Project 1 - Milestone 2
Single Cycle Implementation of RISC-V

CSCE 3301 Computer Architecture Fall 2025

Submitted to Dr. Cherif Salama

Done by Nour Waleed, Dalia Eisa, Omar Zainelabideen

Table of Contents

Introduction:

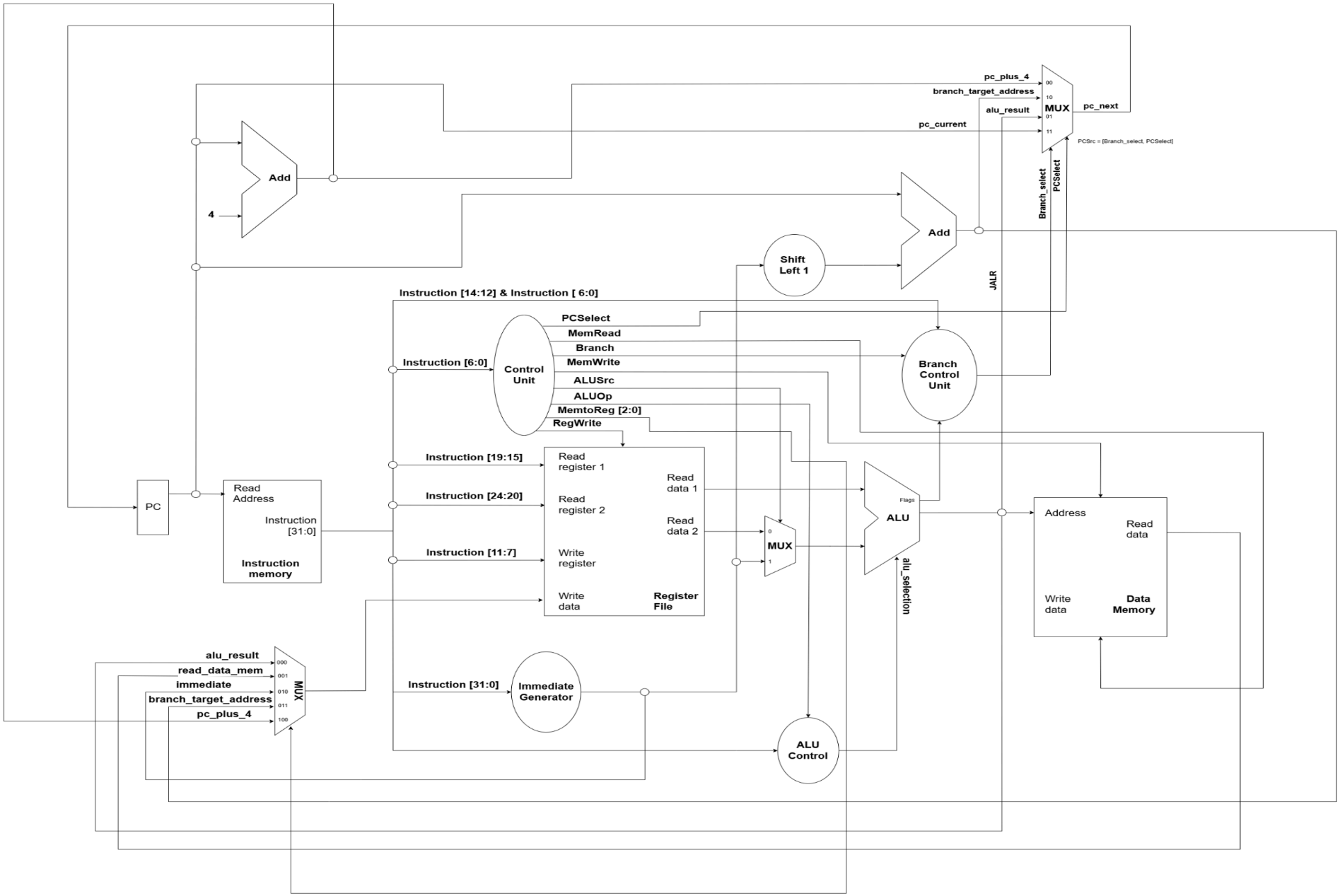
The second milestone of this project required that we implement a single cycle Risc-V architecture Central Processing Unit in Verilog. We were also required to use 2 separate memories for instructions and memory. We were successful in implementing the required deliverables, as well as testing all instructions.

Design Objectives:

- Implement a working single-cycle RISC-V CPU supporting core instructions:
    - Arithmetic/logic (ADD, SUB, AND, OR, XOR, SLT, etc.)
    - Immediate operations (ADDI, ANDI, ORI, etc.)
    - Memory operations (LW, SW, LB, LBU, LH, LHU, SB, SH)
    - Branches (BEQ, BNE, BLT, etc.)
    - Jump instructions (JAL, JALR)
- Ensure modularity (separate ALU, Control, ImmGen, RegFile, PC, memories).

Datapath:

The following block diagram describes the Datapath of our implementation.


pc_plus_4
branch_target_address
alu_result
pc_current
MUX
00
10
01
11
pc_next
PCSrc = [Branch_select, PCSelect]
Branch_select
PCSelect
Add
4
Add
Shift Left 1
JALR
Instruction [14:12] & Instruction [6:0]
PCSelect
MemRead
Branch
MemWrite
ALUSrc
ALUOp
MemtoReg [2:0]
RegWrite
Instruction [6:0]
Control Unit
Branch Control Unit
Instruction [19:15]
Instruction [24:20]
Instruction [11:7]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Register File
PC
Read Address
Instruction [31:0]
Instruction memory
MUX
0
1
Flags
ALU
alu_selection
Address
Read data
Write data
Data Memory
alu_result
read_data_mem
immediate
branch_target_address
pc_plus_4
000
001
010
011
100
MUX
Instruction [31:0]
Immediate Generator
ALU Control

- Modules used:
    - Program Counter (PC)
    - Instruction Memory (InstMem): Byte-addressable, 4 KB, little-endian.
    - Register File
    - Immediate Generator
    - ALU and ALU Control
    - Main Control Unit
    - Data Memory: Byte-addressable (for simplicity, 4 KB)
    - MUXes

Control Unit Design

Instruction Type	RegWrite	MemRead	MemWrite	ALUSrc	ALUOp	MemToReg	Branch	PCSelect
R-type	1	0	0	0	10	0	0	0
I-type	1	0	0	1	10	0	0	0
LW	1	1	0	1	00	1	0	0
SW	0	0	1	1	00	X	0	0
BEQ	0	0	0	0	01	X	1	0
JAL	1	0	0	X	11	0	0	1

Implementation:

Our implementation in Verilog consisted of many files created during the lab sessions, as well as we added other files.

The following is a brief description of what is contained in each of the submitted files:

ALU_Control_Unit.v : Decodes ALUOp and instruction fields to drive the ALU (4-bit ALU_selection) and sets byte_select for LB/LH/LW/LBU/LHU; covers R/I/LUI/AUIPC/JAL/JALR cases.

n_bit_ALU.v : Parameterized ALU implementing ADD/SUB/AND/OR/XOR, SLL/SRL/SRA, SLT/SLTU and producing zero, carry, overflow, sign flags.

Branch_Control_Unit.v : Branch decision logic: looks at opcode/funct3 and ALU flags to assert take_branch for BEQ/BNE/BLT/BGE/BLTU/BGEU; also flags JAL, FENCE/PAUSE, ECALL/EBREAK.

Control_Unit.v : Top-level instruction decoder generating control signals where Branch, MemRead, MemWrite, ALUSrc, RegWrite, PCSelect, MemtoReg, and ALUOp.

Immediate_Generator.v : Builds 32-bit immediates for I/S/B/U/J types with correct sign/zero extension.

InstMem.v : 4 KB byte-addressable instruction memory with combinational read; includes example initialization for testing.

Data_Mem.v : 4 KB byte-addressable data memory: asynchronous read when MemRead=1 with byte_select handling (LB/LH/LW/LBU/LHU) and synchronous writes for SB/SH/SW.

D_FlipFlop.v : Positive-edge-triggered D flip-flop with asynchronous reset.

FA.v : 1-bit full adder primitive.

n_bit_mux2x1.v : Parameterized 2:1 multiplexer selecting between two n-bit inputs by a 1-bit select.

n_bit_mux4x1.v : Parameterized 4:1 multiplexer with a 2-bit select.

n_bit_mux5x1.v : Parameterized 5:1 multiplexer with a 3-bit select.

n_bit_register.v : Parameterized n-bit clocked register for state like PC or pipeline regs.

n_bit_shift_left.v : Logical left-shift of an n-bit bus for branch/jump immediate alignment.

RCA.v : Ripple-carry adder (n-bit) built from full adders; outputs sum and carry-out.

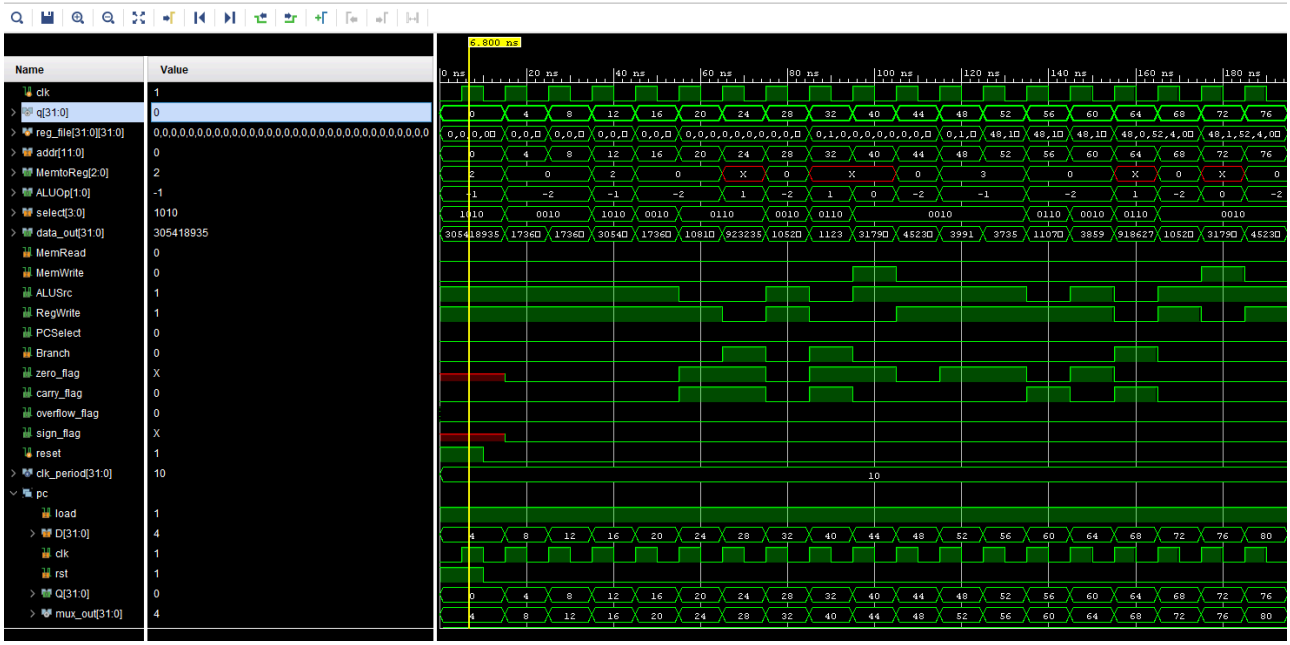
register_file.v : RISC-V 32×32 register file: two combinational read ports, one synchronous write; x0 hard-wired to zero.

Single_Cycle_Processor.v : Top-level single-cycle RISC-V CPU tying together control, ALU, regfile, immediates, branch/jump, and memories.

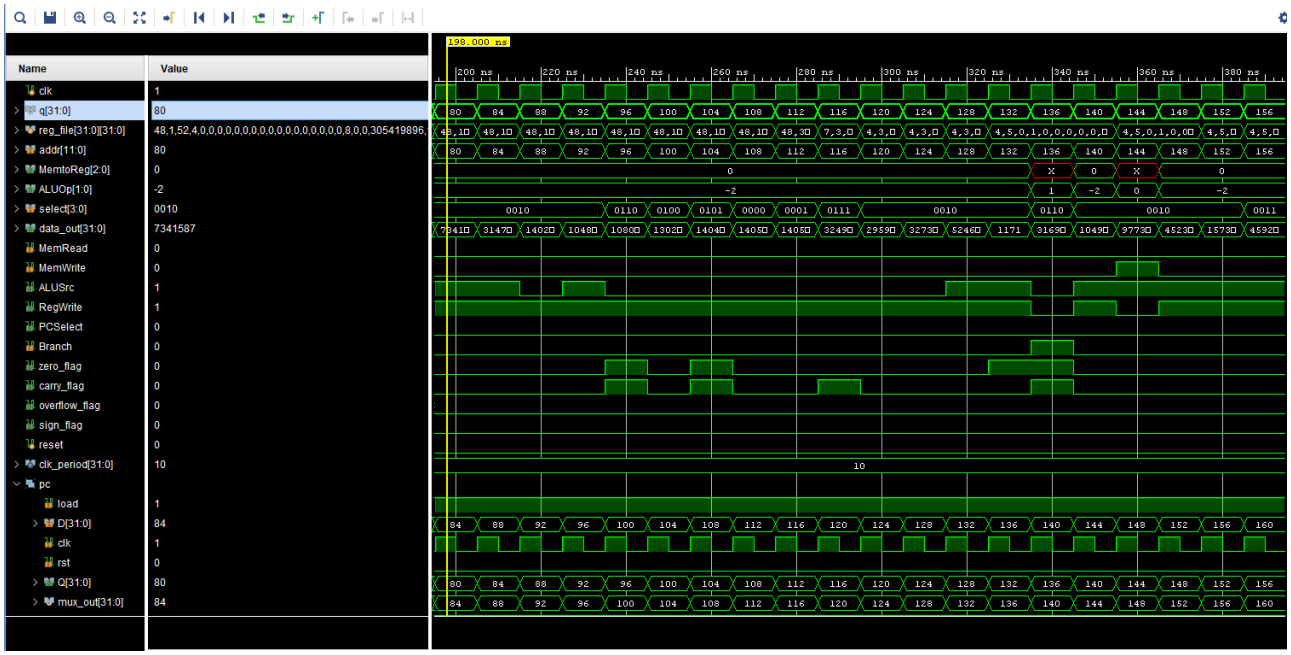
Waveforms of all test cases:

All tests passed successfully.

```
mem[0]=8'hB7; mem[1]=8'h52; mem[2]=8'h34; mem[3]=8'h12; // lui     x5, 0x12345
mem[4]=8'h93; mem[5]=8'h82; mem[6]=8'h82; mem[7]=8'h67; // addi    x5,  x5,  0x678
mem[8]=8'h13; mem[9]=8'h03; mem[10]=8'h80; mem[11]=8'h67; // addi    x6,  x0,  0x678
mem[12]=8'hB7; mem[13]=8'h53; mem[14]=8'h34; mem[15]=8'h12; // lui     x7,  0x12345
mem[16]=8'h93; mem[17]=8'h83; mem[18]=8'h83; mem[19]=8'h67; // addi    x7,  x7,  0x678
mem[20]=8'h33; mem[21]=8'h8E; mem[22]=8'h72; mem[23]=8'h40; // sub     x28, x5,  x7
mem[24]=8'h63; mem[25]=8'h16; mem[26]=8'h0E; mem[27]=8'h00; // bne     x28, x0,  L_LUI_FAIL
mem[28]=8'h13; mem[29]=8'h0F; mem[30]=8'h10; mem[31]=8'h00; // addi    x30, x0,  1
mem[32]=8'h63; mem[33]=8'h04; mem[34]=8'h00; mem[35]=8'h00; // beq     x0,  x0,  L_LUI_STORE
        // L_LUI_FAIL:
mem[36]=8'h13; mem[37]=8'h0F; mem[38]=8'h00; mem[39]=8'h00; // addi    x30, x0,  0
    // L_LUI_STORE:
mem[40]=8'h23; mem[41]=8'h20; mem[42]=8'hE5; mem[43]=8'h01; // sw      x30, 0(x10)
mem[44]=8'h13; mem[45]=8'h05; mem[46]=8'h45; mem[47]=8'h00; // addi    x10, x10, 4
mem[48]=8'h97; mem[49]=8'h0F; mem[50]=8'h00; mem[51]=8'h00; // auipc   x31, 0
mem[52]=8'h97; mem[53]=8'h0E; mem[54]=8'h00; mem[55]=8'h00; // auipc   x29, 0
mem[56]=8'h33; mem[57]=8'h8E; mem[58]=8'hFE; mem[59]=8'h41; // sub     x28, x29, x31
mem[60]=8'h13; mem[61]=8'h0F; mem[62]=8'h00; mem[63]=8'h00; // addi    x30, x0,  0
mem[64]=8'h63; mem[65]=8'h04; mem[66]=8'h0E; mem[67]=8'h00; // beq     x28, x0,  L_AUIPC_STORE
mem[68]=8'h13; mem[69]=8'h0F; mem[70]=8'h10; mem[71]=8'h00; // addi    x30, x0,  1
    // L_AUIPC_STORE:
mem[72]=8'h23; mem[73]=8'h20; mem[74]=8'hE5; mem[75]=8'h01; // sw      x30, 0(x10)
mem[76]=8'h13; mem[77]=8'h05; mem[78]=8'h45; mem[79]=8'h00; // addi    x10, x10, 4
```

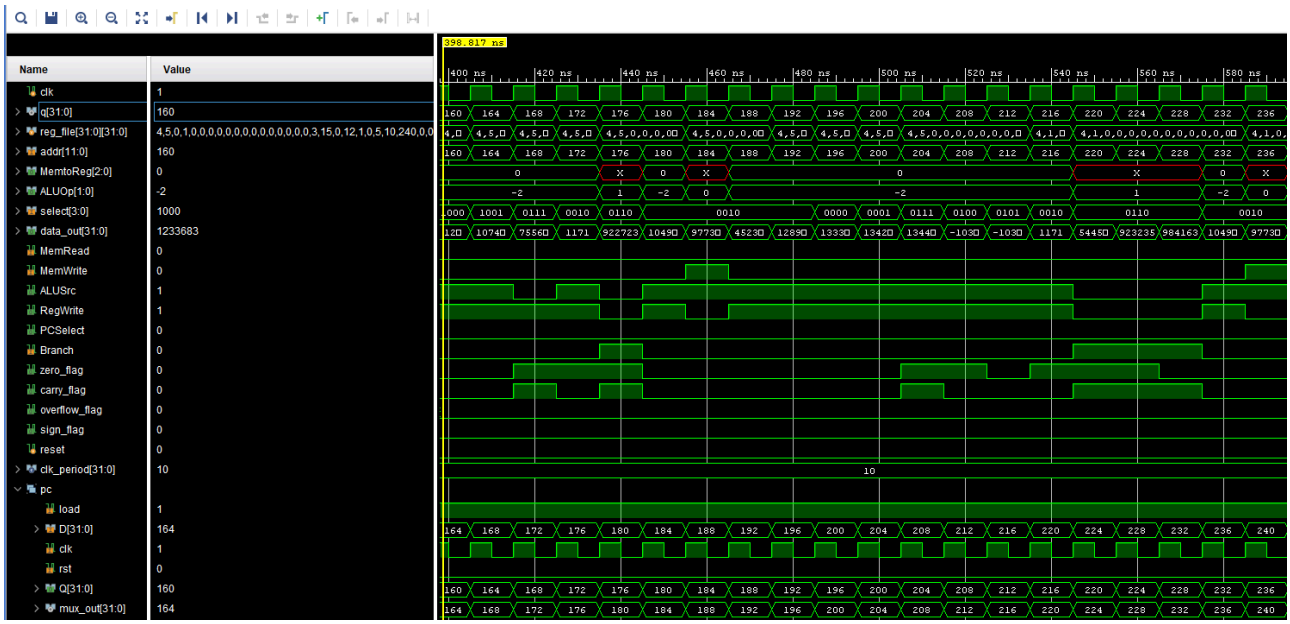


```
mem[80]=8'h13; mem[81]=8'h06; mem[82]=8'h70; mem[83]=8'h00; // addi    x12, x0,  7
mem[84]=8'h93; mem[85]=8'h06; mem[86]=8'h30; mem[87]=8'h00; // addi    x13, x0,  3
mem[88]=8'hB3; mem[89]=8'h02; mem[90]=8'hD6; mem[91]=8'h00; // add     x5,  x12, x13
mem[92]=8'h13; mem[93]=8'h03; mem[94]=8'hA0; mem[95]=8'h00; // addi    x6,  x0,  10
mem[96]=8'hB3; mem[97]=8'h83; mem[98]=8'h62; mem[99]=8'h40; // sub     x7,  x5,  x6
mem[100]=8'h33; mem[101]=8'hAE; mem[102]=8'hC6; mem[103]=8'h00; // slt     x28, x13, x12
mem[104]=8'hB3; mem[105]=8'h3E; mem[106]=8'hD6; mem[107]=8'h00; // sltu    x29, x12, x13
mem[108]=8'h33; mem[109]=8'h7F; mem[110]=8'hD6; mem[111]=8'h00; // and     x30, x12, x13
mem[112]=8'hB3; mem[113]=8'h6F; mem[114]=8'hD6; mem[115]=8'h00; // or      x31, x12, x13
mem[116]=8'hB3; mem[117]=8'hCF; mem[118]=8'hEF; mem[119]=8'h01; // xor     x31, x31, x30
mem[120]=8'hB3; mem[121]=8'h83; mem[122]=8'hC3; mem[123]=8'h01; // add     x7,  x7,  x28
mem[124]=8'hB3; mem[125]=8'h83; mem[126]=8'hF3; mem[127]=8'h01; // add     x7,  x7,  x31
mem[128]=8'h13; mem[129]=8'h0F; mem[130]=8'h50; mem[131]=8'h00; // addi    x30, x0,  5
mem[132]=8'h93; mem[133]=8'h04; mem[134]=8'h00; mem[135]=8'h00; // addi    x9,  x0,  0
mem[136]=8'h63; mem[137]=8'h94; mem[138]=8'hE3; mem[139]=8'h01; // bne     x7,  x30, L_ALU_FAIL
mem[140]=8'h93; mem[141]=8'h04; mem[142]=8'h10; mem[143]=8'h00; // addi    x9,  x0,  1
      // L_ALU_FAIL:
mem[144]=8'h23; mem[145]=8'h20; mem[146]=8'h95; mem[147]=8'h00; // sw      x9,  0(x10)
mem[148]=8'h13; mem[149]=8'h05; mem[150]=8'h45; mem[151]=8'h00; // addi    x10, x10, 4
mem[152]=8'h13; mem[153]=8'h06; mem[154]=8'hF0; mem[155]=8'h00; // addi    x12, x0,  15
mem[156]=8'h93; mem[157]=8'h12; mem[158]=8'h46; mem[159]=8'h00; // slli    x5,  x12, 4
```



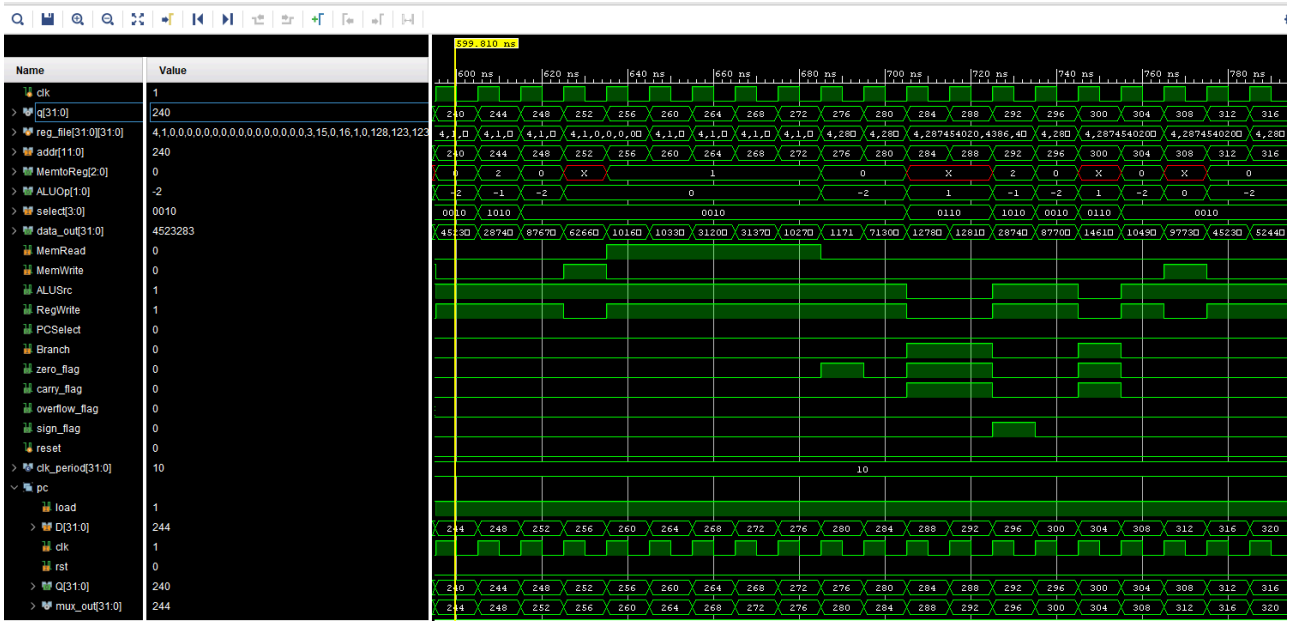

```
mem[160]=8'h13; mem[161]=8'hD3; mem[162]=8'h12; mem[163]=8'h00; // srli    x6,  x5,  1
mem[164]=8'h93; mem[165]=8'hD3; mem[166]=8'h12; mem[167]=8'h40; // srai    x7,  x5,  1
mem[168]=8'h33; mem[169]=8'h4E; mem[170]=8'h73; mem[171]=8'h00; // xor     x28, x6,  x7
mem[172]=8'h93; mem[173]=8'h04; mem[174]=8'h00; mem[175]=8'h00; // addi    x9,  x0,  0
```

```
mem[176]=8'h63; mem[177]=8'h14; mem[178]=8'h0E; mem[179]=8'h00; // bne    x28, x0,  L_SHIFT_FAIL
mem[180]=8'h93; mem[181]=8'h04; mem[182]=8'h10; mem[183]=8'h00; // addi   x9,  x0,  1
      // L_SHIFT_FAIL:
mem[184]=8'h23; mem[185]=8'h20; mem[186]=8'h95; mem[187]=8'h00; // sw     x9,  0(x10)
mem[188]=8'h13; mem[189]=8'h05; mem[190]=8'h45; mem[191]=8'h00; // addi   x10, x10, 4
mem[192]=8'h93; mem[193]=8'h02; mem[194]=8'hB0; mem[195]=8'h07; // addi   x5,  x0,  123
mem[196]=8'h13; mem[197]=8'hF3; mem[198]=8'hF2; mem[199]=8'h07; // andi   x6,  x5,  0x7F
mem[200]=8'h93; mem[201]=8'h63; mem[202]=8'h00; mem[203]=8'h08; // ori    x7,  x0,  0x80
mem[204]=8'h13; mem[205]=8'hCE; mem[206]=8'h03; mem[207]=8'h08; // xori   x28, x7,  0x80
mem[208]=8'h93; mem[209]=8'h2E; mem[210]=8'hF0; mem[211]=8'hFF; // slti   x29, x0,  -1
mem[212]=8'h13; mem[213]=8'h3F; mem[214]=8'hF0; mem[215]=8'hFF; // sltiu  x30, x0,  -1
mem[216]=8'h93; mem[217]=8'h04; mem[218]=8'h00; mem[219]=8'h00; // addi   x9,  x0,  0
mem[220]=8'h63; mem[221]=8'h18; mem[222]=8'h53; mem[223]=8'h00; // bne    x6,  x5,  L_IMM_FAIL
mem[224]=8'h63; mem[225]=8'h16; mem[226]=8'h0E; mem[227]=8'h00; // bne    x28, x0,  L_IMM_FAIL
mem[228]=8'h63; mem[229]=8'h04; mem[230]=8'h0F; mem[231]=8'h00; // beq    x30, x0,  L_IMM_FAIL
mem[232]=8'h93; mem[233]=8'h04; mem[234]=8'h10; mem[235]=8'h00; // addi   x9,  x0,  1
       // L_IMM_FAIL:
mem[236]=8'h23; mem[237]=8'h20; mem[238]=8'h95; mem[239]=8'h00; // sw     x9,  0(x10)
```




```
mem[240]=8'h13; mem[241]=8'h05; mem[242]=8'h45; mem[243]=8'h00; // addi   x10, x10, 4
mem[244]=8'hB7; mem[245]=8'h32; mem[246]=8'h22; mem[247]=8'h11; // lui    x5,  0x11223
mem[248]=8'h93; mem[249]=8'h82; mem[250]=8'h42; mem[251]=8'h34; // addi   x5,  x5,  0x344
mem[252]=8'h23; mem[253]=8'hA0; mem[254]=8'h5F; mem[255]=8'h00; // sw     x5,  0(x31)
mem[256]=8'h03; mem[257]=8'h83; mem[258]=8'h0F; mem[259]=8'h00; // lb     x6,  0(x31)
mem[260]=8'h83; mem[261]=8'hC3; mem[262]=8'h0F; mem[263]=8'h00; // lbu    x7,  0(x31)
mem[264]=8'h03; mem[265]=8'h9E; mem[266]=8'h2F; mem[267]=8'h00; // lh     x28, 2(x31)
mem[268]=8'h83; mem[269]=8'hDE; mem[270]=8'h2F; mem[271]=8'h00; // lhu    x29, 2(x31)
mem[272]=8'h03; mem[273]=8'hAF; mem[274]=8'h0F; mem[275]=8'h00; // lw     x30, 0(x31)
```

mem[276]=8'h93; mem[277]=8'h04; mem[278]=8'h00; mem[279]=8'h00; // addi x9, x0, 0
mem[280]=8'h13; mem[281]=8'h06; mem[282]=8'h40; mem[283]=8'h04; // addi x12, x0, 0x44
mem[284]=8'h63; mem[285]=8'h1C; mem[286]=8'hC3; mem[287]=8'h00; // bne   x6, x12, L_LDST_FAIL
mem[288]=8'h63; mem[289]=8'h9A; mem[290]=8'hC3; mem[291]=8'h00; // bne   x7, x12, L_LDST_FAIL
mem[292]=8'hB7; mem[293]=8'h36; mem[294]=8'h22; mem[295]=8'h11; // lui   x13, 0x11223
mem[296]=8'h93; mem[297]=8'h86; mem[298]=8'h46; mem[299]=8'h34; // addi x13, x13, 0x344
mem[300]=8'h63; mem[301]=8'h14; mem[302]=8'hDF; mem[303]=8'h00; // bne   x30, x13,
L_LDST_FAIL
mem[304]=8'h93; mem[305]=8'h04; mem[306]=8'h10; mem[307]=8'h00; // addi x9, x0, 1
        // L_LDST_FAIL:
mem[308]=8'h23; mem[309]=8'h20; mem[310]=8'h95; mem[311]=8'h00; // sw   x9, 0(x10)
mem[312]=8'h13; mem[313]=8'h05; mem[314]=8'h45; mem[315]=8'h00; // addi x10, x10, 4
mem[316]=8'h13; mem[317]=8'h06; mem[318]=8'h50; mem[319]=8'h00; // addi x12, x0, 5

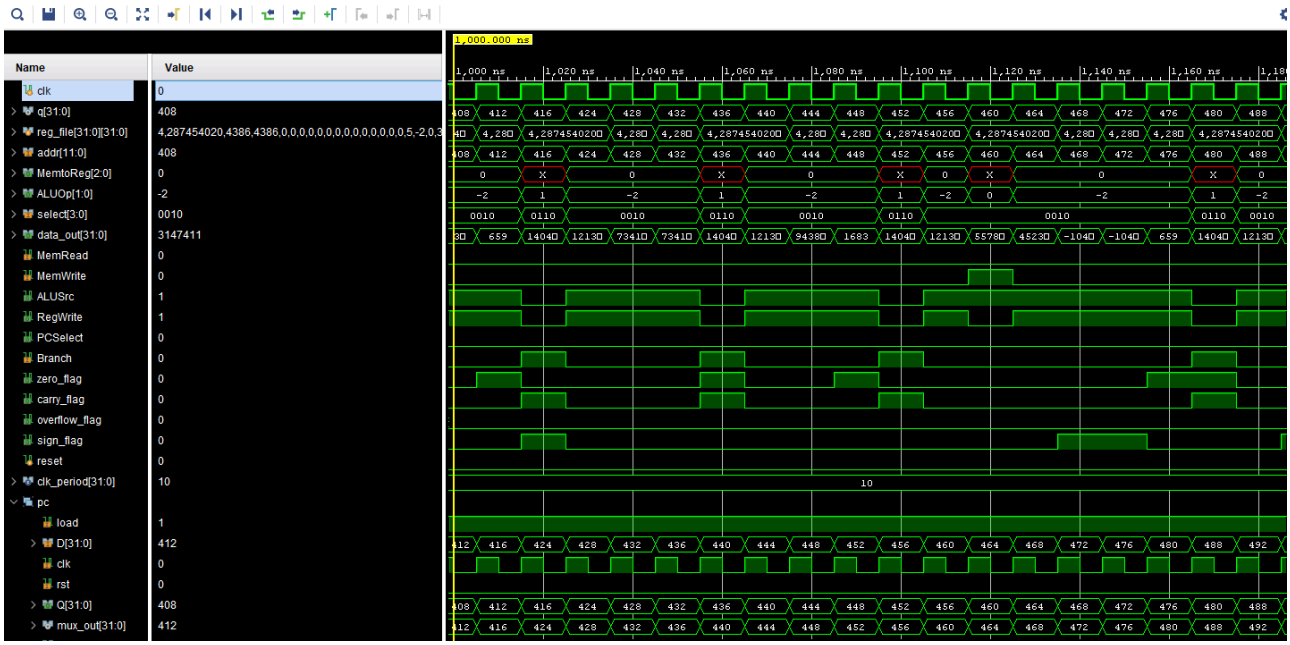


mem[320]=8'h93; mem[321]=8'h06; mem[322]=8'h50; mem[323]=8'h00; // addi x13, x0, 5
mem[324]=8'h93; mem[325]=8'h02; mem[326]=8'h00; mem[327]=8'h00; // addi x5, x0, 0
mem[328]=8'h63; mem[329]=8'h04; mem[330]=8'hD6; mem[331]=8'h00; // beq   x12, x13,
L_BEQ_TAKEN
mem[332]=8'h6F; mem[333]=8'h00; mem[334]=8'h80; mem[335]=8'h00; // jal   x0, L_BEQ_NEXT
        // L_BEQ_TAKEN:
mem[336]=8'h93; mem[337]=8'h82; mem[338]=8'h12; mem[339]=8'h00; // addi x5, x5, 1
          // L_BEQ_NEXT:
mem[340]=8'h93; mem[341]=8'h06; mem[342]=8'h60; mem[343]=8'h00; // addi x13, x0, 6
mem[344]=8'h63; mem[345]=8'h04; mem[346]=8'hD6; mem[347]=8'h00; // beq   x12, x13,
L_BEQ_NOTSHOULD
mem[348]=8'h93; mem[349]=8'h82; mem[350]=8'h12; mem[351]=8'h00; // addi x5, x5, 1
        // L_BEQ_NOTSHOULD:
mem[352]=8'h23; mem[353]=8'h20; mem[354]=8'h55; mem[355]=8'h00; // sw   x5, 0(x10)
mem[356]=8'h13; mem[357]=8'h05; mem[358]=8'h45; mem[359]=8'h00; // addi x10, x10, 4

```
mem[360]=8'h13; mem[361]=8'h06; mem[362]=8'h50; mem[363]=8'h00; // addi    x12, x0,  5
mem[364]=8'h93; mem[365]=8'h06; mem[366]=8'h60; mem[367]=8'h00; // addi    x13, x0,  6
mem[368]=8'h93; mem[369]=8'h02; mem[370]=8'h00; mem[371]=8'h00; // addi    x5,  x0,  0
mem[372]=8'h63; mem[373]=8'h14; mem[374]=8'hD6; mem[375]=8'h00; // bne     x12, x13,
L_BNE_TAKEN
mem[376]=8'h6F; mem[377]=8'h00; mem[378]=8'h80; mem[379]=8'h00; // jal     x0,  L_BNE_NEXT
        //L_BNE_TAKEN:
mem[380]=8'h93; mem[381]=8'h82; mem[382]=8'h12; mem[383]=8'h00; // addi    x5,  x5,  1
        // L_BNE_NEXT:
mem[384]=8'h93; mem[385]=8'h06; mem[386]=8'h50; mem[387]=8'h00; // addi    x13, x0,  5
mem[388]=8'h63; mem[389]=8'h14; mem[390]=8'hD6; mem[391]=8'h00; // bne     x12, x13,
L_BNE_NOTSHOULD
mem[392]=8'h93; mem[393]=8'h82; mem[394]=8'h12; mem[395]=8'h00; // addi    x5,  x5,  1
        // L_BNE_NOTSHOULD:
mem[396]=8'h23; mem[397]=8'h20; mem[398]=8'h55; mem[399]=8'h00; // sw      x5,  0(x10)
mem[400]=8'h13; mem[401]=8'h05; mem[402]=8'h45; mem[403]=8'h00; // addi    x10, x10, 4
mem[404]=8'h13; mem[405]=8'h06; mem[406]=8'hE0; mem[407]=8'hFF; // addi    x12, x0,  -2
```

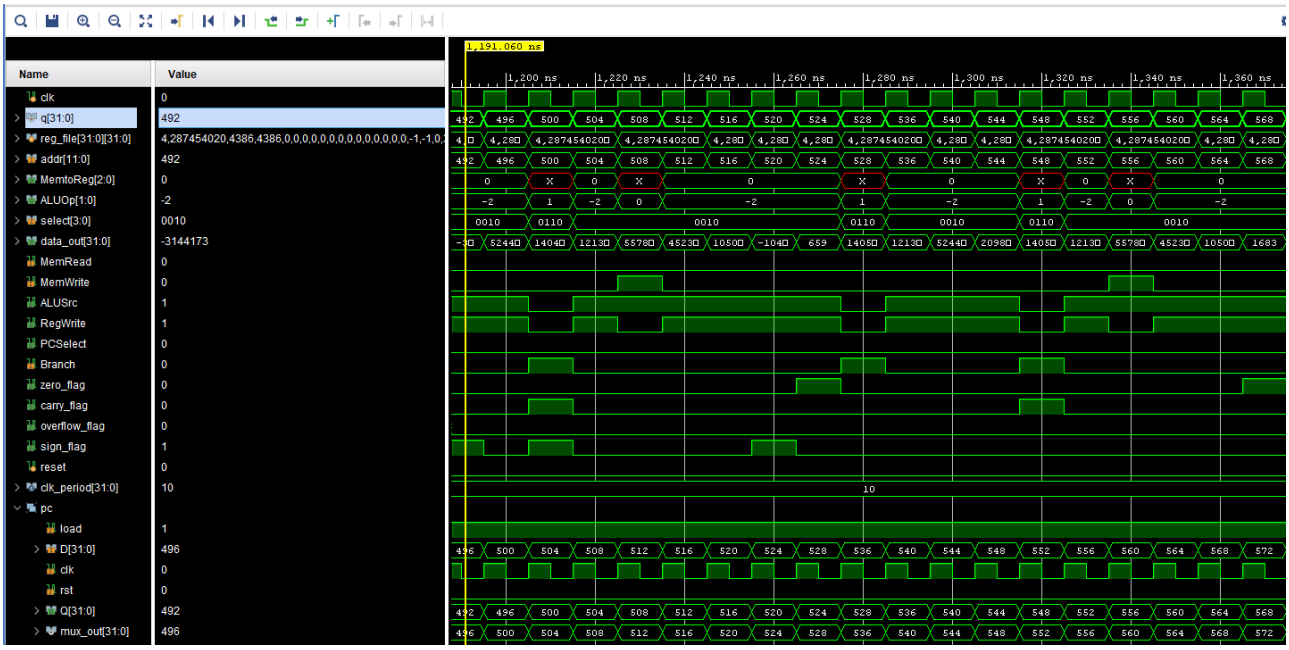

```
mem[408]=8'h93; mem[409]=8'h06; mem[410]=8'h30; mem[411]=8'h00; // addi    x13, x0,  3
mem[412]=8'h93; mem[413]=8'h02; mem[414]=8'h00; mem[415]=8'h00; // addi    x5,  x0,  0
mem[416]=8'h63; mem[417]=8'h44; mem[418]=8'hD6; mem[419]=8'h00; // blt     x12, x13,
L_BLT_TAKEN
mem[420]=8'h6F; mem[421]=8'h00; mem[422]=8'h80; mem[423]=8'h00; // jal     x0,  L_BLT_EQ
        // L_BLT_TAKEN:
mem[424]=8'h93; mem[425]=8'h82; mem[426]=8'h12; mem[427]=8'h00; // addi    x5,  x5,  1
        //  L_BLT_TAKEN:
mem[428]=8'h13; mem[429]=8'h06; mem[430]=8'h70; mem[431]=8'h00; // addi    x12, x0,  7
mem[432]=8'h93; mem[433]=8'h06; mem[434]=8'h70; mem[435]=8'h00; // addi    x13, x0,  7
mem[436]=8'h63; mem[437]=8'h44; mem[438]=8'hD6; mem[439]=8'h00; // blt     x12, x13, L_BLT_NOT1
```

```
mem[440]=8'h93; mem[441]=8'h82; mem[442]=8'h12; mem[443]=8'h00; // addi    x5,  x5,  1
        // L_BLT_NOT1:
mem[444]=8'h13; mem[445]=8'h06; mem[446]=8'h90; mem[447]=8'h00; // addi    x12, x0,  9
mem[448]=8'h93; mem[449]=8'h06; mem[450]=8'h00; mem[451]=8'h00; // addi    x13, x0,  0
mem[452]=8'h63; mem[453]=8'h44; mem[454]=8'hD6; mem[455]=8'h00; // blt     x12, x13, L_BLT_NOT2
mem[456]=8'h93; mem[457]=8'h82; mem[458]=8'h12; mem[459]=8'h00; // addi    x5,  x5,  1
        // L_BLT_NOT2:
mem[460]=8'h23; mem[461]=8'h20; mem[462]=8'h55; mem[463]=8'h00; // sw      x5,  0(x10)
mem[464]=8'h13; mem[465]=8'h05; mem[466]=8'h45; mem[467]=8'h00; // addi    x10, x10, 4
mem[468]=8'h13; mem[469]=8'h06; mem[470]=8'hF0; mem[471]=8'hFF; // addi    x12, x0,  -1
mem[472]=8'h93; mem[473]=8'h06; mem[474]=8'hF0; mem[475]=8'hFF; // addi    x13, x0,  -1
mem[476]=8'h93; mem[477]=8'h02; mem[478]=8'h00; mem[479]=8'h00; // addi    x5,  x0,  0
mem[480]=8'h63; mem[481]=8'h54; mem[482]=8'hD6; mem[483]=8'h00; // bge     x12, x13,
L_BGE_TAKEN
mem[484]=8'h6F; mem[485]=8'h00; mem[486]=8'h80; mem[487]=8'h00; // jal     x0,  L_BGE_NEXT
        // L_BGE_TAKEN:
mem[488]=8'h93; mem[489]=8'h82; mem[490]=8'h12; mem[491]=8'h00; // addi    x5,  x5,  1
```



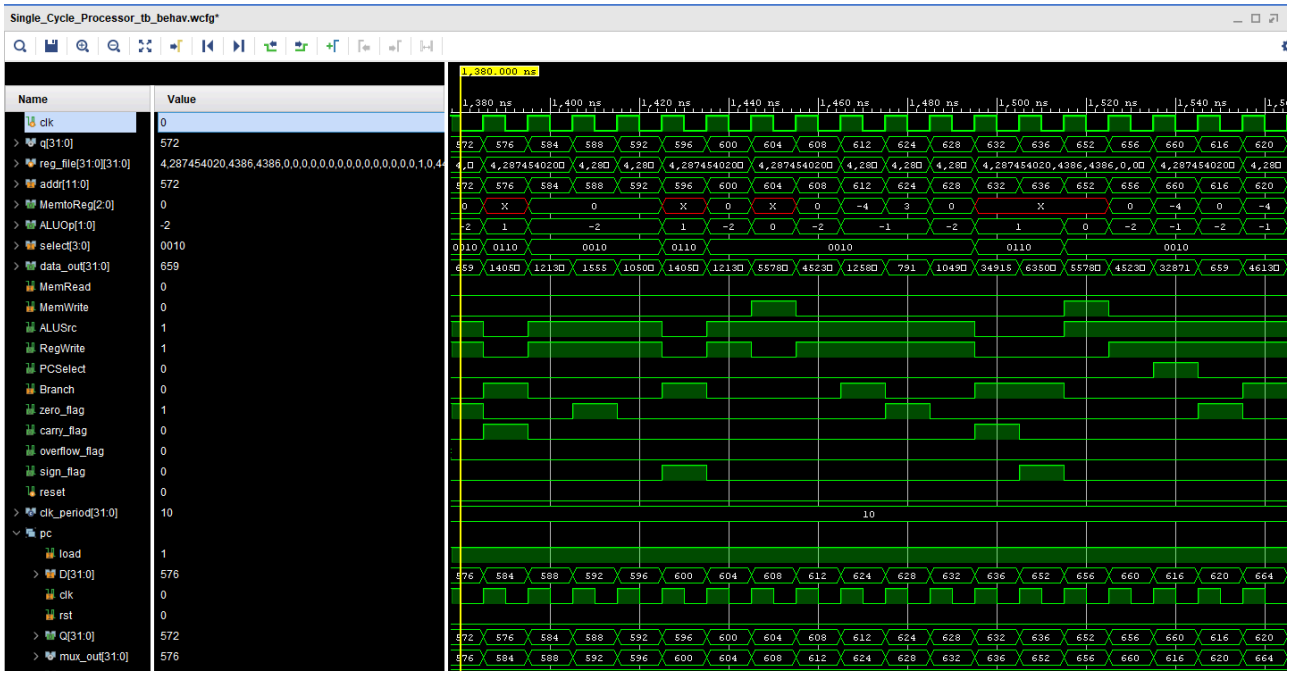

```
mem[492]=8'h13; mem[493]=8'h06; mem[494]=8'hD0; mem[495]=8'hFF; // addi    x12, x0,  -3
mem[496]=8'h93; mem[497]=8'h06; mem[498]=8'h50; mem[499]=8'h00; // addi    x13, x0,  5
mem[500]=8'h63; mem[501]=8'h54; mem[502]=8'hD6; mem[503]=8'h00; // bge     x12, x13, L_BGE_NOT
mem[504]=8'h93; mem[505]=8'h82; mem[506]=8'h12; mem[507]=8'h00; // addi    x5,  x5,  1
        // L_BGE_NOT:
mem[508]=8'h23; mem[509]=8'h20; mem[510]=8'h55; mem[511]=8'h00; // sw      x5,  0(x10)
mem[512]=8'h13; mem[513]=8'h05; mem[514]=8'h45; mem[515]=8'h00; // addi    x10, x10, 4
mem[516]=8'h13; mem[517]=8'h06; mem[518]=8'h10; mem[519]=8'h00; // addi    x12, x0,  1
mem[520]=8'h93; mem[521]=8'h06; mem[522]=8'hF0; mem[523]=8'hFF; // addi    x13, x0,  -1
mem[524]=8'h93; mem[525]=8'h02; mem[526]=8'h00; mem[527]=8'h00; // addi    x5,  x0,  0
```

```
mem[528]=8'h63; mem[529]=8'h64; mem[530]=8'hD6; mem[531]=8'h00; // bltu    x12, x13,
L_BLTU_TAKEN
mem[532]=8'h6F; mem[533]=8'h00; mem[534]=8'h80; mem[535]=8'h00; // jal     x0,  L_BLTU_NEXT
        // L_BLTU_TAKEN:
mem[536]=8'h93; mem[537]=8'h82; mem[538]=8'h12; mem[539]=8'h00; // addi    x5,  x5,  1
        // L_BLTU_NEXT:
mem[540]=8'h13; mem[541]=8'h06; mem[542]=8'h50; mem[543]=8'h00; // addi    x12, x0,  5
mem[544]=8'h93; mem[545]=8'h06; mem[546]=8'h20; mem[547]=8'h00; // addi    x13, x0,  2
mem[548]=8'h63; mem[549]=8'h64; mem[550]=8'hD6; mem[551]=8'h00; // bltu    x12, x13, L_BLTU_NOT
mem[552]=8'h93; mem[553]=8'h82; mem[554]=8'h12; mem[555]=8'h00; // addi    x5,  x5,  1
        // L_BLTU_NOT:
mem[556]=8'h23; mem[557]=8'h20; mem[558]=8'h55; mem[559]=8'h00; // sw      x5,  0(x10)
mem[560]=8'h13; mem[561]=8'h05; mem[562]=8'h45; mem[563]=8'h00; // addi    x10, x10, 4
mem[564]=8'h13; mem[565]=8'h06; mem[566]=8'h10; mem[567]=8'h00; // addi    x12, x0,  1
mem[568]=8'h93; mem[569]=8'h06; mem[570]=8'h00; mem[571]=8'h00; // addi    x13, x0,  0
```




```
mem[568]=8'h93; mem[569]=8'h06; mem[570]=8'h00; mem[571]=8'h00; // addi    x13, x0,  0
mem[572]=8'h93; mem[573]=8'h02; mem[574]=8'h00; mem[575]=8'h00; // addi    x5,  x0,  0
mem[576]=8'h63; mem[577]=8'h74; mem[578]=8'hD6; mem[579]=8'h00; // bgeu    x12, x13,
L_BGEU_TAKEN
mem[580]=8'h6F; mem[581]=8'h00; mem[582]=8'h80; mem[583]=8'h00; // jal     x0,  L_BGEU_NEXT
        // L_BGEU_TAKEN:
mem[584]=8'h93; mem[585]=8'h82; mem[586]=8'h12; mem[587]=8'h00; // addi    x5,  x5,  1
        // L_BGEU_NEXT:
mem[588]=8'h13; mem[589]=8'h06; mem[590]=8'h00; mem[591]=8'h00; // addi    x12, x0,  0
mem[592]=8'h93; mem[593]=8'h06; mem[594]=8'h10; mem[595]=8'h00; // addi    x13, x0,  1
mem[596]=8'h63; mem[597]=8'h74; mem[598]=8'hD6; mem[599]=8'h00; // bgeu    x12, x13,
L_BGEU_NOT
mem[600]=8'h93; mem[601]=8'h82; mem[602]=8'h12; mem[603]=8'h00; // addi    x5,  x5,  1
```

```
        // L_BGEU_NOT:
mem[604]=8'h23; mem[605]=8'h20; mem[606]=8'h55; mem[607]=8'h00; // sw     x5,  0(x10)
mem[608]=8'h13; mem[609]=8'h05; mem[610]=8'h45; mem[611]=8'h00; // addi   x10, x10, 4
mem[612]=8'hEF; mem[613]=8'h00; mem[614]=8'hC0; mem[615]=8'h00; // jal    x1,  L_JAL_TGT
mem[616]=8'h93; mem[617]=8'h02; mem[618]=8'h00; mem[619]=8'h00; // addi   x5,  x0,  0
mem[620]=8'h6F; mem[621]=8'h00; mem[622]=8'hC0; mem[623]=8'h02; // jal    x0,  L_JAL_DONE
```




```
mem[624]=8'h17; mem[625]=8'h03; mem[626]=8'h00; mem[627]=8'h00; // auipc  x6,  0
mem[628]=8'h93; mem[629]=8'h02; mem[630]=8'h10; mem[631]=8'h00; // addi   x5,  x0,  1
mem[632]=8'h63; mem[633]=8'h88; mem[634]=8'h00; mem[635]=8'h00; // beq    x1,  x0,  L_JAL_FAIL
mem[636]=8'h63; mem[637]=8'hE8; mem[638]=8'h60; mem[639]=8'h00; // bltu   x1,  x6,  L_JAL_STORE
mem[640]=8'h93; mem[641]=8'h02; mem[642]=8'h00; mem[643]=8'h00; // addi   x5,  x0,  0
mem[644]=8'h6F; mem[645]=8'h00; mem[646]=8'h80; mem[647]=8'h00; // jal    x0,  L_JAL_STORE
          // L_JAL_FAIL:
mem[648]=8'h93; mem[649]=8'h02; mem[650]=8'h00; mem[651]=8'h00; // addi   x5,  x0,  0
        // L_JAL_STORE:
mem[652]=8'h23; mem[653]=8'h20; mem[654]=8'h55; mem[655]=8'h00; // sw     x5,  0(x10)
mem[656]=8'h13; mem[657]=8'h05; mem[658]=8'h45; mem[659]=8'h00; // addi   x10, x10, 4
mem[660]=8'h67; mem[661]=8'h80; mem[662]=8'h00; mem[663]=8'h00; // jalr   x0,  x1,  0
        // L_JAL_DONE:
mem[664]=8'h13; mem[665]=8'h02; mem[666]=8'h00; mem[667]=8'h00; // addi   x4,  x0,  0
mem[668]=8'h6F; mem[669]=8'h02; mem[670]=8'h80; mem[671]=8'h00; // jal    x4,  L_CALLEE
mem[672]=8'h6F; mem[673]=8'h00; mem[674]=8'h80; mem[675]=8'h00; // jal    x0,  L_AFTER_CALL
        // L_CALLEE:
mem[676]=8'h67; mem[677]=8'h00; mem[678]=8'h02; mem[679]=8'h00; // jalr   x0,  x4,  0
          // L_AFTER_CALL:
mem[680]=8'h93; mem[681]=8'h02; mem[682]=8'h10; mem[683]=8'h00; // addi   x5,  x0,  1
mem[684]=8'h23; mem[685]=8'h20; mem[686]=8'h55; mem[687]=8'h00; // sw     x5,  0(x10)
```

```
mem[688]=8'h13; mem[689]=8'h05; mem[690]=8'h45; mem[691]=8'h00; // addi    x10, x10, 4
        // end:
mem[692]=8'h63; mem[693]=8'h00; mem[694]=8'h00; mem[695]=8'h00; // beq     x0,  x0,  end
      // ret_here:
mem[696]=8'h13; mem[697]=8'h00; mem[698]=8'h00; mem[699]=8'h00; // addi    x0,  x0,  0   (nop)
```