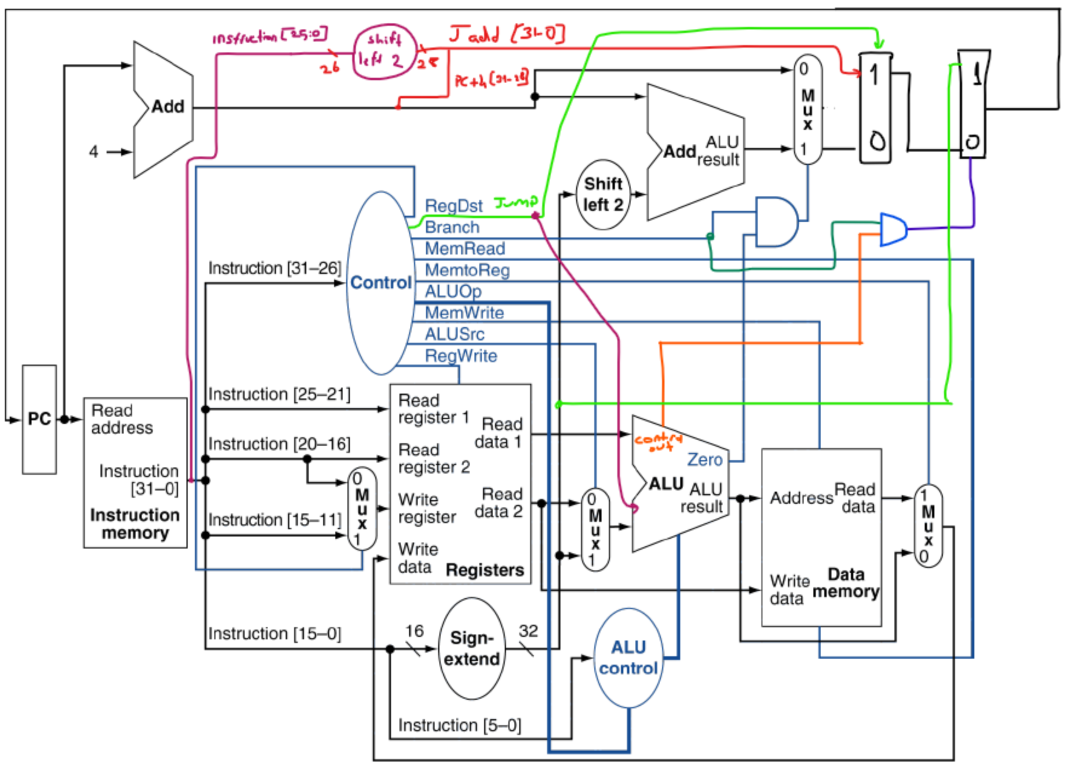
CENG311- Computer Architecture Fall 2022
PROGRAMMING ASSIGNMENT #3

DESIGN:



- My primary goal was to complete the "addi,nor and jump" instructions.
- For this reason, I first started by designing the jump directive. I bought Jump's designs as we saw in the lesson. We shift the 25-0 bits of the directive by two and add them with bits 31-28 of pc to get the address. The result is obtained according to the Jump flag.
- Since the nor directive is r-format, there was no change until alucontrol. Since we looked at the green sheet and the opcode of nor was 100111, I added an if block accordingly. I gave 4'b0101 as the output. There is an if block for the output in Alu.
- The opcode had to be looked at for the Addi instruction. For this reason, I provided control with a cable for this in control. When the Aluop output is 0, the necessary addition for lw, sw and addi is performed.
- I have defined 3 outputs for Aluop. Because looking at the number of opcodes, I couldn't define enough different instructions with 2 of them. The table below is an explanation for this.

ALUOP	321
I type	000
R type	001
Branch	010
Ben	011
Bvf	100

❖ For the changes/additions made for “ben and the bvf” instructions, I will first mention the differences in the code after the above image.

- There are 3 different possibilities on the PC. These muxs are branch, jump and newly added "ben and bvf" addresses respectively.

- I used an and operation as the selection bit for the 3.mux above. In this process, I've and'ed the branch and control_out bits.
- The branch flag becomes 1 with ben, bvf and branch. If control_out is 1 that means the instruction is "ben or bvf" and the required condition is met in the previous instruction.
- The reason why the jump flag is connected to alu will be explained in more detail below, but briefly, bvf or ben comes right after the jump and it is already there at the address in the jump.

```
                jump label_1
        label_1: bvf label_2
```

- In our new instructions, as you mentioned in the document, we sign extend the 16 bits in the instruction and use it as an address. And I'm moving that into the mux.

CODE

- There are 4 cables added to the Control. These are used to determine the instruction from the opcode. (ben,bvf,addi,j)
- The flags for addi are alusrc, regwrite.
- The flag for beq, and ben is branch.
- The flag for j is jump.
- The required aluop results are mentioned above.
- After control, the flags are active/inactive to the required parts. However, since aluops go to alucontrol, I will talk about them next.
- When we look at the changes made in the code, we can see the added cables because there is an increase in the number of inputs. The nor instruction has been added under the if block of the r format.
- Ben and bvf are decided on aluop and their outputs are 1000 and 1001 respectively. While instruction number is increasing gout became not insufficient and I expand it to 4 bits.
- My general logic in Alu was as follows. If I am annotating the result of an operation with alu_out, there is a possibility of a sign, overflow or zero. I implemented this based on each instruction. The biggest reason for me to make this distinction is that in some cases, if the result is zero and it means there is no need to take not of alu_out etc., the zero register is assigned directly. The same situation exists in other registers in different ways.
- It calculates the result of an instruction and changes the svz registers according to the result. But if the jump (the case mentioned above) comes before bvf or ben it should reset svz as it does not involve any arithmetic operation. If we don't, svz will be affected by the instruction before the jump.
- The condition is checked when ben or bvf. control_out is 1 if true, 0 otherwise. control_out is here as the flag that controls the branch event. Then the svz is reset.
- 2 extra muxes in the Processor are defined and the result of the 6th one is assigned to the PC.
- Cables that need to be defined in line with the additions made in other components were defined. Added signext jump_sext. and operation which decides with control_out and branch is done.

Important Note: When I simulated the program, each instruction was running first for itself, then with the dataa and datab of the next instruction. Bvf and Ben were not working properly since the second made affected the values in the svz. For each instruction to run once, I put a clock-like mechanism inside the alu32. Update fixes the number of alu32's work to one in each cycle.

I would like to explain with an example why there are separate calculations for each instruction in alu32. Example: If we take 1 and 1 in the nand operation, the result will be 0. However, this should not be considered an overflow. For this reason, each is evaluated in its own if block.

Instruction to binary and then hexadecimal:

Label	Instruction	Binary	Hex
	(pc = 0x00) — j label_1 //→ label_1 should be 0x0C	00 0010 0000 0000 0000 0000 0000 00 0011	08000003
label_2:			
	(pc = 0x04) — add $r4, $r4, $r5 // →r4 = (0xFFFFFFFF + 0x80000000)	00 0000 00100 00101 00100 00000 10 0000	00852020
	(pc = 0x08) — bvf label_3 // label_3 should be 0x18, V = 1	00 0101 00000 00000 00000 00000 01 1000	14000018
label_1:			
	(pc = 0x0C) — nor $r1, $r2, $r3 // ~(r2 = 0x05 | r3 = 0x0A) → r1 = 0xFFFFFFF0	00 0000 00010 00011 00001 00000 10 0111	00430827
	(pc = 0x10) — addi $r4, $r1, 0x0F // r4 = 0xFFFFFFFF, S = 1	00 1000 00001 00100 00000 00000 00 1111	2024000F
	(pc = 0x14) — ben label_2 // label_2 should be 0x04	00 0110 00000 00000 00000 00000 00 0100	18000004
label_3:			
	(pc = 0x18) — j exit // exit should be 0x40	00 0010 0000 0000 0000 0000 00 0001 0000	08000010
exit:			
	(pc = 0x40) // we do not care the code after exit branch.		

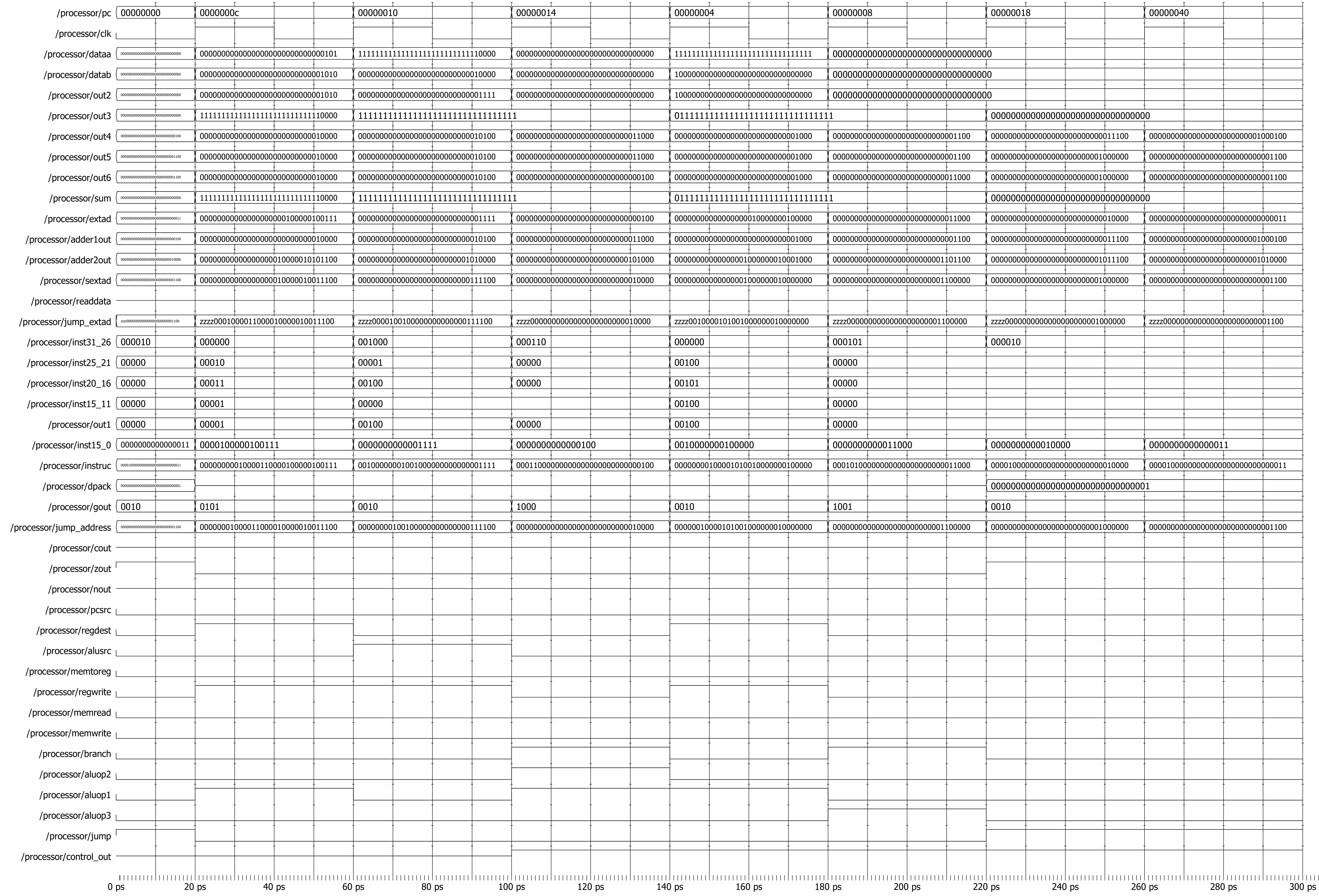
Registers before start:

Register	Hex Value
r0	00000000
r1	00000000
r2	00000005
r3	0000000A
r4	00000010
r5	80000000

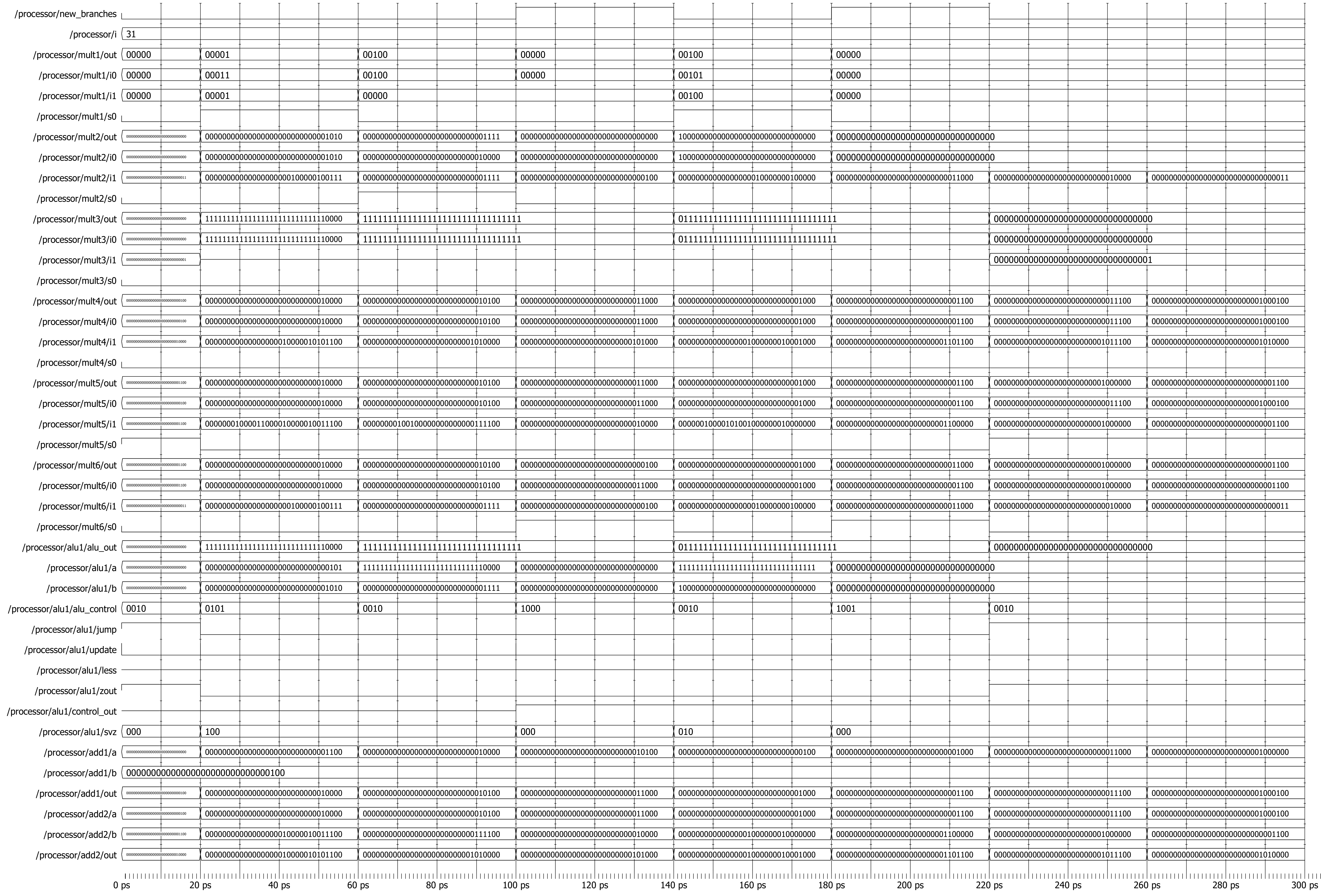
Step by step changing at registers:

Note: pc, dataa, datab etc. the register values are not included here as they are available in the dataflow provided below.

Instruction	Changed Register - Hex Value
(pc = 0x00) — j label_1	None
(pc = 0x04) — add $r4, $r4, $r5	r4 = 7FFFFFFF
(pc = 0x08) — bvf label_3	None
(pc = 0x0C) — nor $r1, $r2, $r3	r1 = FFFFFFF0
(pc = 0x10) — addi $r4, $r1, 0x0F	r4 = FFFFFFFF
(pc = 0x14) — ben label_2	None
(pc = 0x18) — j exit	None

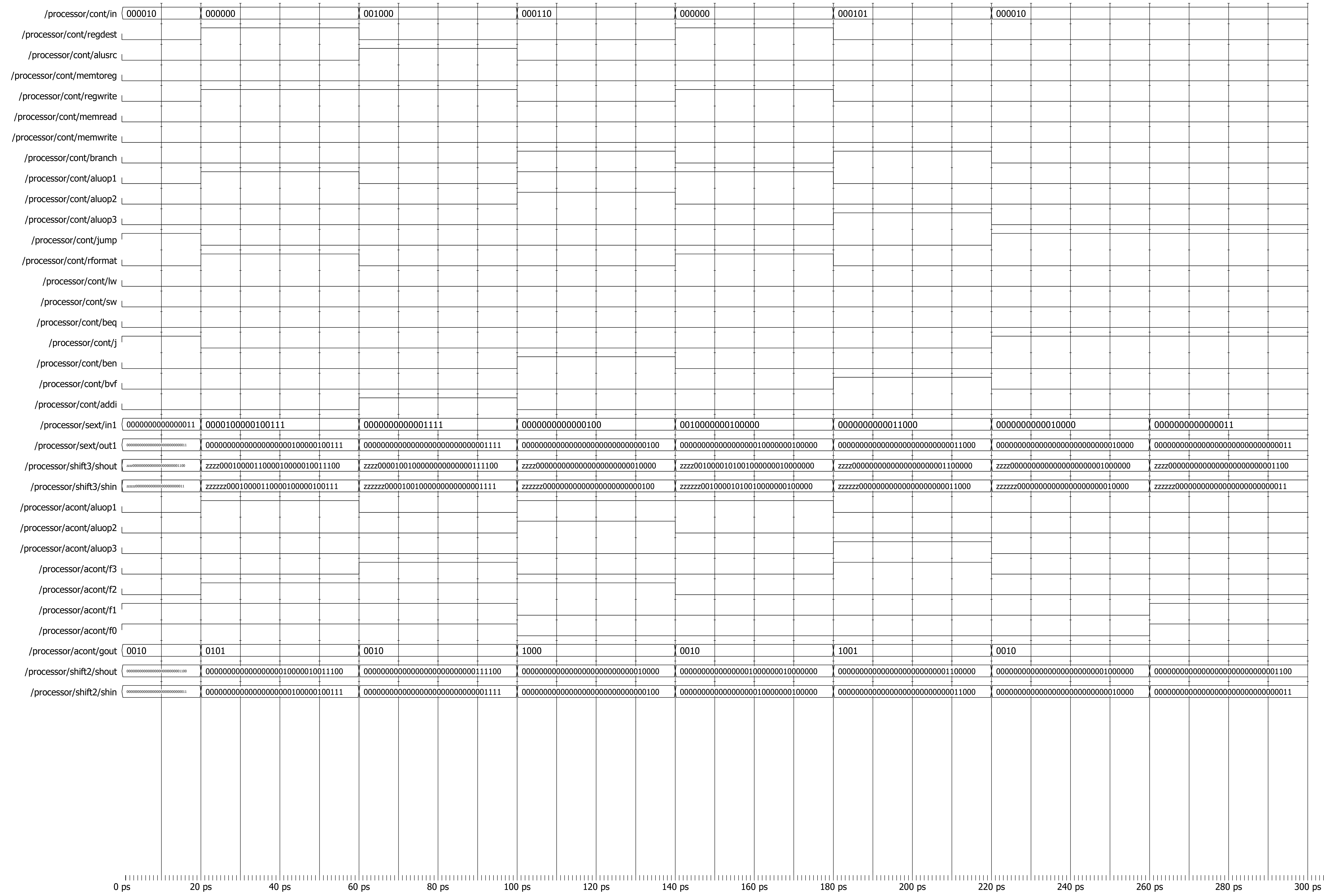


Entity:processor Architecture: Date: Fri Dec 23 19:39:17 AST 2022   Row: 1 Page: 1



```
/processor/new_branches
/processor/i                 31
/processor/mult1/out         00000  00001  00100  00000  00100  00000
/processor/mult1/i0          00000  00011  00100  00000  00101  00000
/processor/mult1/i1          00000  00001  00000  00100  00000
/processor/mult1/s0
/processor/mult2/out         00000000000000000000000000000000  00000000000000000000000000001010  00000000000000000000000000001111  00000000000000000000000000000000  10000000000000000000000000000000  00000000000000000000000000000000
/processor/mult2/i0          00000000000000000000000000000000  00000000000000000000000000001010  00000000000000000000000000010000  00000000000000000000000000000000  10000000000000000000000000000000  00000000000000000000000000000000
/processor/mult2/i1          00000000000000000000000000000011  00000000000000000000100000100111  00000000000000000000000000001111  00000000000000000000000000000100  00000000000000000010000000100000  00000000000000000000000000011000  00000000000000000000000000010000  00000000000000000000000000000011
/processor/mult2/s0
/processor/mult3/out         00000000000000000000000000000000  11111111111111111111111111110000  11111111111111111111111111111111  01111111111111111111111111111111  00000000000000000000000000000000
/processor/mult3/i0          00000000000000000000000000000000  11111111111111111111111111110000  11111111111111111111111111111111  01111111111111111111111111111111  00000000000000000000000000000000
/processor/mult3/i1          00000000000000000000000000000001  00000000000000000000000000000001
/processor/mult3/s0
/processor/mult4/out         00000000000000000000000000000100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000011000  00000000000000000000000000001000  00000000000000000000000000001100  00000000000000000000000000011100  00000000000000000000000001000100
/processor/mult4/i0          00000000000000000000000000000100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000011000  00000000000000000000000000001000  00000000000000000000000000001100  00000000000000000000000000011100  00000000000000000000000001000100
/processor/mult4/i1          00000000000000000000000000010000  00000000000000000100000101011100  00000000000000000000000001010000  00000000000000000000000000101000  00000000000000001000000010001000  00000000000000000000000001101100  00000000000000000000000001011100  00000000000000000000000001010000
/processor/mult4/s0
/processor/mult5/out         00000000000000000000000000001100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000011000  00000000000000000000000000001000  00000000000000000000000000001100  00000000000000000000000001000000  00000000000000000000000000001100
/processor/mult5/i0          00000000000000000000000000000100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000011000  00000000000000000000000000001000  00000000000000000000000000001100  00000000000000000000000000011100  00000000000000000000000001000100
/processor/mult5/i1          00000000000000000000000000001100  00000001000011000010000010011100  00000000100100000000000000111100  00000000000000000000000000010000  00000010000101001000000010000000  00000000000000000000000001100000  00000000000000000000000001000000  00000000000000000000000000001100
/processor/mult5/s0
/processor/mult6/out         00000000000000000000000000001100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000000100  00000000000000000000000000001000  00000000000000000000000000011000  00000000000000000000000001000000  00000000000000000000000000001100
/processor/mult6/i0          00000000000000000000000000001100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000011000  00000000000000000000000000001000  00000000000000000000000000001100  00000000000000000000000001000000  00000000000000000000000000001100
/processor/mult6/i1          00000000000000000000000000000011  00000000000000000000100000100111  00000000000000000000000000001111  00000000000000000000000000000100  00000000000000000010000000100000  00000000000000000000000000011000  00000000000000000000000000010000  00000000000000000000000000000011
/processor/mult6/s0
/processor/alu1/alu_out      00000000000000000000000000000000  11111111111111111111111111110000  11111111111111111111111111111111  01111111111111111111111111111111  00000000000000000000000000000000
/processor/alu1/a            00000000000000000000000000000000  00000000000000000000000000000101  11111111111111111111111111110000  00000000000000000000000000000000  11111111111111111111111111111111  00000000000000000000000000000000
/processor/alu1/b            00000000000000000000000000000000  00000000000000000000000000001010  00000000000000000000000000001111  00000000000000000000000000000000  10000000000000000000000000000000  00000000000000000000000000000000
/processor/alu1/alu_control  0010  0101  0010  1000  0010  1001  0010
/processor/alu1/jump
/processor/alu1/update
/processor/alu1/less
/processor/alu1/zout
/processor/alu1/control_out
/processor/alu1/svz          000  100  000  010  000
/processor/add1/a            00000000000000000000000000000000  00000000000000000000000000001100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000000100  00000000000000000000000000001000  00000000000000000000000000011000  00000000000000000000000001000000
/processor/add1/b            00000000000000000000000000000100
/processor/add1/out          00000000000000000000000000000100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000011000  00000000000000000000000000001000  00000000000000000000000000001100  00000000000000000000000000011100  00000000000000000000000001000100
/processor/add2/a            00000000000000000000000000000100  00000000000000000000000000010000  00000000000000000000000000010100  00000000000000000000000000011000  00000000000000000000000000001000  00000000000000000000000000001100  00000000000000000000000000011100  00000000000000000000000001000100
/processor/add2/b            00000000000000000000000000001100  00000000000000000100000010011100  00000000000000000000000000111100  00000000000000000000000000010000  00000000000000001000000010000000  00000000000000000000000001100000  00000000000000000000000001000000  00000000000000000000000000001100
/processor/add2/out          00000000000000000000000000010000  00000000000000000100000101011100  00000000000000000000000001010000  00000000000000000000000000101000  00000000000000001000000010001000  00000000000000000000000001101100  00000000000000000000000001011100  00000000000000000000000001010000

0 ps   20 ps   40 ps   60 ps   80 ps   100 ps   120 ps   140 ps   160 ps   180 ps   200 ps   220 ps   240 ps   260 ps   280 ps   300 ps
```


/processor/cont/in
000010
000000
001000
000110
000000
000101
000010
/processor/cont/regdest
/processor/cont/alusrc
/processor/cont/memtoreg
/processor/cont/regwrite
/processor/cont/memread
/processor/cont/memwrite
/processor/cont/branch
/processor/cont/aluop1
/processor/cont/aluop2
/processor/cont/aluop3
/processor/cont/jump
/processor/cont/rformat
/processor/cont/lw
/processor/cont/sw
/processor/cont/beq
/processor/cont/j
/processor/cont/ben
/processor/cont/bvf
/processor/cont/addi
/processor/sext/in1
0000000000000011
0000100000100111
0000000000001111
0000000000000100
0010000000100000
0000000000011000
0000000000010000
0000000000000011
/processor/sext/out1
00000000000000000000100000100111
00000000000000000000000000001111
00000000000000000000000000000100
00000000000000000010000000100000
00000000000000000000000000011000
00000000000000000000000000010000
00000000000000000000000000000011
/processor/shift3/shout
zzzz0001000011000010000100111100
zzzz0000100100000000000000111100
zzzz0000000000000000000000010000
zzzz0010000101001000000010000000
zzzz0000000000000000000001100000
zzzz0000000000000000000001000000
zzzz0000000000000000000000001100
/processor/shift3/shin
zzzzzz0001000011000010000100111
zzzzzz0000100100000000000001111
zzzzzz0000000000000000000000100
zzzzzz0010000101001000000100000
zzzzzz0000000000000000000011000
zzzzzz0000000000000000000010000
zzzzzz0000000000000000000000011
/processor/acont/aluop1
/processor/acont/aluop2
/processor/acont/aluop3
/processor/acont/f3
/processor/acont/f2
/processor/acont/f1
/processor/acont/f0
/processor/acont/gout
0010
0101
0010
1000
0010
1001
0010
/processor/shift2/shout
00000000000000000010000010011100
00000000000000000000000000111100
00000000000000000000000000010000
00000000000000001000000010000000
00000000000000000000000001100000
00000000000000000000000001000000
00000000000000000000000000001100
/processor/shift2/shin
00000000000000000000100000100111
00000000000000000000000000001111
00000000000000000000000000000100
00000000000000000010000000100000
00000000000000000000000000011000
00000000000000000000000000010000
00000000000000000000000000000011
0 ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps
200 ps
220 ps
240 ps
260 ps
280 ps
300 ps