CprE 381 – Computer Organization and Assembly-Level Programming

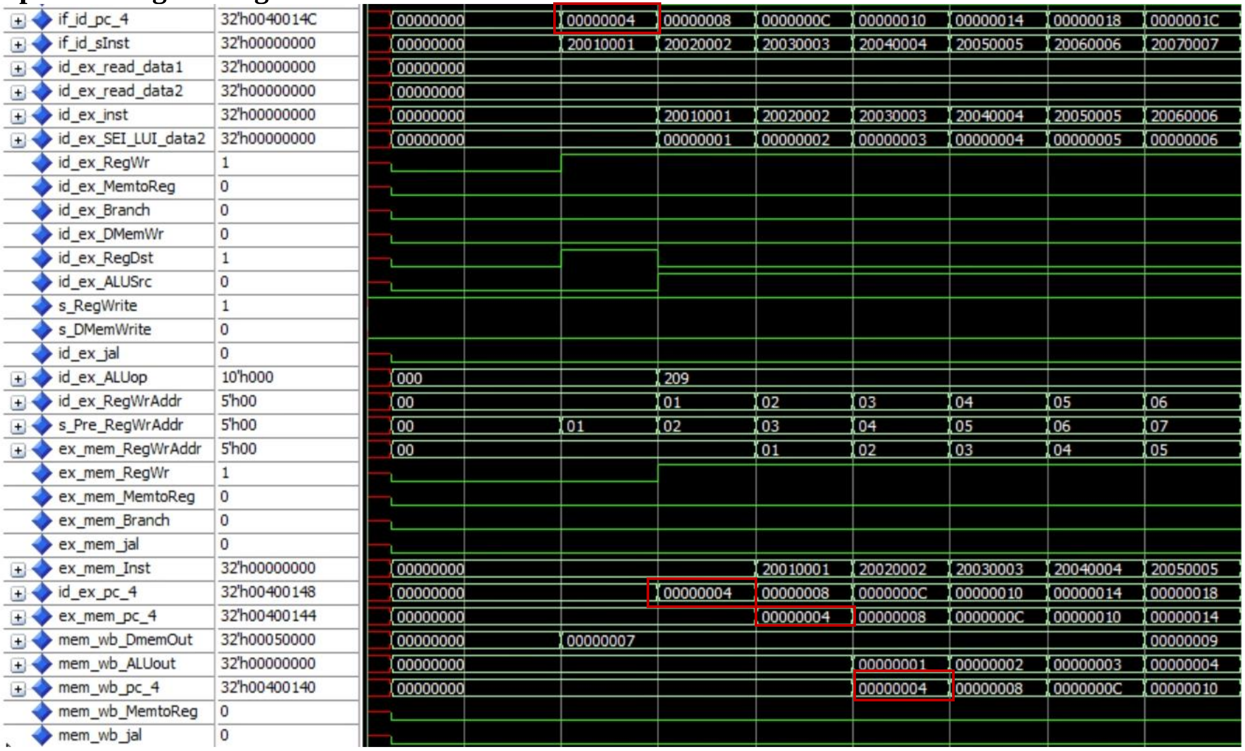
MIPS Pipelined Processor w/wo Data Dependencies

Author: Samuel Ferguson

Pipeline Register Signals

IF/ID Stage	ID/EX Stage	EX/MEM Stage	MEM/WB Stage
Flush	Flush	Flush	Flush
Stall	Stall	Stall	Stall
PC+4	Reg Write	Reg Write	Reg Write
Instruction	MemtoReg	MemtoReg	MemtoReg
	Branch	Branch	DMEM data out
	BranchNE	Mem Write	DMEM address in
	MemWrite	ALUout	Reg Write Address
	RegDst	ALU in B	JAL from Ctrl
	ALUop from Ctrl	Rs	PC + 4
	ALUSrc	Rt	
	Register rs Data	Reg Write Address	
	Register rt Data	JAL from Ctrl	
	Imed	Jump from Ctrl	
	Reg Write Address	PC + 4	
	Rs		
	Rt		
	Jump from Ctrl		
	JAL from Ctrl		
	PC+4		
	Branch Address		

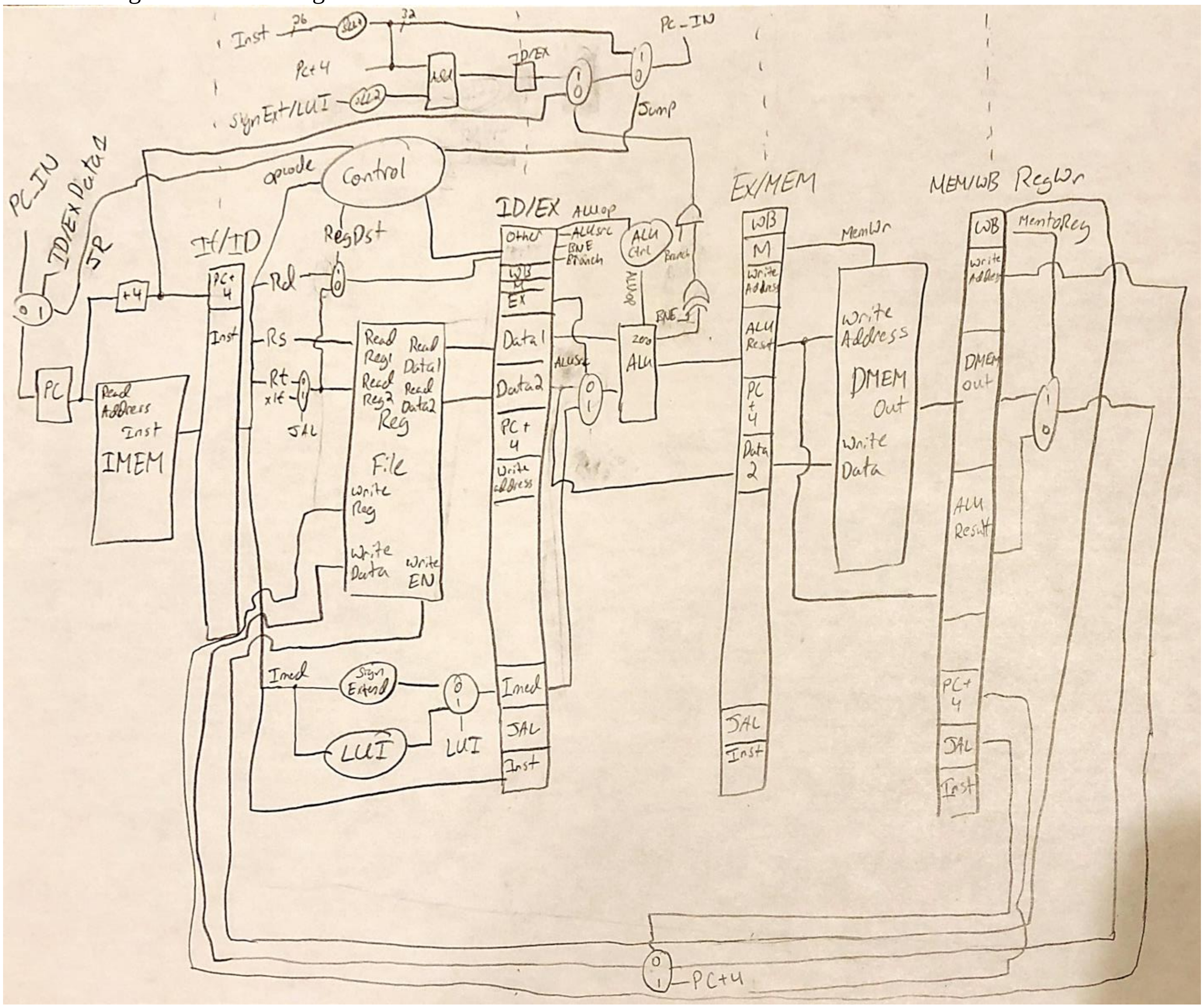
Pipelined Register Signals Waveform



if_id_pc_4	32'h0040014C
if_id_sInst	32'h00000000
id_ex_read_data1	32'h00000000
id_ex_read_data2	32'h00000000
id_ex_inst	32'h00000000
id_ex_SEI_LUI_data2	32'h00000000
id_ex_RegWr	1
id_ex_MemtoReg	0
id_ex_Branch	0
id_ex_DMemWr	0
id_ex_RegDst	1
id_ex_ALUSrc	0
s_RegWrite	1
s_DMemWrite	0
id_ex_jal	0
id_ex_ALUop	10'h000
id_ex_RegWrAddr	5'h00
s_Pre_RegWrAddr	5'h00
ex_mem_RegWrAddr	5'h00
ex_mem_RegWr	1
ex_mem_MemtoReg	0
ex_mem_Branch	0
ex_mem_jal	0
ex_mem_Inst	32'h00000000
id_ex_pc_4	32'h00400148
ex_mem_pc_4	32'h00400144
mem_wb_DmemOut	32'h00050000
mem_wb_ALUout	32'h00000000
mem_wb_pc_4	32'h00400140
mem_wb_MemtoReg	0
mem_wb_jal	0

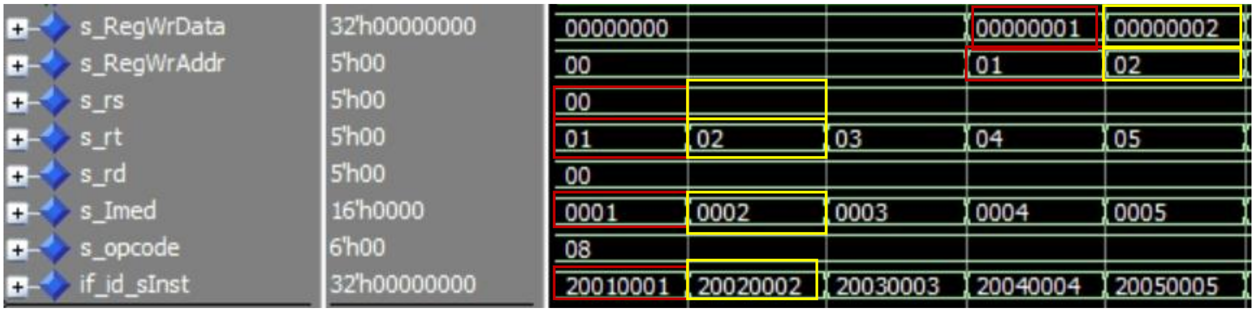
PC + 4 is the only signal that propagates through all four stages of the pipeline because it is needed for the Jump and Link instruction to change the register write data to the PC +4 data when the instruction was executed. As you can see in the waveform, the PC+4 signal propagates from if_id_PC_4 to mem_wb_pc_4 four cycles later. The same is done with the next PC + 4 value of x8 and can be seen directly after each highlighted PC + 4 signal.

This is the schematic of the pipelined MIPS processor. The main challenges I faced with implementing the pipelined processor was pipelining address instructions such as Jump And Link and Branch. The Jump And Link instruction needed to be pipelined through all four registers to that it could store the address to register 31. The Branch instruction was pipelined to the EX stage where the ALU could calculate the Zero. For the assembly level instructions I needed to add three nops after each RAW instruction because that was the amount of registers between the WB stage and the ID stage. For jump and JAL instructions, I needed to use one nop so that the ID stage could jump without loading an instruction into the IF/ID register. I needed three nops after branch instructions because I needed to stop instructions from being loaded into the IF/ID register and ID/EX register and then nop for one more cycle to clear the current PC address. The jump register instruction needed two nops after the instruction to clear the PC value and then stop the IF/ID register from loading in a false instruction.



Demonstration of Instructions without data dependency handling.

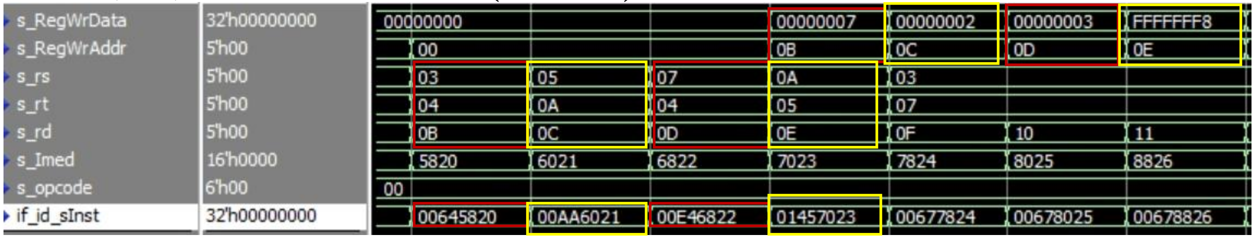
ADDI
ADDI $1, $0, 1 (RED)
ADDI $2, $0, 2 (YELLOW)



```
s_RegWrData   32'h00000000   00000000                          00000001   00000002
s_RegWrAddr   5'h00          00                                01         02
s_rs          5'h00          00
s_rt          5'h00          01        02        03        04         05
s_rd          5'h00          00
s_Imed        16'h0000       0001      0002      0003      0004       0005
s_opcode      6'h00          08
if_id_sInst   32'h00000000   20010001  20020002  20030003  20040004   20050005
```

ADD & SUB & ADDU & SUBU

Add $11, $3, $4 (RED) #Place 3+4=7 in $11
Addu $12, $5, $10 (YELLOW) #Place -3 + 5= 2 in $12
Sub $13, $7, $4 #Place 7-4=3 in $13 (RED)
Subu $14, $10, $5 #Place -3-5=-8 in $14 (YELLOW)

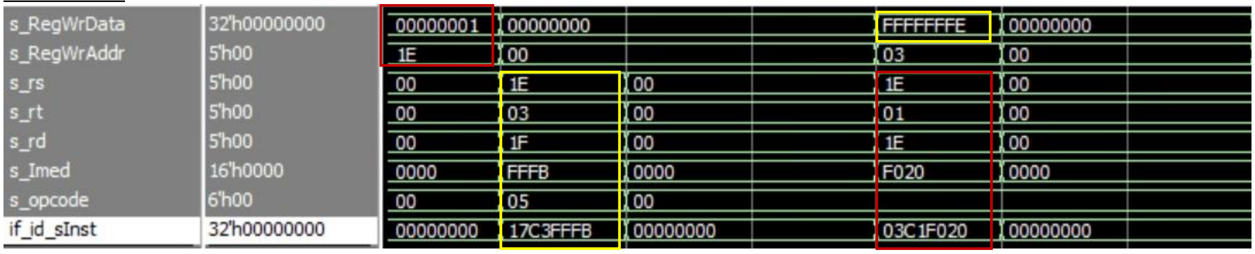


```
s_RegWrData   32'h00000000   00000000                                  00000007  00000002  00000003  FFFFFFF8
s_RegWrAddr   5'h00          00                                        0B        0C        0D        0E
s_rs          5'h00          03        05        07        0A        03
s_rt          5'h00          04        0A        04        05        07
s_rd          5'h00          0B        0C        0D        0E        0F        10        11
s_Imed        16'h0000       5820      6021      6822      7023      7824      8025      8826
s_opcode      6'h00          00
if_id_sInst   32'h00000000   00645820  00AA6021  00E46822  01457023  00677824  00678025  00678826
```

BNE

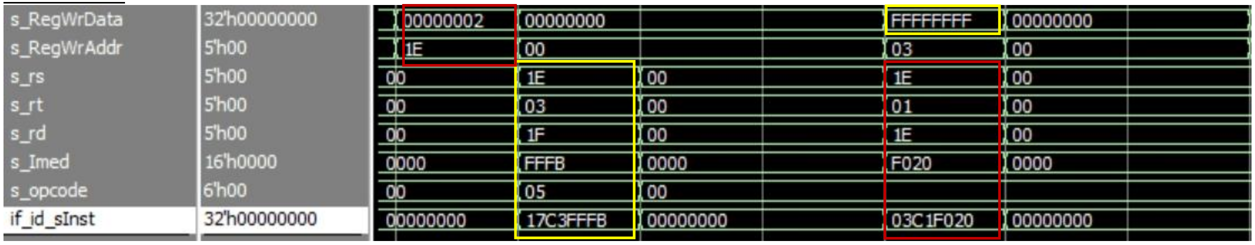
Looper:
    add $30, $30, $1 (RED) #Add 1 to $30 until $30 == 3
    nop
    nop
    nop
    bne $30, $3, Looper (YELLOW)
nop
nop
nop
add $11, $3, $4 (BLUE) # Place 7 in $11

Iteration 1



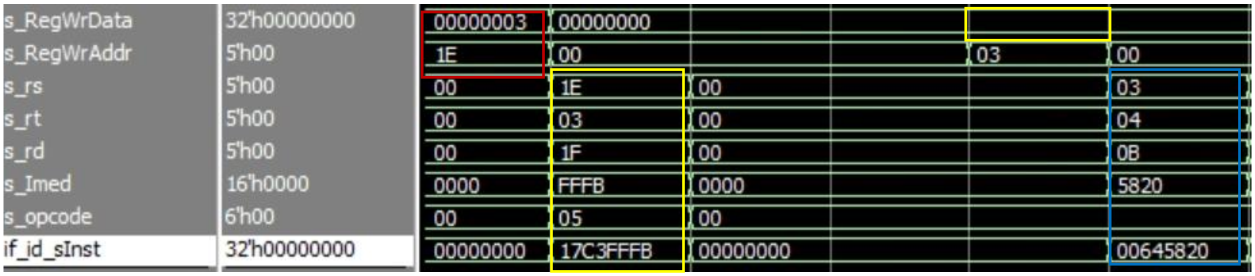
```
s_RegWrData   32'h00000000   00000001   00000000                        FFFFFFFE   00000000
s_RegWrAddr   5'h00          1E         00                              03         00
s_rs          5'h00          00         1E         00                   1E         00
s_rt          5'h00          00         03         00                   01         00
s_rd          5'h00          00         1F         00                   1E         00
s_Imed        16'h0000       0000       FFFB       0000                 F020       0000
s_opcode      6'h00          00         05         00
if_id_sInst   32'h00000000   00000000   17C3FFFB   00000000             03C1F020   00000000
```

Iteration 2



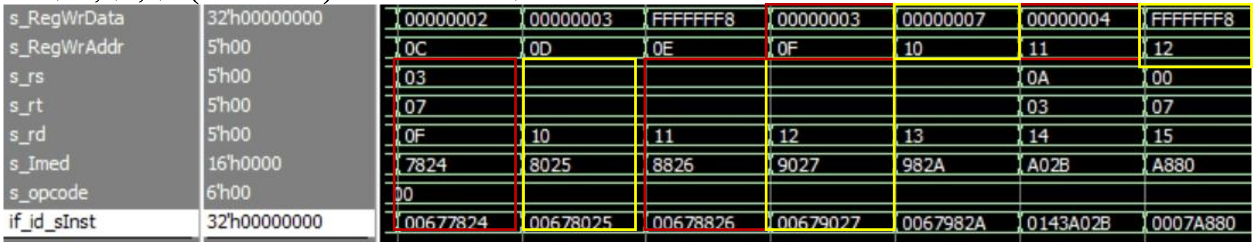
```
s_RegWrData   32'h00000000   00000002   00000000                        FFFFFFFF   00000000
s_RegWrAddr   5'h00          1E         00                              03         00
s_rs          5'h00          00         1E         00                   1E         00
s_rt          5'h00          00         03         00                   01         00
s_rd          5'h00          00         1F         00                   1E         00
s_Imed        16'h0000       0000       FFFB       0000                 F020       0000
s_opcode      6'h00          00         05         00
if_id_sInst   32'h00000000   00000000   17C3FFFB   00000000             03C1F020   00000000
```

Iteration 3


s_RegWrData 32'h00000000 00000003 00000000
s_RegWrAddr 5'h00 1E 00 03 00
s_rs 5'h00 00 1E 00 03
s_rt 5'h00 00 03 00 04
s_rd 5'h00 00 1F 00 0B
s_Imed 16'h0000 0000 FFFB 0000 5820
s_opcode 6'h00 00 05 00
if_id_sInst 32'h00000000 00000000 17C3FFFB 00000000 00645820

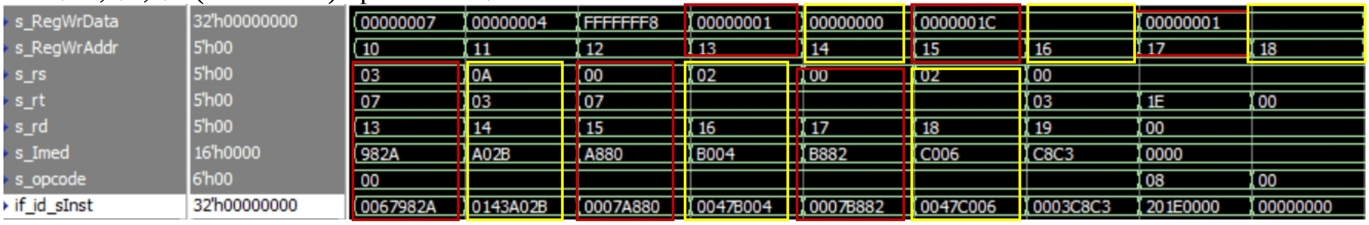
AND & OR & XOR & NOR

and $15, $3, $7 (RED) #Place 3 in $15
or $16, $3, $7 (YELLOW) #Place 7 in $16
xor $17, $3, $7 (RED) #Place 4 in $17
nor $18, $3, $7 (YELLOW) #Place a -8 in $18


s_RegWrData 32'h00000000 00000002 00000003 FFFFFFF8 00000003 00000007 00000004 FFFFFFF8
s_RegWrAddr 5'h00 0C 0D 0E 0F 10 11 12
s_rs 5'h00 03 0A 00
s_rt 5'h00 07 03 07
s_rd 5'h00 0F 10 11 12 13 14 15
s_Imed 16'h0000 7824 8025 8826 9027 982A A02B A880
s_opcode 6'h00 00
if_id_sInst 32'h00000000 00677824 00678025 00678826 00679027 0067982A 0143A02B 0007A880

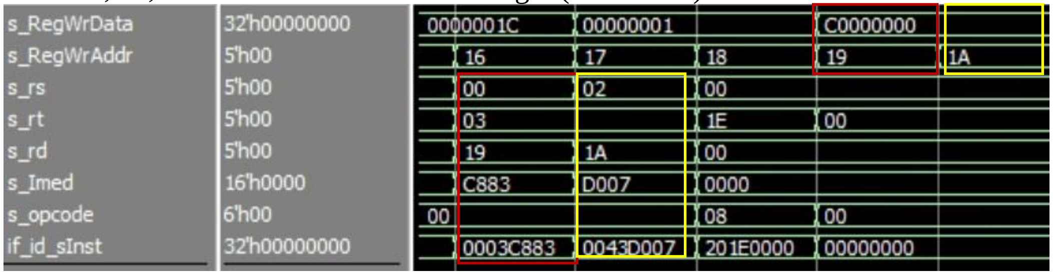
SLT & SLTU & SLL & SLV & SRL & SRLV

slt $19, $3, $7 (RED) #Place 1 in $19
sltu $20, $10, $3 (YELLOW) #Place 0 in $20
sll $21, $7, 2 (RED) #place 28 in $21
sllv $22, $7, $2 (YELLOW) #place 28 in $22
srl $23, $7, 2 (RED) #Place 1 in $23
srlv $24, $7, $2 (YELLOW) #place 1 in $24


s_RegWrData 32'h00000000 00000007 00000004 FFFFFFF8 00000001 00000000 0000001C 00000001
s_RegWrAddr 5'h00 10 11 12 13 14 15 16 17 18
s_rs 5'h00 03 0A 00 02 00 02 00
s_rt 5'h00 07 03 07 03 1E 00
s_rd 5'h00 13 14 15 16 17 18 19 00
s_Imed 16'h0000 982A A02B A880 B004 B882 C006 C8C3 0000
s_opcode 6'h00 00 08 00
if_id_sInst 32'h00000000 0067982A 0143A02B 0007A880 0047B004 0007B882 0047C006 0003C8C3 201E0000 00000000

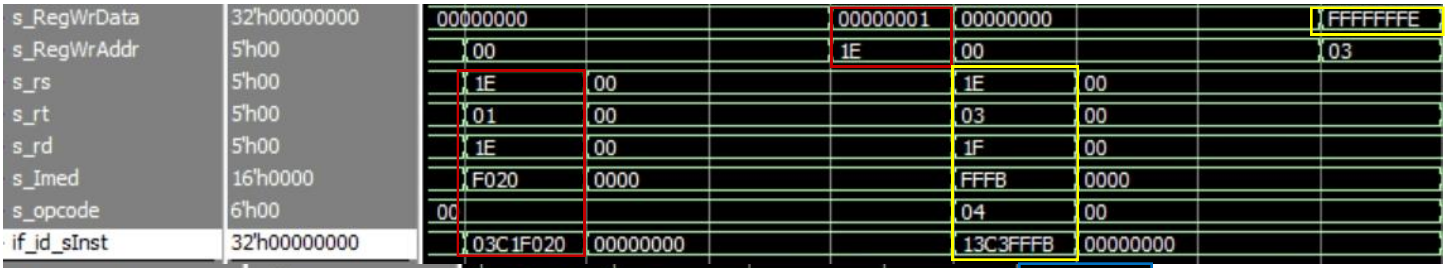
SRA & SRV

sra $25, $3, 2 #Sift x3 sra 2 bits to right (RED)
srav $26, $3, $2 #shift x3 sra 2 bits to the right (YELLOW)

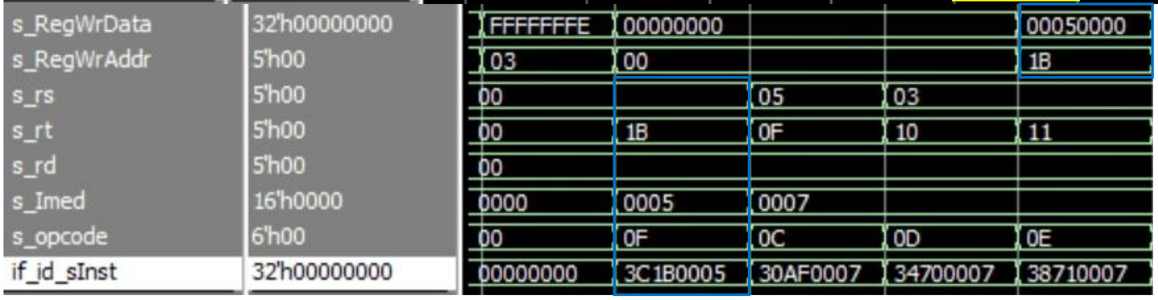

s_RegWrData 32'h00000000 0000001C 00000001 C0000000
s_RegWrAddr 5'h00 16 17 18 19 1A
s_rs 5'h00 00 02 00
s_rt 5'h00 03 1E 00
s_rd 5'h00 19 1A 00
s_Imed 16'h0000 C883 D007 0000
s_opcode 6'h00 00 08 00
if_id_sInst 32'h00000000 0003C883 0043D007 201E0000 00000000

BEQ & LUI

```
Looper_2:
      add $30, $30, $1 (RED) #Add 1 to $30 and branch if $30 == 3
      nop
      nop
      nop
      beq $30, $3, Looper_2 (YELLOW) #$30 = 1 so will continue
nop
nop
nop
lui $27, 5 (BLUE) #Place x00050000 in $27
```



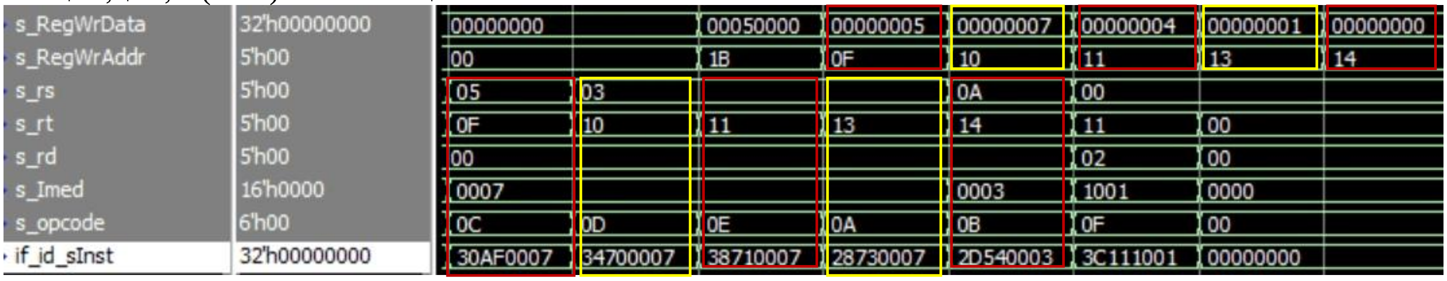
```
s_RegWrData   32'h00000000   00000000  00000001  00000000  FFFFFFFE
s_RegWrAddr   5'h00          00        1E        00        03
s_rs          5'h00          1E        00        1E        00
s_rt          5'h00          01        00        03        00
s_rd          5'h00          1E        00        1F        00
s_Imed        16'h0000       F020      0000      FFFB      0000
s_opcode      6'h00          00        04        00
if_id_sInst   32'h00000000   03C1F020  00000000  13C3FFFB  00000000
```



```
s_RegWrData   32'h00000000   FFFFFFFE  00000000  00050000
s_RegWrAddr   5'h00          03        00        1B
s_rs          5'h00          00        05        03
s_rt          5'h00          00        1B        0F        10        11
s_rd          5'h00          00
s_Imed        16'h0000       0000      0005      0007
s_opcode      6'h00          00        0F        0C        0D        0E
if_id_sInst   32'h00000000   00000000  3C1B0005  30AF0007  34700007  38710007
```

ANDI & ORI & XORI & SLTI & SLTIU

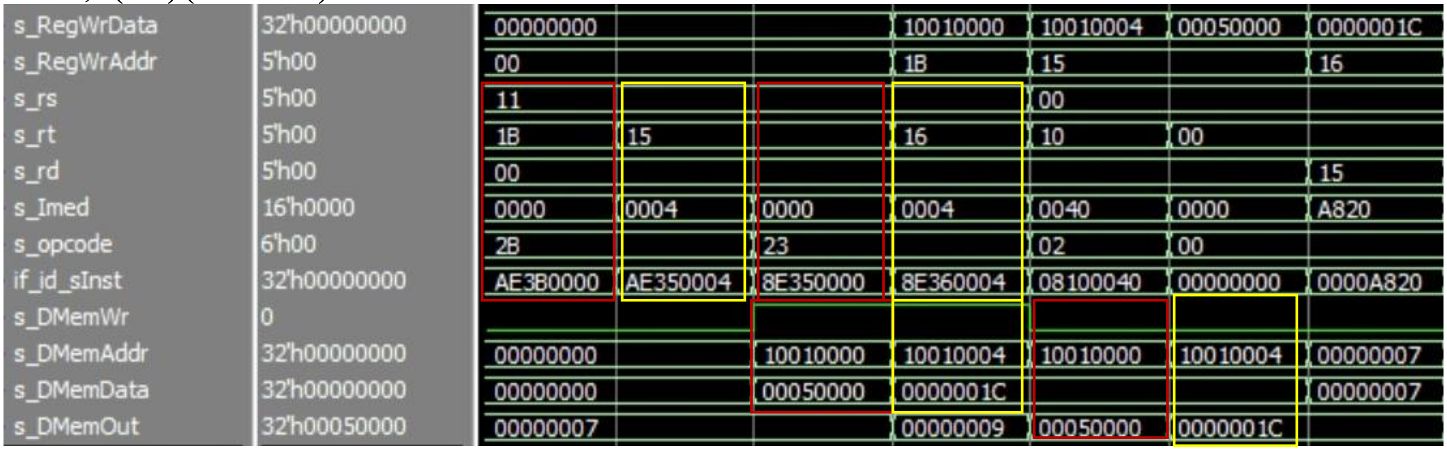
andi $15, $5, (RED) 7 #Place 5 in $15
ori $16, $3, 7 (YELLOW) #Place 7 in $16
xori $17, $3, 7 (RED) #Place 4 in $17
slti $19, $3, 7 (YELLOW) #Place 1 in $19
sltiu $20, $10, 3 (RED) #Place 0 in $20



```
s_RegWrData   32'h00000000   00000000  00050000  00000005  00000007  00000004  00000001  00000000
s_RegWrAddr   5'h00          00        1B        0F        10        11        13        14
s_rs          5'h00          05        03        0A        00
s_rt          5'h00          0F        10        11        13        14        11        00
s_rd          5'h00          00        02        00
s_Imed        16'h0000       0007      0003      1001      0000
s_opcode      6'h00          0C        0D        0E        0A        0B        0F        00
if_id_sInst   32'h00000000   30AF0007  34700007  38710007  28730007  2D540003  3C111001  00000000
```

LW & SW

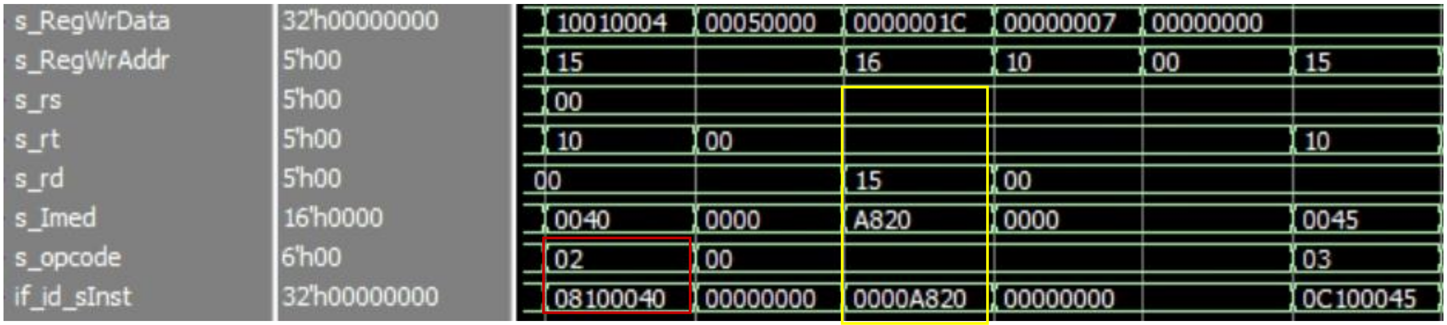
sw $27, 0($17) (RED) #Place x00050000 in address 4
sw $21, 4($17) (YELLOW) #Place x1c in address 8
lw $21, 0($17) (RED) #Place x00050000 in $21
lw $22, 4($17) (YELLOW) #Place x1c in $22



```
s_RegWrData   32'h00000000   00000000  10010000  10010004  00050000  0000001C
s_RegWrAddr   5'h00          00        1B        15        16
s_rs          5'h00          11        00
s_rt          5'h00          1B        15        16        10        00
s_rd          5'h00          00        15
s_Imed        16'h0000       0000      0004      0000      0004      0040      0000      A820
s_opcode      6'h00          2B        23        02        00
if_id_sInst   32'h00000000   AE3B0000  AE350004  8E350000  8E360004  08100040  00000000  0000A820
s_DMemWr      0
s_DMemAddr    32'h00000000   00000000  10010000  10010004  10010000  10010004  00000007
s_DMemData    32'h00000000   00000000  00050000  0000001C  00000007
s_DMemOut     32'h00050000   00000007  00000009  00050000  0000001C
```

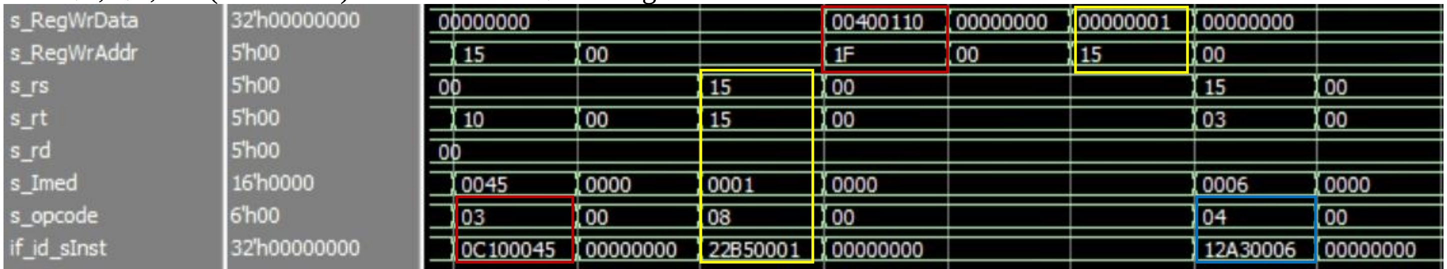
Jump

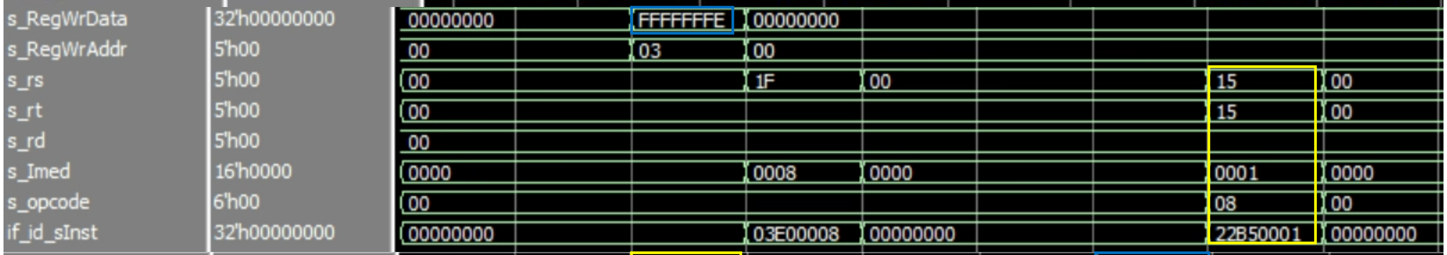
j skip_add (RED)
nop
add $21, $22, $0 #Place x1c in $21(THIS SHOULD BE SKIPPED)
skip_add:
    add $21, $0, $0 (YELLOW) #Place x0 in $21

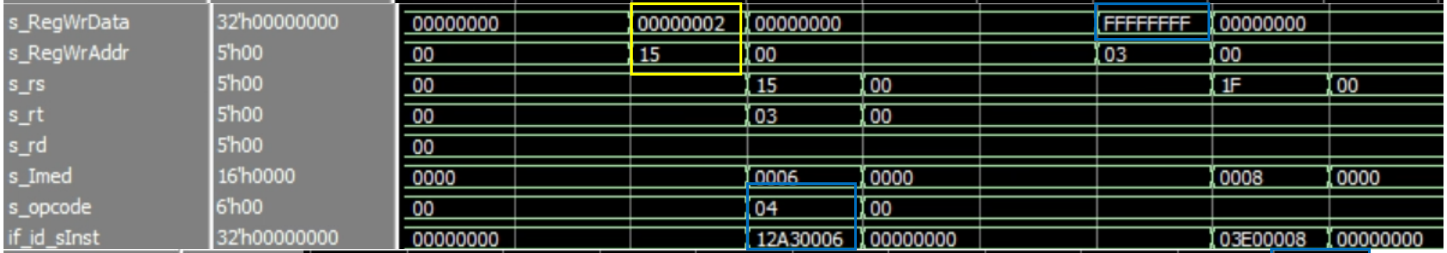

s_RegWrData 32'h00000000 10010004 00050000 0000001C 00000007 00000000
s_RegWrAddr 5'h00 15 16 10 00 15
s_rs 5'h00 00
s_rt 5'h00 10 00 10
s_rd 5'h00 00 15 00
s_Imed 16'h0000 0040 0000 A820 0000 0045
s_opcode 6'h00 02 00 03
if_id_sInst 32'h00000000 08100040 00000000 0000A820 00000000 0C100045

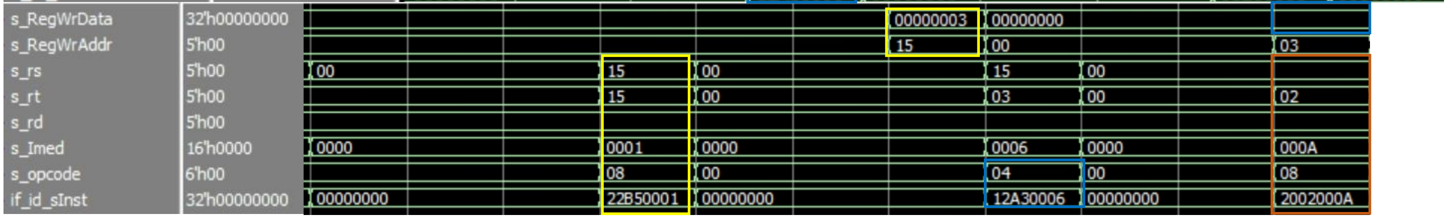
JAL & JR

```
jal task (RED)
nop
task: #Loops until $21 equals 3
      add $21, $21, 1 (YELLOW) #increment $21 by 1 three times
      nop
      nop
      nop
      beq $21, $3, exit_task (BLUE)
      nop
      nop
      nop
      jr $ra (RED)
nop
nop
exit_task:
addi  $2,  $0,  10 (ORANGE) # Place "10" in $v0 to signal an "exit" or "halt"
```


s_RegWrData 32'h00000000 00000000 00400110 00000000 00000001 00000000
s_RegWrAddr 5'h00 15 00 1F 00 15 00
s_rs 5'h00 00 15 00 15 00
s_rt 5'h00 10 00 15 00 03 00
s_rd 5'h00 00
s_Imed 16'h0000 0045 0000 0001 0000 0006 0000
s_opcode 6'h00 03 00 08 00 04 00
if_id_sInst 32'h00000000 0C100045 00000000 22B50001 00000000 12A30006 00000000

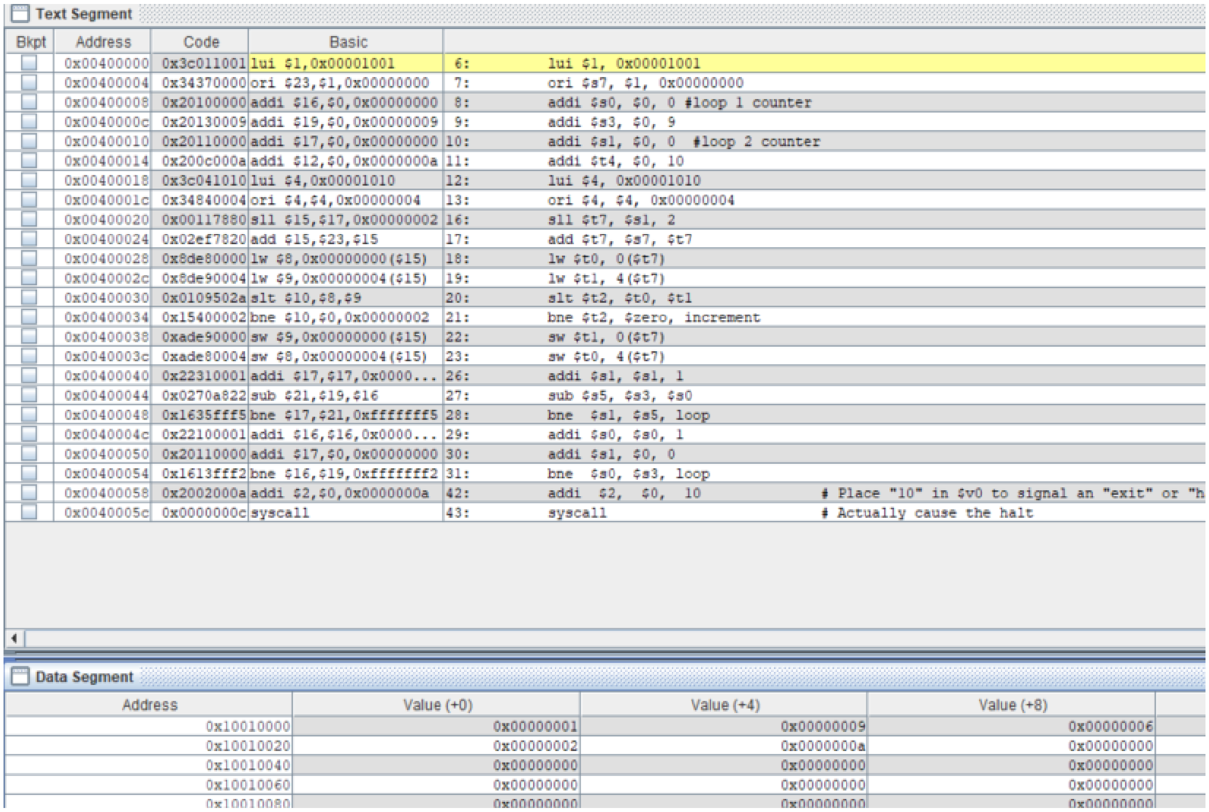

s_RegWrData 32'h00000000 00000000 FFFFFFFE 00000000
s_RegWrAddr 5'h00 00 03 00
s_rs 5'h00 00 1F 00 15 00
s_rt 5'h00 00 15 00
s_rd 5'h00 00
s_Imed 16'h0000 0000 0008 0000 0001 0000
s_opcode 6'h00 00 08 00
if_id_sInst 32'h00000000 00000000 03E00008 00000000 22B50001 00000000


s_RegWrData 32'h00000000 00000000 00000002 00000000 FFFFFFFF 00000000
s_RegWrAddr 5'h00 00 15 00 03 00
s_rs 5'h00 00 15 00 1F 00
s_rt 5'h00 00 03 00
s_rd 5'h00 00
s_Imed 16'h0000 0000 0006 0000 0008 0000
s_opcode 6'h00 00 04 00
if_id_sInst 32'h00000000 00000000 12A30006 00000000 03E00008 00000000


s_RegWrData 32'h00000000 00000003 00000000
s_RegWrAddr 5'h00 15 00 03
s_rs 5'h00 00 15 00 15 00
s_rt 5'h00 15 00 03 00 02
s_rd 5'h00
s_Imed 16'h0000 0000 0001 0000 0006 0000 000A
s_opcode 6'h00 08 00 04 00 08
if_id_sInst 32'h00000000 00000000 22B50001 00000000 12A30006 00000000 2002000A

Overall the new forwarding unit and Hazard control unit have completely gotten rid of the nops in the code. However, branch changes include register flushing. So, there are intermediate nops being added to the instructions to accommodate for the delay of getting the new instruction addresses. The forwarding unit has allowed for a massive reduction in the amount of nops needed in the assembly level instruction and does not require any flushing or stalls except for instructions that are four apart. This allows for a much greater speedup in the overall processor.

Demonstration of an assembly bubble sort application to be run on processor.



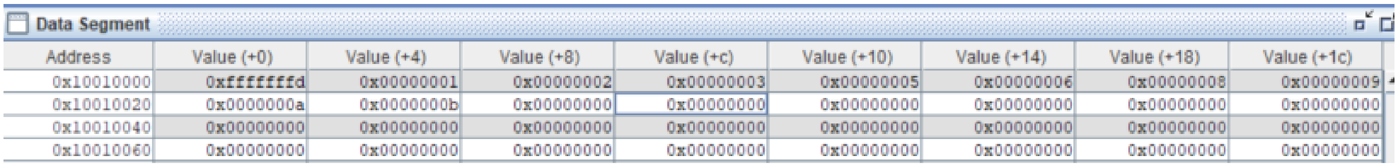
Text Segment

Bkpt	Address	Code	Basic	
□	0x00400000	0x3c011001	lui $1,0x00001001	6: lui $1, 0x00001001
□	0x00400004	0x34370000	ori $23,$1,0x00000000	7: ori $s7, $1, 0x00000000
□	0x00400008	0x20100000	addi $16,$0,0x00000000	8: addi $s0, $0, 0 #loop 1 counter
□	0x0040000c	0x20130009	addi $19,$0,0x00000009	9: addi $s3, $0, 9
□	0x00400010	0x20110000	addi $17,$0,0x00000000	10: addi $s1, $0, 0 #loop 2 counter
□	0x00400014	0x200c000a	addi $12,$0,0x0000000a	11: addi $t4, $0, 10
□	0x00400018	0x3c041010	lui $4,0x00001010	12: lui $4, 0x00001010
□	0x0040001c	0x34840004	ori $4,$4,0x00000004	13: ori $4, $4, 0x00000004
□	0x00400020	0x00117880	sll $15,$17,0x00000002	16: sll $t7, $s1, 2
□	0x00400024	0x02ef7820	add $15,$23,$15	17: add $t7, $s7, $t7
□	0x00400028	0x8de80000	lw $8,0x00000000($15)	18: lw $t0, 0($t7)
□	0x0040002c	0x8de90004	lw $9,0x00000004($15)	19: lw $t1, 4($t7)
□	0x00400030	0x0109502a	slt $10,$8,$9	20: slt $t2, $t0, $t1
□	0x00400034	0x15400002	bne $10,$0,0x00000002	21: bne $t2, $zero, increment
□	0x00400038	0xade90000	sw $9,0x00000000($15)	22: sw $t1, 0($t7)
□	0x0040003c	0xade80004	sw $8,0x00000004($15)	23: sw $t0, 4($t7)
□	0x00400040	0x22310001	addi $17,$17,0x0000...	26: addi $s1, $s1, 1
□	0x00400044	0x0270a822	sub $21,$19,$16	27: sub $s5, $s3, $s0
□	0x00400048	0x1635fff5	bne $17,$21,0xfffffff5	28: bne $s1, $s5, loop
□	0x0040004c	0x22100001	addi $16,$16,0x0000...	29: addi $s0, $s0, 1
□	0x00400050	0x20110000	addi $17,$0,0x00000000	30: addi $s1, $0, 0
□	0x00400054	0x1613fff2	bne $16,$19,0xfffffff2	31: bne $s0, $s3, loop
□	0x00400058	0x2002000a	addi $2,$0,0x0000000a	42: addi $2, $0, 10 # Place "10" in $v0 to signal an "exit" or "h
□	0x0040005c	0x0000000c	syscall	43: syscall # Actually cause the halt

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0x00000001	0x00000009	0x00000006
0x10010020	0x00000002	0x0000000a	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000

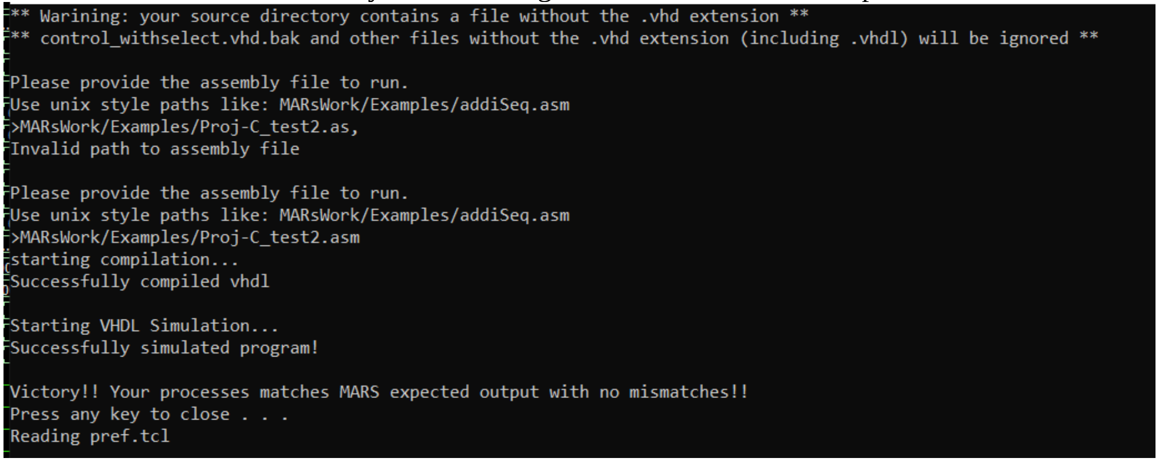
The order of values stored in memory are 1, 9, 6, 3, 5, 8, -3, 11, 2, 10. This is the starting order the values are sorted from.



Data Segment

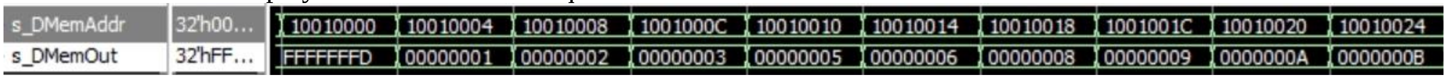
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0xfffffffd	0x00000001	0x00000002	0x00000003	0x00000005	0x00000006	0x00000008	0x00000009
0x10010020	0x0000000a	0x0000000b	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Once sorted, the values are ordered to -3, 1, 2, 3, 5, 6, 8, 9, 10, 11.
Mars simulation showed victory when running the code for validation in processor.



```
** Warining: your source directory contains a file without the .vhd extension **
** control_withselect.vhd.bak and other files without the .vhd extension (including .vhdl) will be ignored **

Please provide the assembly file to run.
Use unix style paths like: MARsWork/Examples/addiSeq.asm
>MARsWork/Examples/Proj-C_test2.as,
Invalid path to assembly file

Please provide the assembly file to run.
Use unix style paths like: MARsWork/Examples/addiSeq.asm
>MARsWork/Examples/Proj-C_test2.asm
starting compilation...
Successfully compiled vhdl

Starting VHDL Simulation...
Successfully simulated program!

Victory!! Your processes matches MARS expected output with no mismatches!!
Press any key to close . . .
Reading pref.tcl
```

Now that the MARs result confirms the validity of our program, I can run the testbench program to compare the MARs output and our processor's output. So, at the end of the end of bubble sort application, I added a lw instruction for each address to display the data values in sequential order.

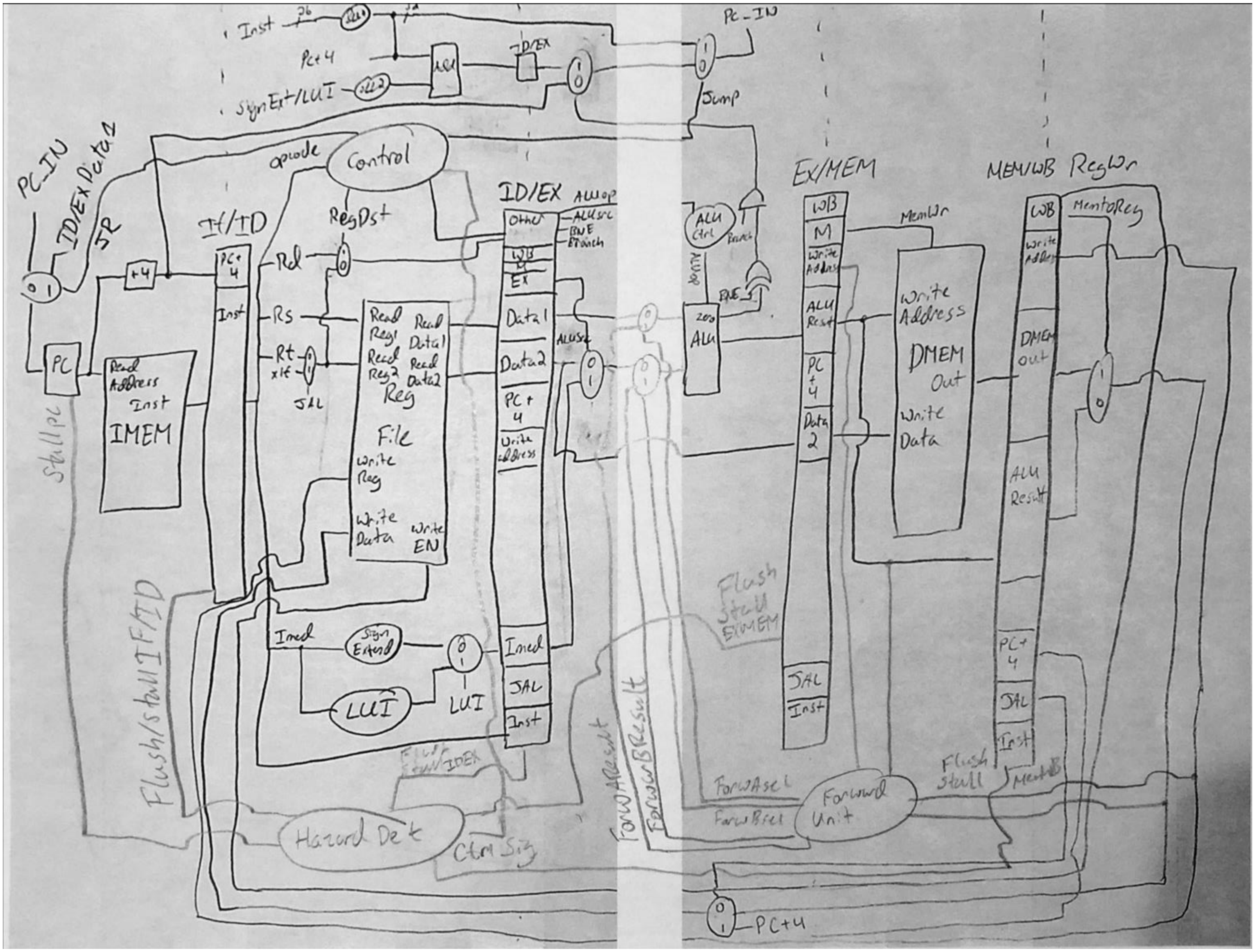


s_DMemAddr	32'h00...	10010000	10010004	10010008	1001000C	10010010	10010014	10010018	1001001C	10010020	10010024
s_DMemOut	32'hFF...	FFFFFFFD	00000001	00000002	00000003	00000005	00000006	00000008	00000009	0000000A	0000000B

Timing of Processor without dependency handling.

The maximum frequency is 50.5 MHz. The critical path is 22.751 ns, it goes through id_ex_register to the ALU, to the instruction_ctrl_mod, to finally the instruction_address_module.

High-level schematic drawing of the interconnection between components for the MIPS Hardware-scheduled pipelined processor with data dependency handling.

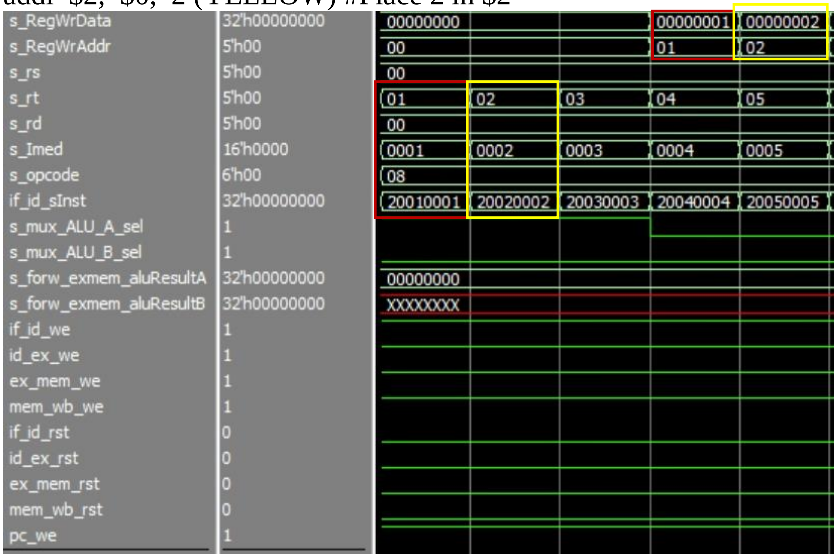


Demonstration of Instructions with data dependency handling.

ADDI

addi $1, $0, 1 (RED) #Place 1 in $1

addi $2, $0, 2 (YELLOW) #Place 2 in $2



s_RegWrData 32'h00000000 00000000 00000001 00000002
s_RegWrAddr 5'h00 00 01 02
s_rs 5'h00 00
s_rt 5'h00 01 02 03 04 05
s_rd 5'h00 00
s_Imed 16'h0000 0001 0002 0003 0004 0005
s_opcode 6'h00 08
if_id_sInst 32'h00000000 20010001 20020002 20030003 20040004 20050005
s_mux_ALU_A_sel 1
s_mux_ALU_B_sel 1
s_forw_exmem_aluResultA 32'h00000000 00000000
s_forw_exmem_aluResultB 32'h00000000 XXXXXXXX
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1

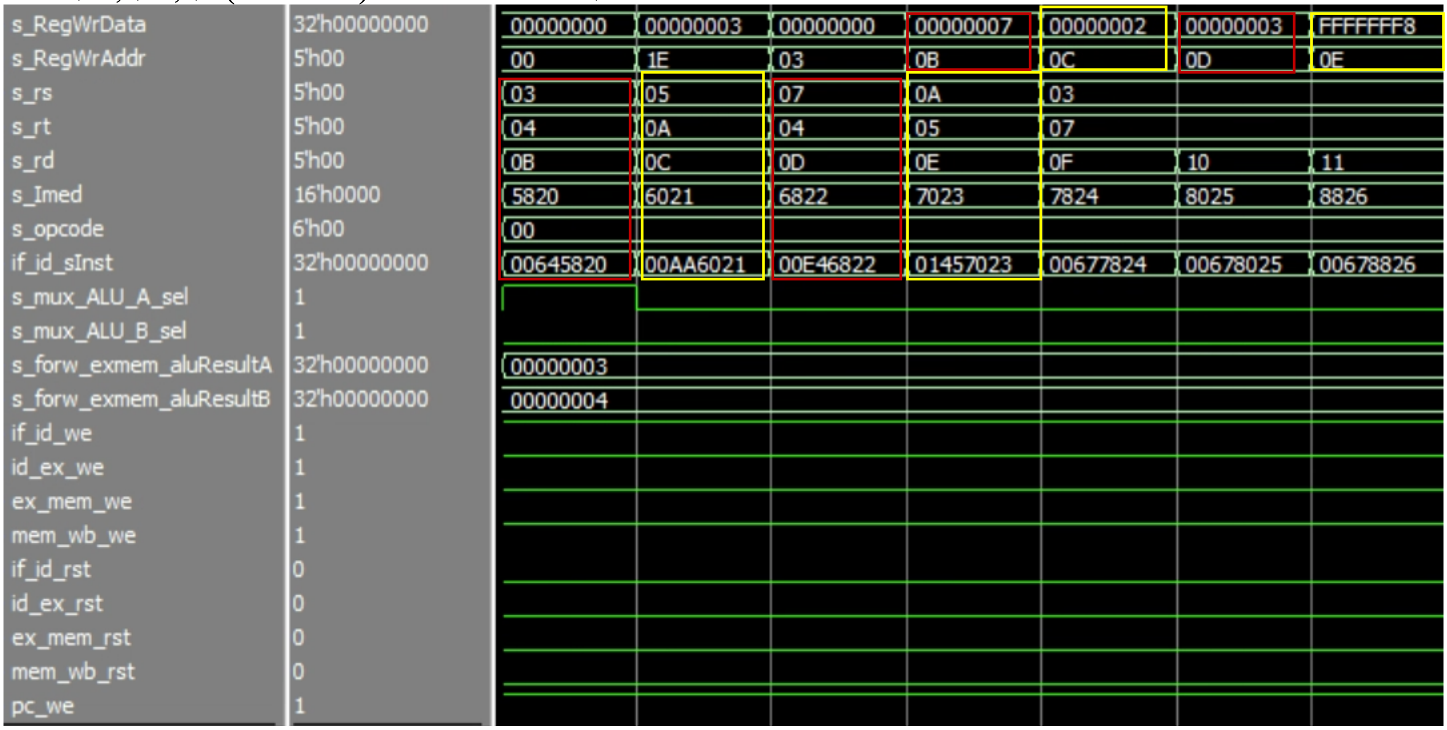
ADD & SUB & ADDU & SUBU

Add $11, $3, $4 (RED) #Place 3+4=7 in $11

Addu $12, $5, $10 (YELLOW) #Place -3 + 5= 2 in $12

Sub $13, $7, $4 (RED) #Place 7-4=3 in $13

Subu $14, $10, $5 (YELLOW) #Place -3-5=-8 in $14



s_RegWrData 32'h00000000 00000000 00000003 00000000 00000007 00000002 00000003 FFFFFFF8
s_RegWrAddr 5'h00 00 1E 03 0B 0C 0D 0E
s_rs 5'h00 03 05 07 0A 03
s_rt 5'h00 04 0A 04 05 07
s_rd 5'h00 0B 0C 0D 0E 0F 10 11
s_Imed 16'h0000 5820 6021 6822 7023 7824 8025 8826
s_opcode 6'h00 00
if_id_sInst 32'h00000000 00645820 00AA6021 00E46822 01457023 00677824 00678025 00678826
s_mux_ALU_A_sel 1
s_mux_ALU_B_sel 1
s_forw_exmem_aluResultA 32'h00000000 00000003
s_forw_exmem_aluResultB 32'h00000000 00000004
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1

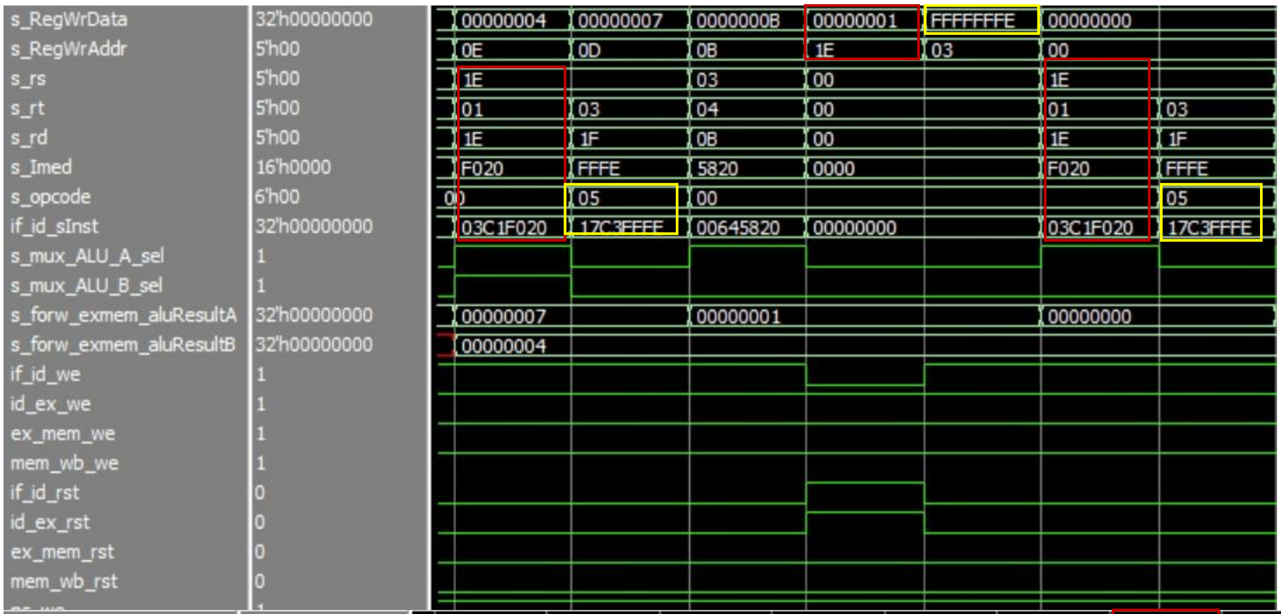
BNE

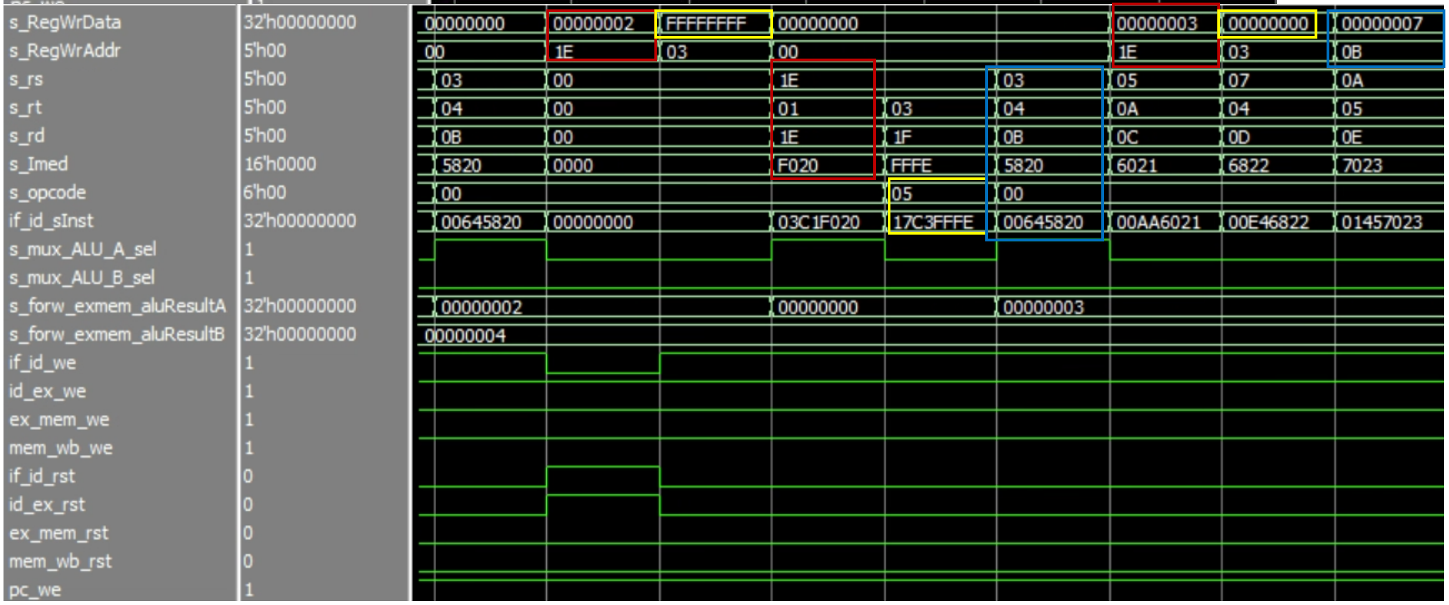
Looper:

    add $30, $30, $1 (RED) #add 1 to $30 unitl $30 == 1

    bne $30, $3, Looper (YELLOW)

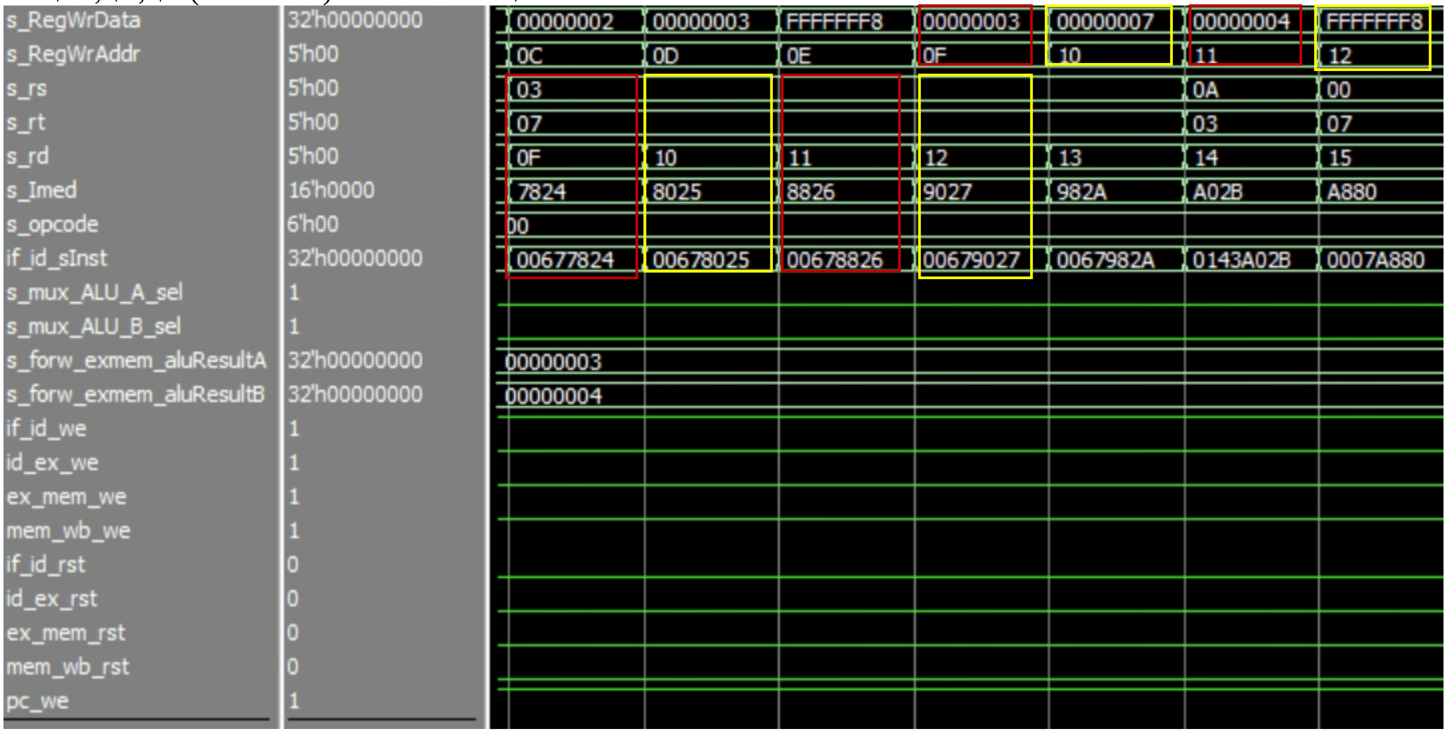
add $11, $3, $4 (BLUE) #Place 7 in $11


s_RegWrData 32'h00000000
s_RegWrAddr 5'h00
s_rs 5'h00
s_rt 5'h00
s_rd 5'h00
s_Imed 16'h0000
s_opcode 6'h00
if_id_sInst 32'h00000000
s_mux_ALU_A_sel 1
s_mux_ALU_B_sel 1
s_forw_exmem_aluResultA 32'h00000000
s_forw_exmem_aluResultB 32'h00000000
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
00000004 00000007 0000000B 00000001 FFFFFFFE 00000000
0E 0D 0B 1E 03 00
1E 03 00 1E
01 03 04 00 01 03
1E 1F 0B 00 1E 1F
F020 FFFE 5820 0000 F020 FFFE
00 05 00 05
03C1F020 17C3FFFE 00645820 00000000 03C1F020 17C3FFFE
00000007 00000001 00000000
00000004


s_RegWrData 32'h00000000
s_RegWrAddr 5'h00
s_rs 5'h00
s_rt 5'h00
s_rd 5'h00
s_Imed 16'h0000
s_opcode 6'h00
if_id_sInst 32'h00000000
s_mux_ALU_A_sel 1
s_mux_ALU_B_sel 1
s_forw_exmem_aluResultA 32'h00000000
s_forw_exmem_aluResultB 32'h00000000
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1
00000000 00000002 FFFFFFFF 00000000 00000003 00000000 00000007
00 1E 03 00 1E 03 0B
03 00 1E 03 05 07 0A
04 00 01 03 04 0A 04 05
0B 00 1E 1F 0B 0C 0D 0E
5820 0000 F020 FFFE 5820 6021 6822 7023
00 05 00
00645820 00000000 03C1F020 17C3FFFE 00645820 00AA6021 00E46822 01457023
00000002 00000000 00000003
00000004

AND & OR & XOR & NOR

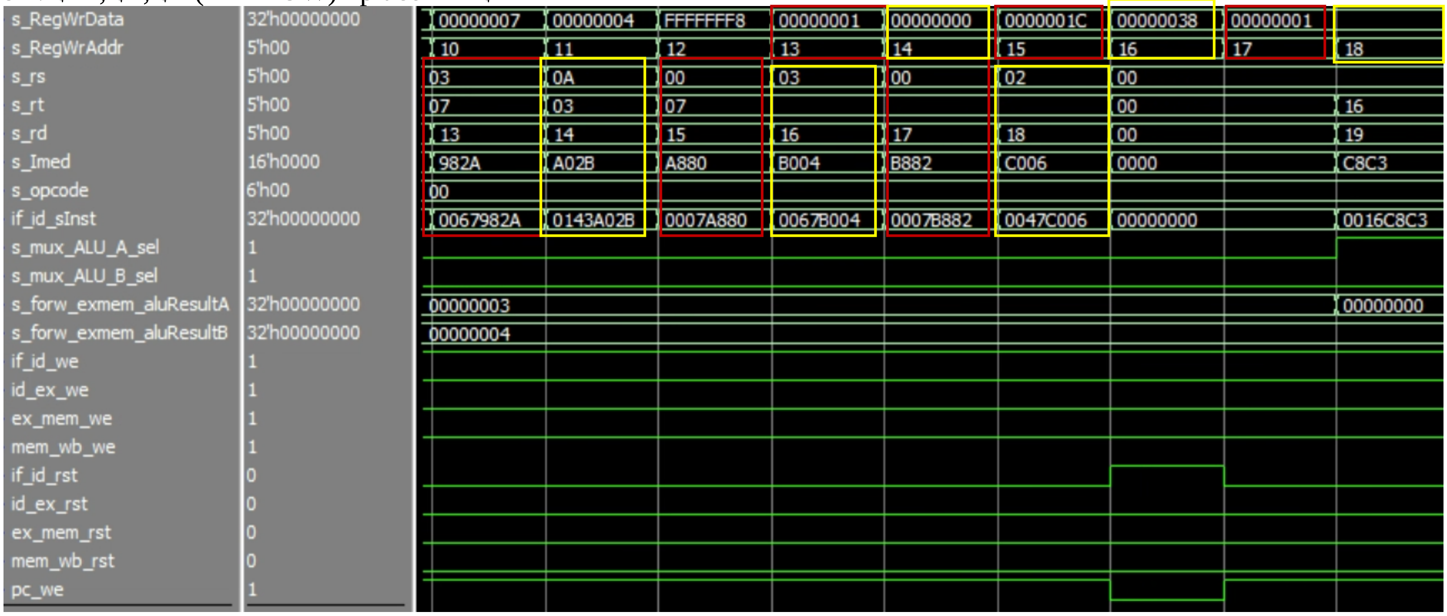
and $15, $3, $7 (RED) #Place 3 in $15
or $16, $3, $7 (YELLOW) #Place 7 in $16
xor $17, $3, $7 (RED) #Place 4 in $17
nor $18, $3, $7 (YELLOW) #Place a -8 in $18



```
s_RegWrData              32'h00000000  00000002 00000003 FFFFFFF8 00000003 00000007 00000004 FFFFFFF8
s_RegWrAddr              5'h00         0C 0D 0E 0F 10 11 12
s_rs                     5'h00         03 0A 00
s_rt                     5'h00         07 03 07
s_rd                     5'h00         0F 10 11 12 13 14 15
s_Imed                   16'h0000      7824 8025 8826 9027 982A A02B A880
s_opcode                 6'h00         00
if_id_sInst              32'h00000000  00677824 00678025 00678826 00679027 0067982A 0143A02B 0007A880
s_mux_ALU_A_sel          1
s_mux_ALU_B_sel          1
s_forw_exmem_aluResultA  32'h00000000  00000003
s_forw_exmem_aluResultB  32'h00000000  00000004
if_id_we                 1
id_ex_we                 1
ex_mem_we                1
mem_wb_we                1
if_id_rst                0
id_ex_rst                0
ex_mem_rst               0
mem_wb_rst               0
pc_we                    1
```

SLT & SLTU & SLL & SLV & SRL & SRLV

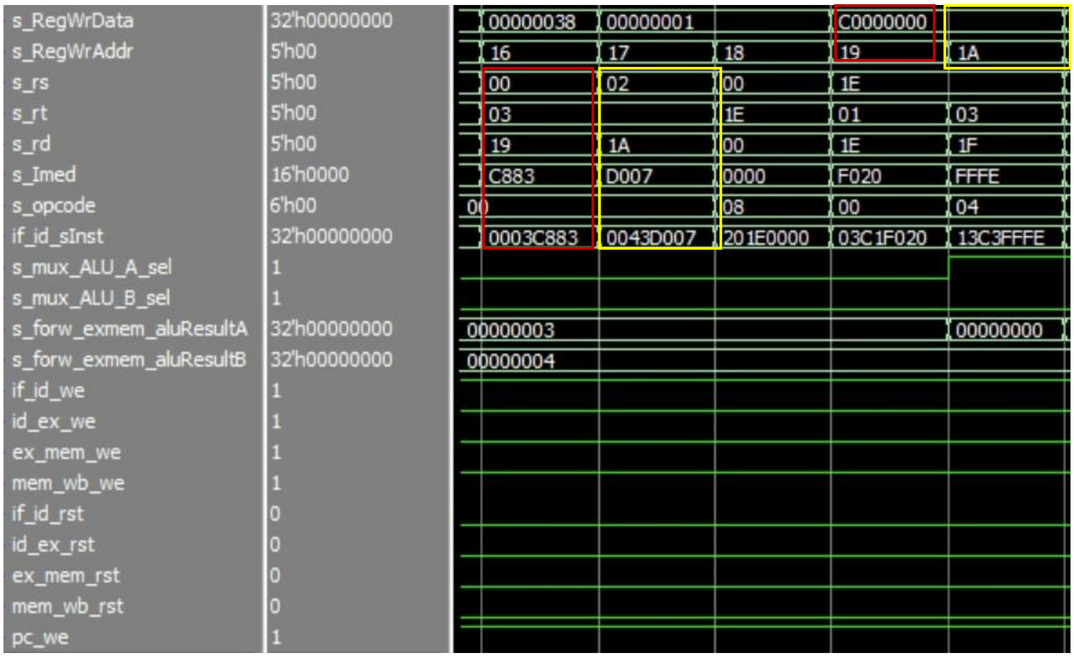
slt $19, $3, $7 (RED) #Place 1 in $19
sltu $20, $10, $3 (YELLOW) #Place 0 in $20
sll $21, $7, 2 (RED) #place 28 in $21
sllv $22, $7, $3 (YELLOW) #place 56 in $22
srl $23, $7, 2 (RED) #Place 1 in $23
srlv $24, $7, $2 (YELLOW) #place 1 in $24



```
s_RegWrData              32'h00000000  00000007 00000004 FFFFFFF8 00000001 00000000 0000001C 00000038 00000001
s_RegWrAddr              5'h00         10 11 12 13 14 15 16 17 18
s_rs                     5'h00         03 0A 00 03 00 02 00
s_rt                     5'h00         07 03 07 00 16
s_rd                     5'h00         13 14 15 16 17 18 00 19
s_Imed                   16'h0000      982A A02B A880 B004 B882 C006 0000 C8C3
s_opcode                 6'h00         00
if_id_sInst              32'h00000000  0067982A 0143A02B 0007A880 0067B004 0007B882 0047C006 00000000 0016C8C3
s_mux_ALU_A_sel          1
s_mux_ALU_B_sel          1
s_forw_exmem_aluResultA  32'h00000000  00000003 00000000
s_forw_exmem_aluResultB  32'h00000000  00000004
if_id_we                 1
id_ex_we                 1
ex_mem_we                1
mem_wb_we                1
if_id_rst                0
id_ex_rst                0
ex_mem_rst               0
mem_wb_rst               0
pc_we                    1
```

SRA & SRAV

sra $25, $3, 2 #Sift x3 sra 2 bits to right (RED)
srav $26, $3, $2 #shift x3 sra 2 bits to the right (YELLOW)



```
s_RegWrData               32'h00000000   00000038 | 00000001 |          | C0000000 |
s_RegWrAddr               5'h00          16       | 17       | 18       | 19       | 1A
s_rs                      5'h00          00       | 02       | 00       | 1E       |
s_rt                      5'h00          03       |          | 1E       | 01       | 03
s_rd                      5'h00          19       | 1A       | 00       | 1E       | 1F
s_Imed                    16'h0000       C883     | D007     | 0000     | F020     | FFFE
s_opcode                  6'h00          00       |          | 08       | 00       | 04
if_id_sInst               32'h00000000   0003C883 | 0043D007 | 201E0000 | 03C1F020 | 13C3FFFE
s_mux_ALU_A_sel           1
s_mux_ALU_B_sel           1
s_forw_exmem_aluResultA   32'h00000000   00000003                                   | 00000000
s_forw_exmem_aluResultB   32'h00000000   00000004
if_id_we                  1
id_ex_we                  1
ex_mem_we                 1
mem_wb_we                 1
if_id_rst                 0
id_ex_rst                 0
ex_mem_rst                0
mem_wb_rst                0
pc_we                     1
```

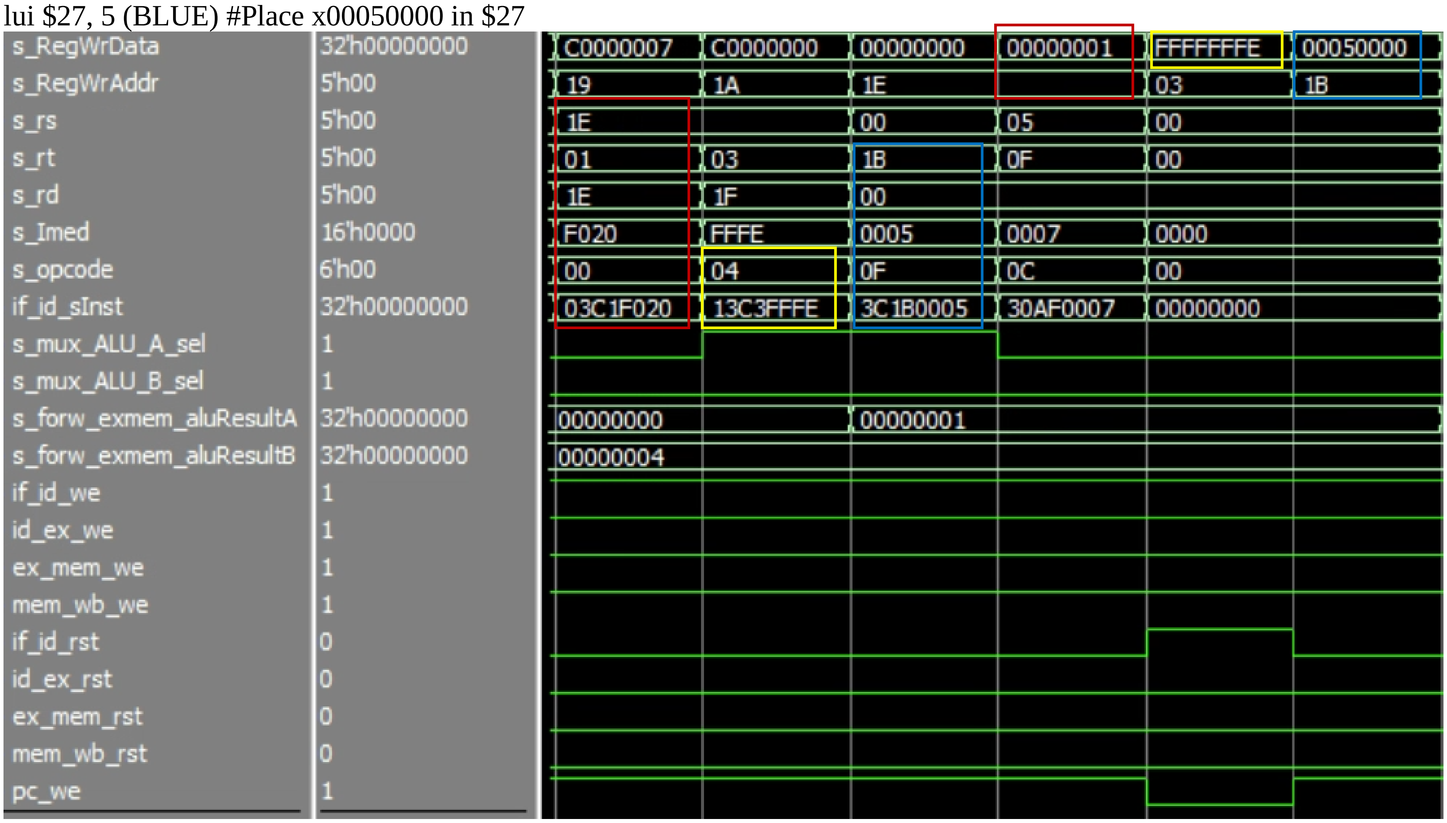
BEQ & LUI

Looper_2:

add $30, $30, $1 (RED) #Add 1 to $20 and branch if $30 == 3

beq $30, $3, Looper_2 (YELLOW) #$30 = 1so will continue

lui $27, 5 (BLUE) #Place x00050000 in $27



```
s_RegWrData               32'h00000000   C0000007 | C0000000 | 00000000 | 00000001 | FFFFFFFE | 00050000
s_RegWrAddr               5'h00          19       | 1A       | 1E       |          | 03       | 1B
s_rs                      5'h00          1E       |          | 00       | 05       | 00
s_rt                      5'h00          01       | 03       | 1B       | 0F       | 00
s_rd                      5'h00          1E       | 1F       | 00
s_Imed                    16'h0000       F020     | FFFE     | 0005     | 0007     | 0000
s_opcode                  6'h00          00       | 04       | 0F       | 0C       | 00
if_id_sInst               32'h00000000   03C1F020 | 13C3FFFE | 3C1B0005 | 30AF0007 | 00000000
s_mux_ALU_A_sel           1
s_mux_ALU_B_sel           1
s_forw_exmem_aluResultA   32'h00000000   00000000            | 00000001
s_forw_exmem_aluResultB   32'h00000000   00000004
if_id_we                  1
id_ex_we                  1
ex_mem_we                 1
mem_wb_we                 1
if_id_rst                 0
id_ex_rst                 0
ex_mem_rst                0
mem_wb_rst                0
pc_we                     1
```

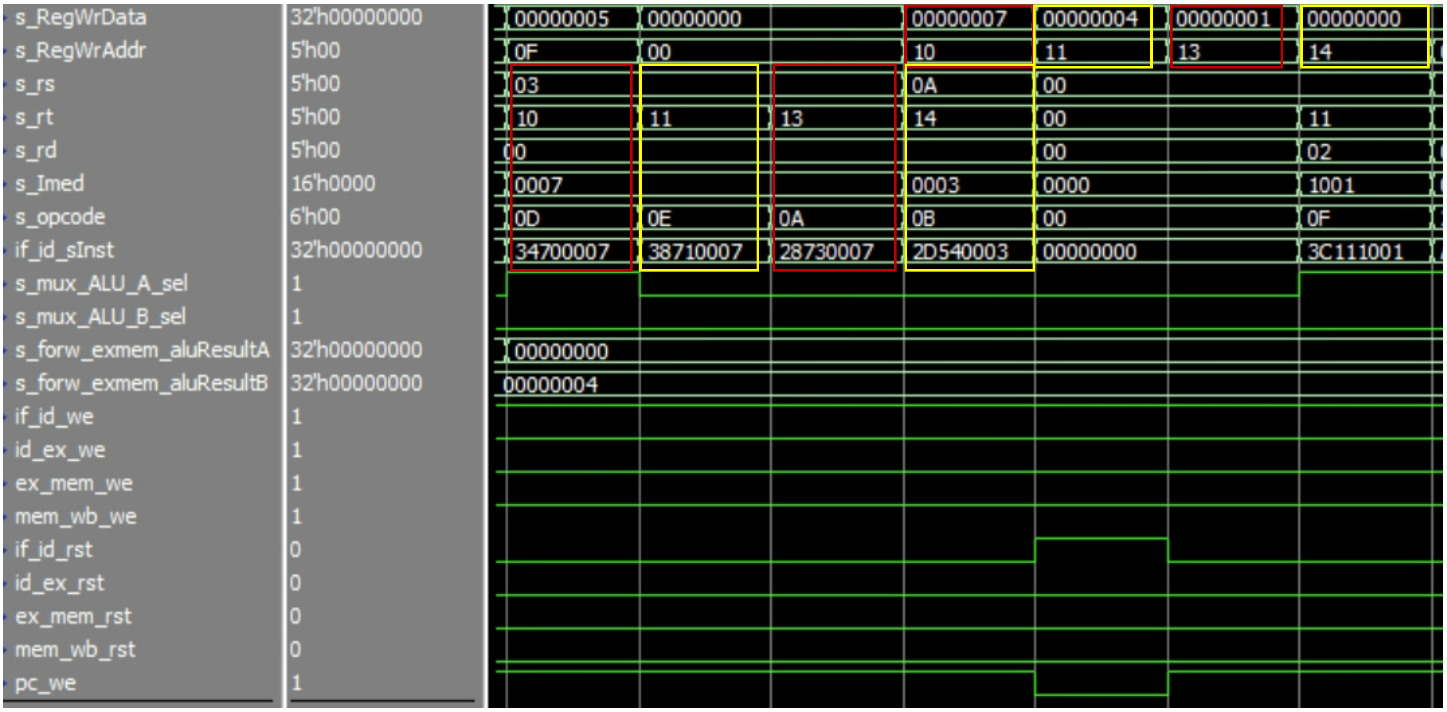
ANDI $ ORI & XORI & SLTI & SLTIU

andi $15, $5, (RED) 7 #Place 5 in $15

ori $16, $3, 7 (YELLOW) #Place 7 in $16

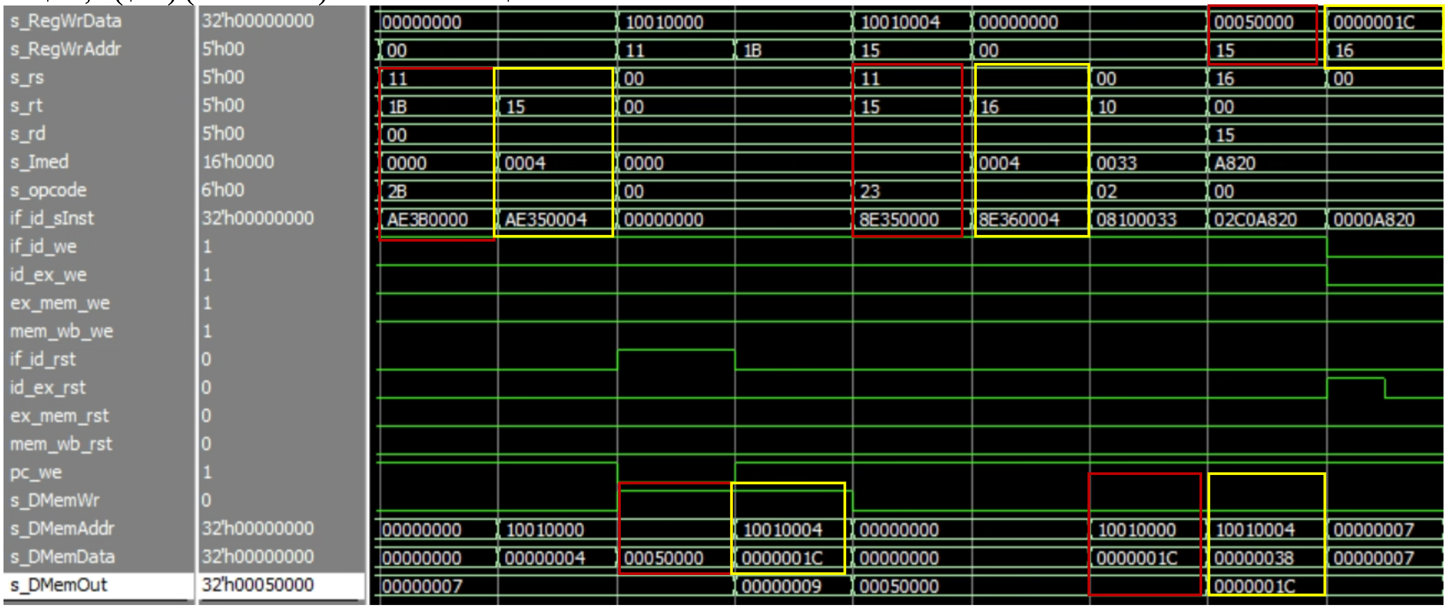
xori $17, $3, 7 (RED) #Place 4 in $17

slti $19, $3, 7 (YELLOW) #Place 1 in $19


s_RegWrData 32'h00000000 00000005 00000000 00000007 00000004 00000001 00000000
s_RegWrAddr 5'h00 0F 00 10 11 13 14
s_rs 5'h00 03 0A 00
s_rt 5'h00 10 11 13 14 00 11
s_rd 5'h00 00 00 02
s_Imed 16'h0000 0007 0003 0000 1001
s_opcode 6'h00 0D 0E 0A 0B 00 0F
if_id_sInst 32'h00000000 34700007 38710007 28730007 2D540003 00000000 3C111001
s_mux_ALU_A_sel 1
s_mux_ALU_B_sel 1
s_forw_exmem_aluResultA 32'h00000000 00000000
s_forw_exmem_aluResultB 32'h00000000 00000004
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1

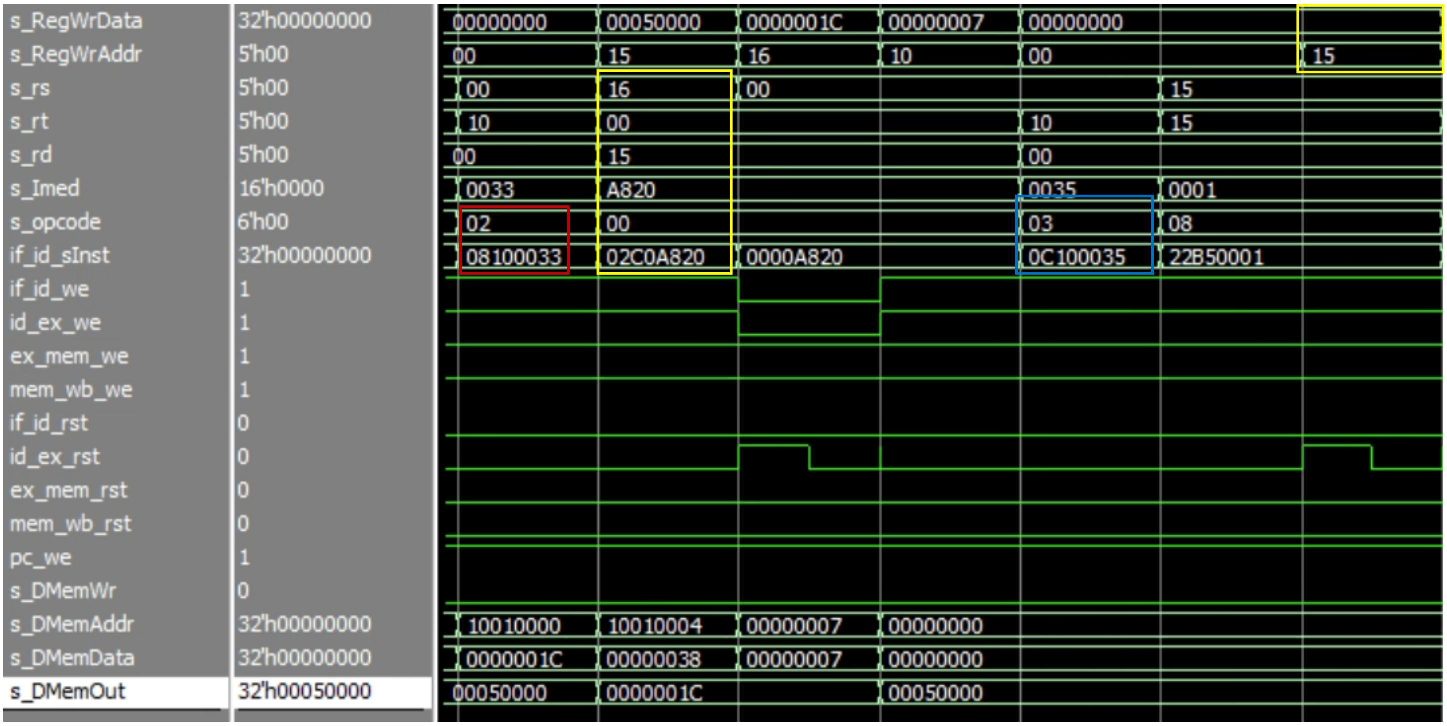
LW & SW

sw $27, 0($17) (RED) #Place x00050000 in address 4
sw $21, 4($17) (YELLOW) #Place x1c in address 8
lw $21, 0($17) (RED) #Place x00050000 in $21
lw $22, 4($17) (YELLOW) #Place x1c in $22


s_RegWrData 32'h00000000 00000000 10010000 10010004 00000000 00050000 0000001C
s_RegWrAddr 5'h00 00 11 1B 15 00 15 16
s_rs 5'h00 11 00 11 00 16 00
s_rt 5'h00 1B 15 00 15 16 10 00
s_rd 5'h00 00 15
s_Imed 16'h0000 0000 0004 0000 0004 0033 A820
s_opcode 6'h00 2B 00 23 02 00
if_id_sInst 32'h00000000 AE3B0000 AE350004 00000000 8E350000 8E360004 08100033 02C0A820 0000A820
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1
s_DMemWr 0
s_DMemAddr 32'h00000000 00000000 10010000 10010004 00000000 10010000 10010004 00000007
s_DMemData 32'h00000000 00000000 00000004 00050000 0000001C 00000000 0000001C 00000038 00000007
s_DMemOut 32'h00050000 00000007 00000009 00050000 0000001C

Jump

j skip_add (RED)
add $21, $22, $0 #Place x1c in $21 (THIS SHOULD BE SKIPPED)
skip_add:
    add $21, $0, $0 (YELLOW) #Place x0 in $21
jal task (BLUE)


s_RegWrData 32'h00000000 00000000 00050000 0000001C 00000007 00000000
s_RegWrAddr 5'h00 00 15 16 10 00 15
s_rs 5'h00 00 16 00 15
s_rt 5'h00 10 00 10 15
s_rd 5'h00 00 15 00
s_Imed 16'h0000 0033 A820 0035 0001
s_opcode 6'h00 02 00 03 08
if_id_sInst 32'h00000000 08100033 02C0A820 0000A820 0C100035 22B50001
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1
s_DMemWr 0
s_DMemAddr 32'h00000000 10010000 10010004 00000007 00000000
s_DMemData 32'h00000000 0000001C 00000038 00000007 00000000
s_DMemOut 32'h00050000 00050000 0000001C 00050000

JAL & JR

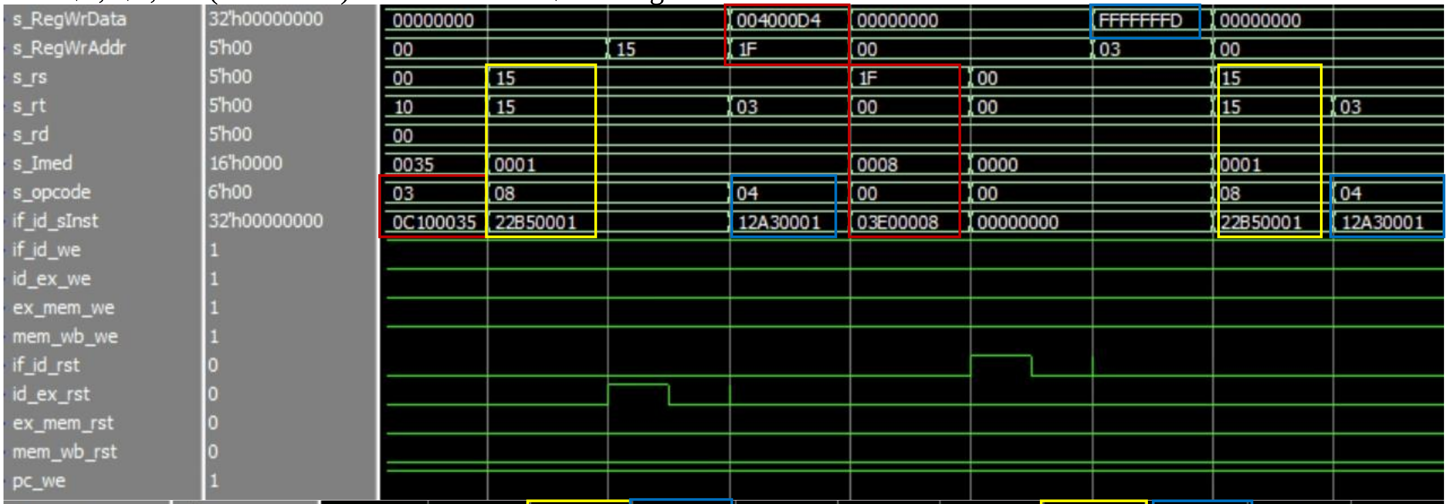
jal task (RED)

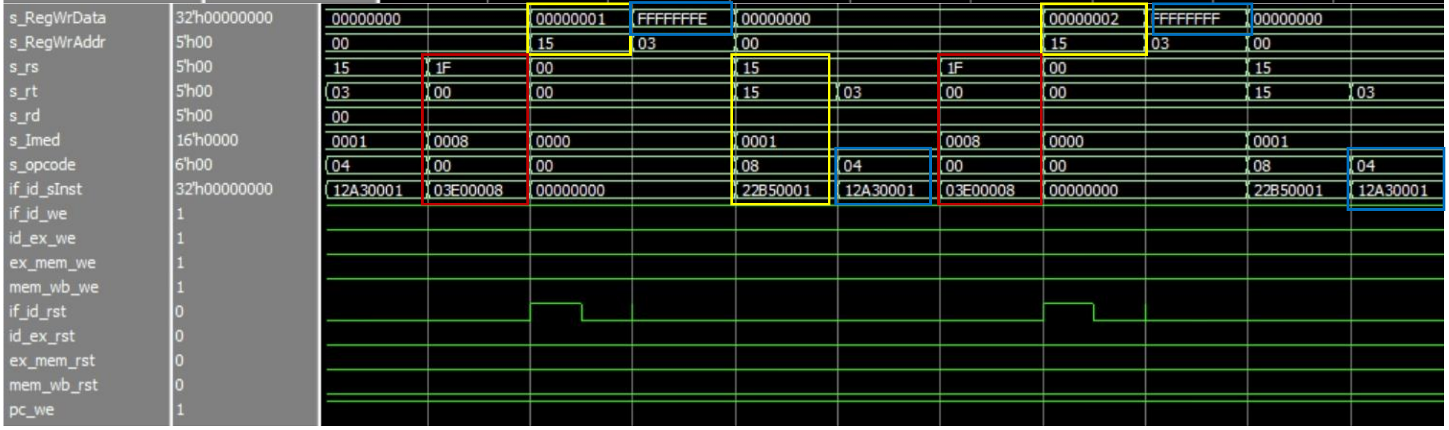
task: #Loops until $21 equals 3

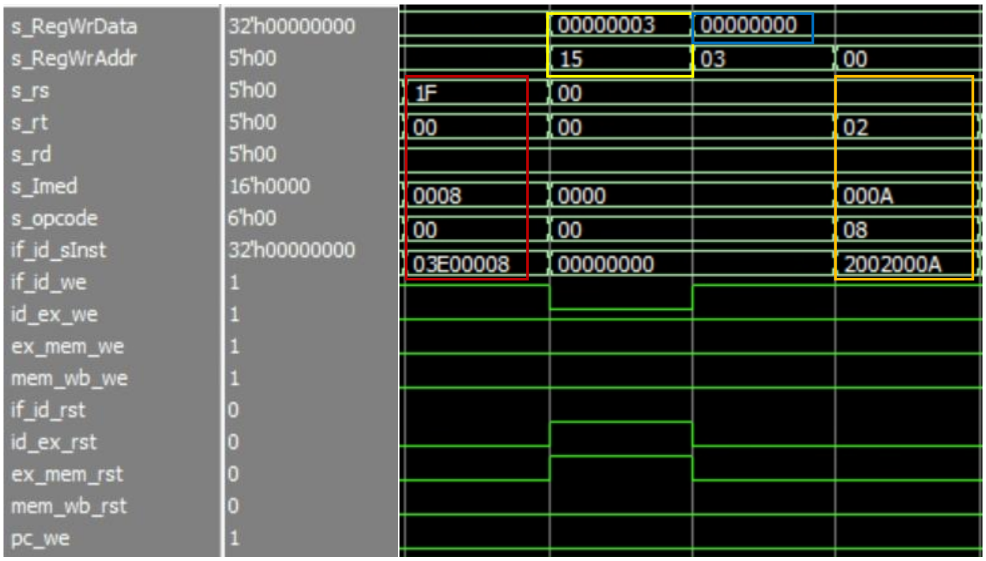
    add $21, $21, 1 (YELLOW) #increment $21 by 1 three times
    beq $21, $3, exit_task (BLUE)
    jr $ra (RED)

exit_task:

addi $2, $0, 10 (ORANGE) # Place "10" in $v0 to signal an "exit" or "halt"

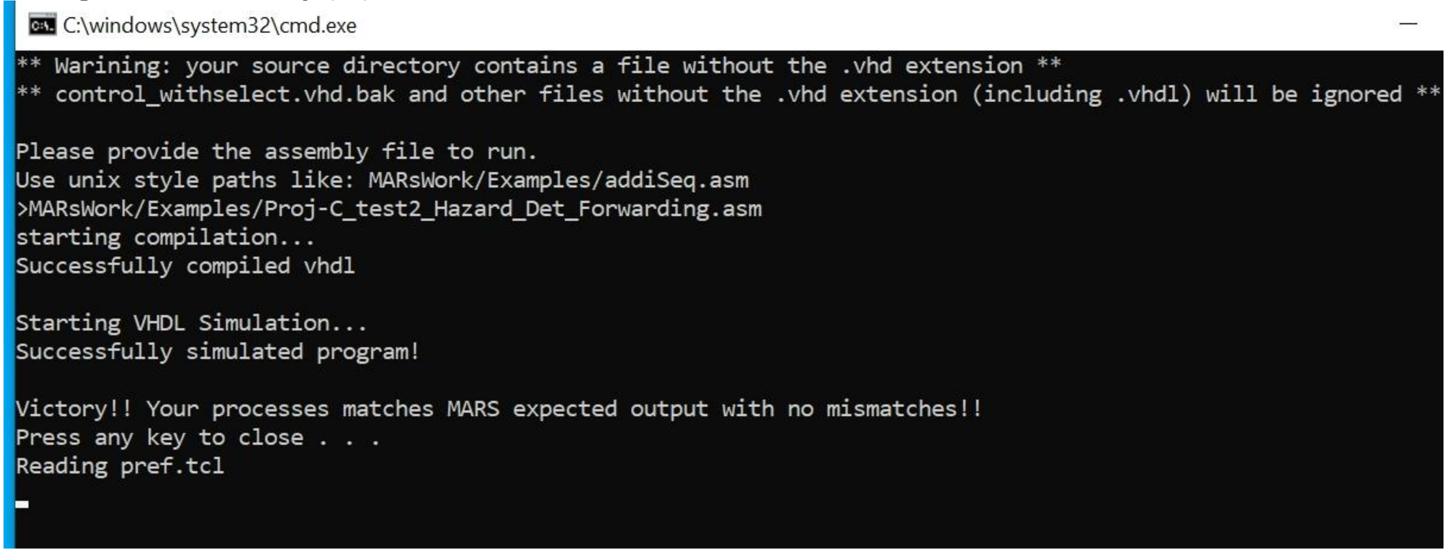

s_RegWrData 32'h00000000 00000000 004000D4 00000000 FFFFFFFD 00000000
s_RegWrAddr 5'h00 00 15 1F 00 03 00
s_rs 5'h00 00 15 1F 00 15
s_rt 5'h00 10 15 03 00 00 15 03
s_rd 5'h00 00
s_Imed 16'h0000 0035 0001 0008 0000 0001
s_opcode 6'h00 03 08 04 00 00 08 04
if_id_sInst 32'h00000000 0C100035 22B50001 12A30001 03E00008 00000000 22B50001 12A30001
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1


s_RegWrData 32'h00000000 00000000 00000001 FFFFFFFE 00000000 00000002 FFFFFFFF 00000000
s_RegWrAddr 5'h00 00 15 03 00 15 03 00
s_rs 5'h00 15 1F 00 15 1F 00 15
s_rt 5'h00 03 00 00 15 03 00 00 15 03
s_rd 5'h00 00
s_Imed 16'h0000 0001 0008 0000 0001 0008 0000 0001
s_opcode 6'h00 04 00 00 08 04 00 00 08 04
if_id_sInst 32'h00000000 12A30001 03E00008 00000000 22B50001 12A30001 03E00008 00000000 22B50001 12A30001
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1

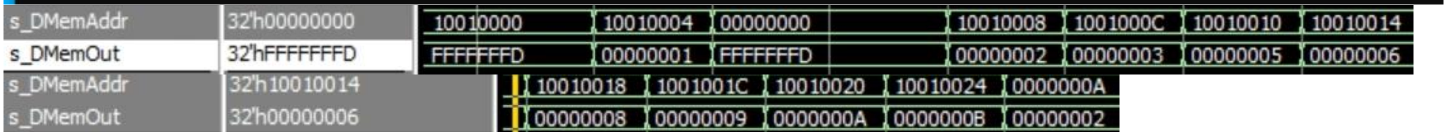


BUBBLE SORT without nops

Bubble sort was reintroduced without the majority of the nops except for two nops for the data dependencies pertaining to loading from data memory (lw).



```
** Warining: your source directory contains a file without the .vhd extension **
** control_withselect.vhd.bak and other files without the .vhd extension (including .vhdl) will be ignored **

Please provide the assembly file to run.
Use unix style paths like: MARsWork/Examples/addiSeq.asm
>MARsWork/Examples/Proj-C_test2_Hazard_Det_Forwarding.asm
starting compilation...
Successfully compiled vhdl

Starting VHDL Simulation...
Successfully simulated program!

Victory!! Your processes matches MARS expected output with no mismatches!!
Press any key to close . . .
Reading pref.tcl
```



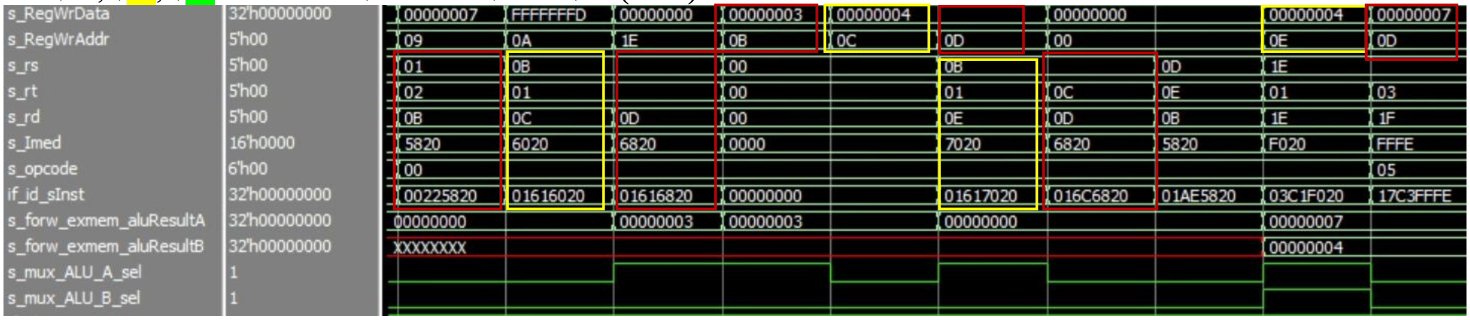
From the waveform, the stored values are still in the correct order of addresses.

Address	Value
X10010000	xFFFFFFFD
X10010004	x00000001
X10010008	x00000002
X1001000c	x00000003
X10010010	x00000005
X10010014	x00000006
X10010018	x00000008
X1001001c	x00000009
X10010020	x0000000A
X10010024	x0000000B

FORWARD DEPENDENCIES

add $11, $1, $2 #Place 3 in $11 (RED)
add $12, $11, $1 #Place 4 in $12 RAW $11 (YELLOW)
add $13, $11, $1 #Place 4 in $13 RAW $11 (RED)

add $14, $11, $1 #Place 4 in $14 RAW $11 (YELLOW)
add $13, $11, $12 #Place 7 in $13 RAW $11 & $12 (RED)

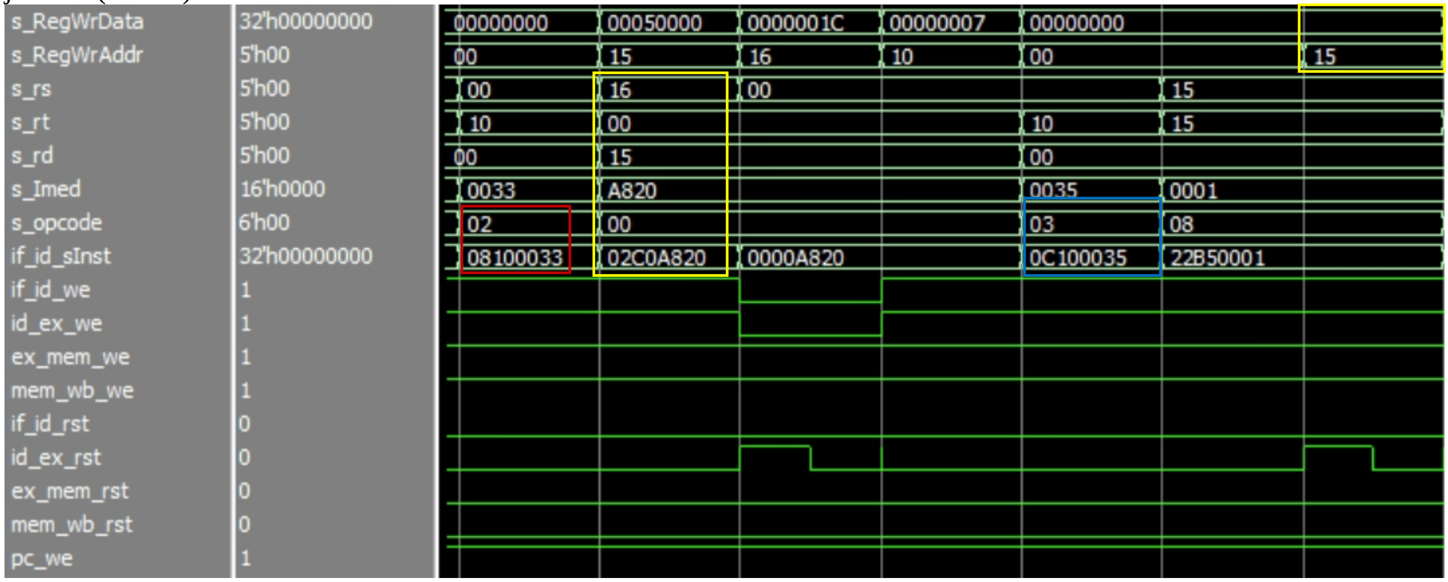

s_RegWrData 32'h00000000 00000007 FFFFFFFD 00000000 00000003 00000004 00000000 00000004 00000007
s_RegWrAddr 5'h00 09 0A 1E 0B 0C 0D 00 0E 0D
s_rs 5'h00 01 0B 00 0B 0D 1E
s_rt 5'h00 02 01 00 01 0C 0E 01 03
s_rd 5'h00 0B 0C 0D 00 0E 0D 0B 1E 1F
s_Imed 16'h0000 5820 6020 6820 0000 7020 6820 5820 F020 FFFE
s_opcode 6'h00 00 05
if_id_sInst 32'h00000000 00225820 01616020 01616820 00000000 01617020 016C6820 01AE5820 03C1F020 17C3FFFE
s_forw_exmem_aluResultA 32'h00000000 00000000 00000003 00000003 00000000 00000007
s_forw_exmem_aluResultB 32'h00000000 XXXXXXXX 00000004
s_mux_ALU_A_sel 1
s_mux_ALU_B_sel 1

From the wave form, you can see that the register values are being forwarded through the mux signals connected to the ALU when there is a read after write.

HAZARD DETECTION

Jump & JAL

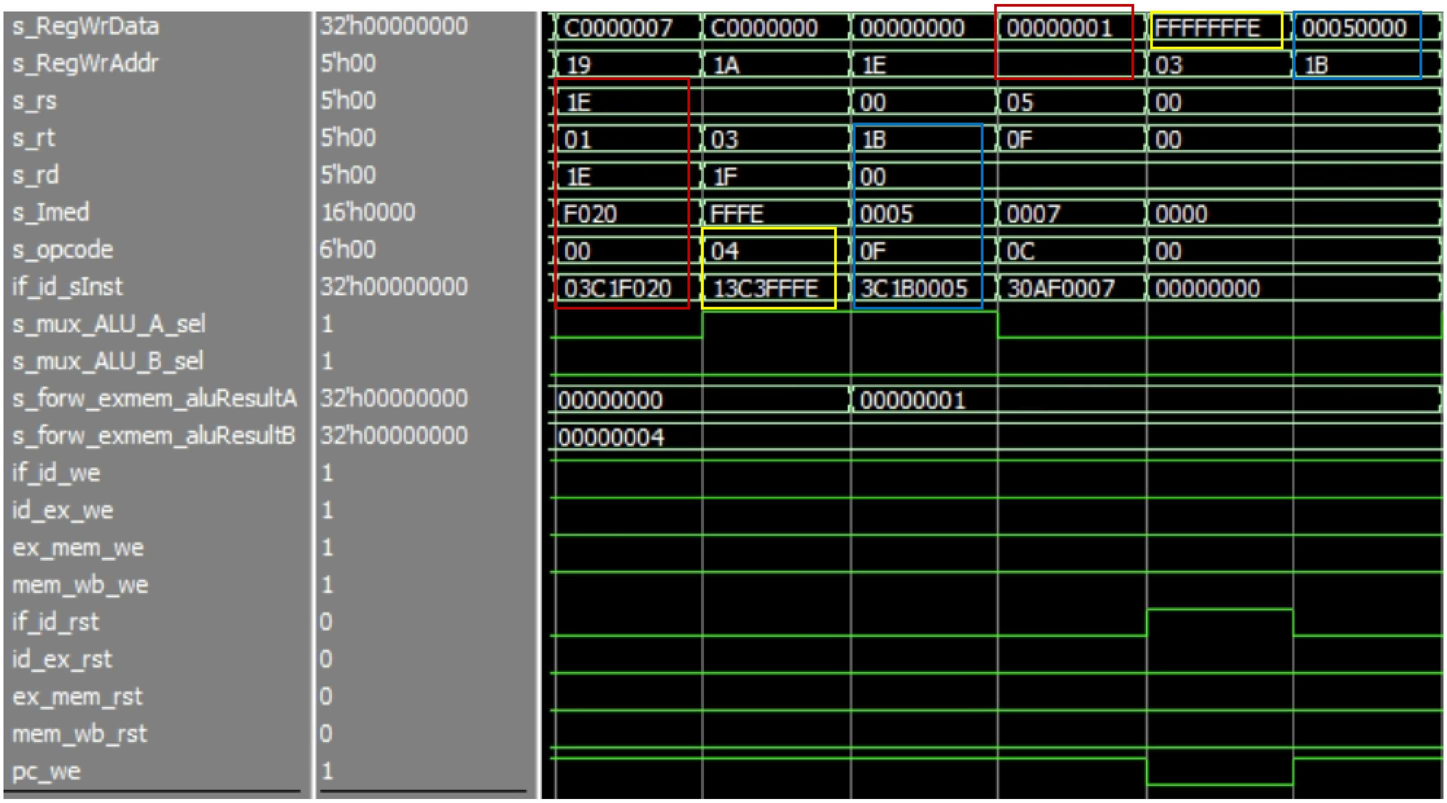
j skip_add (RED)
add $21, $22, $0 #Place x1c in $21 (THIS SHOULD BE SKIPPED)
skip_add:
add $21, $0, $0 (YELLOW) #Place x0 in $21
jal task (BLUE)


s_RegWrData 32'h00000000 00000000 00050000 0000001C 00000007 00000000
s_RegWrAddr 5'h00 00 15 16 10 00 15
s_rs 5'h00 00 16 00 15
s_rt 5'h00 10 00 10 15
s_rd 5'h00 00 15 00
s_Imed 16'h0000 0033 A820 0035 0001
s_opcode 6'h00 02 00 03 08
if_id_sInst 32'h00000000 08100033 02C0A820 0000A820 0C100035 22B50001
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1

As you can see, based off the waveform, when a Jump instruction is called, the ID/EX register is flushed to erase the bogus instruction.

Branch

Looper_2:
add $30, $30, $1 (RED) #Add 1 to $20 and branch if $30 == 3
beq $30, $3, Looper_2 (YELLOW) #$30 = 1so will continue
lui $27, 5 (BLUE) #Place x00050000 in $27


s_RegWrData 32'h00000000 C0000007 C0000000 00000000 00000001 FFFFFFFE 00050000
s_RegWrAddr 5'h00 19 1A 1E 03 1B
s_rs 5'h00 1E 00 05 00
s_rt 5'h00 01 03 1B 0F 00
s_rd 5'h00 1E 1F 00
s_Imed 16'h0000 F020 FFFE 0005 0007 0000
s_opcode 6'h00 00 04 0F 0C 00
if_id_sInst 32'h00000000 03C1F020 13C3FFFE 3C1B0005 30AF0007 00000000
s_mux_ALU_A_sel 1
s_mux_ALU_B_sel 1
s_forw_exmem_aluResultA 32'h00000000 00000000 00000001
s_forw_exmem_aluResultB 32'h00000000 00000004
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1

As you can see the IF/ID register is flushed and the PC is stalled when a branch instruction is sent from the control unit to the hazard control unit.

JR

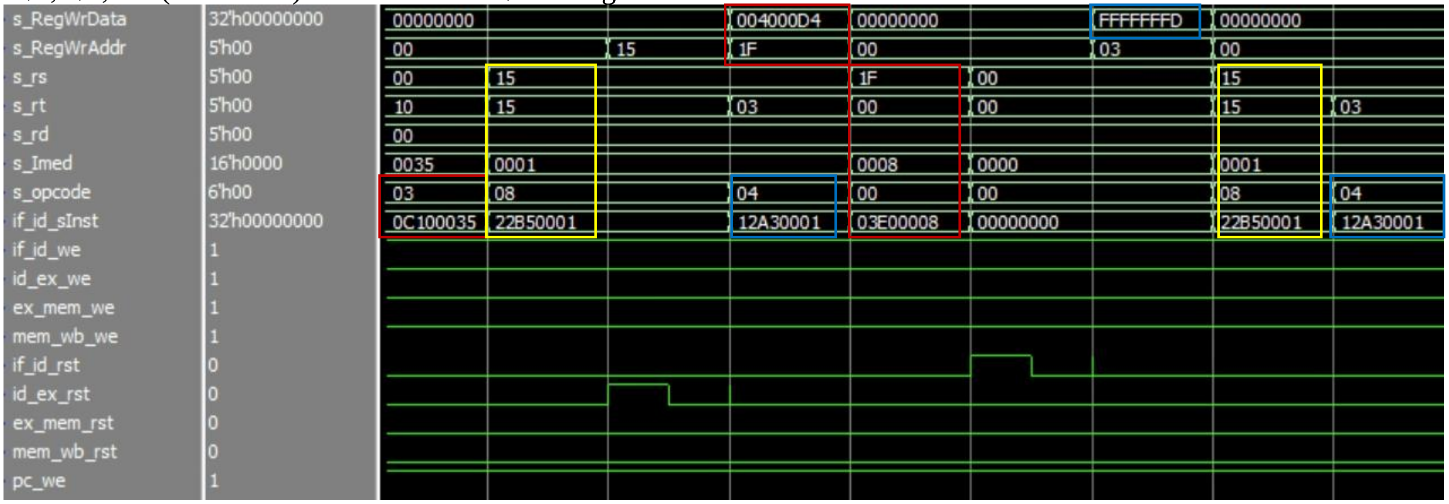
jal task (RED)

task: #Loops until $21 equals 3

    add $21, $21, 1 (YELLOW) #increment $21 by 1 three times
    beq $21, $3, exit_task (BLUE)
    jr $ra (RED)

exit_task:

addi $2, $0, 10 (ORANGE) # Place "10" in $v0 to signal an "exit" or "halt"


s_RegWrData 32'h00000000 00000000 004000D4 00000000 FFFFFFFD 00000000
s_RegWrAddr 5'h00 00 15 1F 00 03 00
s_rs 5'h00 00 15 1F 00 15
s_rt 5'h00 10 15 03 00 00 15 03
s_rd 5'h00 00
s_Imed 16'h0000 0035 0001 0008 0000 0001
s_opcode 6'h00 03 08 04 00 00 08 04
if_id_sInst 32'h00000000 0C100035 22B50001 12A30001 03E00008 00000000 22B50001 12A30001
if_id_we 1
id_ex_we 1
ex_mem_we 1
mem_wb_we 1
if_id_rst 0
id_ex_rst 0
ex_mem_rst 0
mem_wb_rst 0
pc_we 1

From the waveform, after the jr is called the IF/ID register is flushed on the following clock cycle to erase the bogus instruction.

Timing Report of Synthesis with data dependency handling.

The maximum frequency of the updated processor is 35.46MHz. The critical path is 31.141ns and follows: ID/EX register to the forwarding unit, forwarding unit to the Mux_B_forwarding unit, from Mux B to ALU, from the ALU to the hazard detection unit, and finally from hazard detection to the IF/ID register.

To improve the frequency further, I would need to redesign the hazard detection unit to use less resets because this only adds to the overall CPI and would also reduce the critical path length in the processor.