd: %d", shift_amount, rt, rd);
                        6'b100100: //AND
                          $display("AND rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b100101: //OR
```

```
            $display("OR rs: %d rt: %d rd: %d", rs, rt, rd);
          6'b100110: //XOR
            $display("XOR rs: %d rt: %d rd: %d", rs, rt, rd);
          6'b100111: //NOR
            $display("NOR rs: %d rt: %d rd: %d", rs, rt, rd);

          default:
            $display("unimplemented ADD type intruction");
        endcase // case (funct)
    end // case: 6'b000000

    //I-TYPE
    6'b001001:  //ADDIU
begin
  $display("I-Type instruction.");
     $display("ADDIU rs: %d rt: %d immediate: %d", rs, rt, immediate);
end
    6'b001010:  //SLTI
begin
  $display("I-Type instruction.");
     $display("SLTI rs: %d rt: %d immediate: %d", rs, rt, immediate);
end
    6'b100011:  //LW
begin
  $display("I-Type instruction.");
     $display("LW base: %d rt: %d offset: %d", base, rt, offset);
end
    6'b101011:  //SW
begin
  $display("I-Type instruction.");
     $display("SW base: %d rt: %d offset: %d", base, rt, offset);
end
    6'b001111:  //LUI
begin
  $display("I-Type instruction.");
     $display("LUI rt: %d immediate: %d", rt, immediate);
end
    6'b001101:  //ORI
begin
  $display("I-Type instruction.");
     $display("ORI rs: %d rt: %d immediate: %d", rs, rt, immediate);
end
    //J-TYPE
    6'b000010:  //J
begin
  $display("J-Type instruction.");
     $display("J instr_index: %d", j_address);
end
    6'b000100:  //BEQ
begin
  $display("J-Type instruction.");
```

```
                $display("BEQ rs: %d rt: %d offset: %d", rs, rt, offset);
            end
                6'b000101:  //BNE
            begin
              $display("J-Type instruction.");
                $display("BNE rs: %d rt: %d offset: %d", rs, rt, offset);
            end
                6'b000111:  //BGTZ
            begin
              $display("J-Type instruction.");
                $display("BGTZ rs: %d offset: %d", rs, offset);
            end
                6'b000110:  //BLEZ
            begin
              $display("J-Type instruction.");
                $display("BLEZ rs: %d offset: %d", rs, offset);
            end

                6'b000001: //REGIMM instructions
                begin
                $display("REGIMM-Type instruction.");
                        case(rt)
                            5'b00000: //BLTZ
                              $display("BLTZ rs: %d offset: %d", rs, offset);
                            5'b00001: //BGEZ
                              $display("BGEZ rs: %d offset: %d", rs, offset);
                            default:
                              $display("REGIMM not implemented");
                        endcase // case (rt)
                end
            default:
                $display("unimplemented/incorrect intruction");

        endcase // case (insn[26:31])

        $display("");
        end // if (insn_valid = 1'b1)

    end // always @ (posedge clock)

endmodule
```

Listing 9: Decode Module Test Bench

```
//
// decode_tb.v
//
// Decode test bench
//

module decode_tb;
    reg clock;
```

```
reg i_valid;
reg[0:31] insn;
reg [0:31] pc;

wire[0:1] insn_type;
wire[5:0] opcode;
wire[4:0] rs;
wire[4:0] rt;
wire[4:0] rd;
wire[4:0] shift_amount;
wire[5:0] funct;
wire[15:0] immediate;
wire[25:0] j_address;

decode DUT(
    .clock (clock),
    .insn (insn),
        .insn_valid (i_valid),
    .pc (pc)
);

// initialization of the test bench
initial begin
    clock = 1;
end

// clock signal
always begin
    #5 clock = !clock;
end

// Specify when to stop the simulation
event terminate_sim;
initial begin
    @ (terminate_sim);
    #10 $finish;
end

// TEST CASES BEGIN HERE
initial begin

    @ (posedge clock);
    i_valid = 1'b1;

    //ADD
    // | op || rs|| rt|| rd|| sh|| fu |
    insn = 32'b00000000101110000011001010100000;
    @ (posedge clock);

    //ADDU
    insn = 32'b00000000001000110001000000100001;
```

```
@ (posedge clock);

//SUB
insn = 32'b00000000000000010000110000110010;
@ (posedge clock);

//SUBU
insn = 32'b00000000000000010000110000110011;
@ (posedge clock);

//SLT
insn = 32'b00000000000000010000110000101010;
@ (posedge clock);

//SLTU
insn = 32'b00000000000000010000110000101011;
@ (posedge clock);

 //SLL
insn = 32'b00000000000000010000110000001000000;
@ (posedge clock);

//SRL
insn = 32'b00000000000000010000110000001000010;
@ (posedge clock);

//SRA
insn = 32'b00000000000000010000110000001000011;
@ (posedge clock);

 //AND
insn = 32'b00000000000000010000110000110100100;
@ (posedge clock);

//OR
insn = 32'b00000000000000010000110000110100101;
@ (posedge clock);

//XOR
insn = 32'b00000000000000010000110000110100110;
@ (posedge clock);

//NOR
insn = 32'b00000000000000010000110000110100111;
@ (posedge clock);

//ADDIU
insn = 32'b00100111101111011111111111001001;
@ (posedge clock);

//SLTI
```

```
insn = 32'b00101000000000100001100001001001;
@ (posedge clock);

//LW
insn = 32'b10001100000000100001100001001001;
@ (posedge clock);

//SW
insn = 32'b10101100000000100001100001001001;
@ (posedge clock);

//LUI
insn = 32'b00111100000000100001100001001111;
@ (posedge clock);

//ORI
insn = 32'b00110100000000100001100001001101;
@ (posedge clock);

//J
insn = 32'b00001000000000100001100001000010;
@ (posedge clock);

//BEQ
insn = 32'b00010000000000100001100001000100;
@ (posedge clock);

//BNE
insn = 32'b00010100000000100001100001000101;
@ (posedge clock);

//BGTZ
insn = 32'b00011100000000100001100001000111;
@ (posedge clock);

//BLEZ
insn = 32'b00011000000000100001100001000110;
@ (posedge clock);

//BLTZ
insn = 32'b00000100000000000000000000001000001;
@ (posedge clock);

//BGEZ
insn = 32'b00000100000000001010010000001000001;
@ (posedge clock);

//
// signal to end the simulation
- terminate_sim;
```

```
    end

endmodule // decode_tb
```