CS61C F20 Midterm Solutions

Instructors: Dan Garcia, Borivje Nikolic

Head TAs: Stephan Kaminsky, Cece McMahon

Question Breakdown

Section 1: Float (*5 pts*, *60 minutes*)
Section 2: Quest Clobber—C (*10 pts*, *120 minutes*)
Section 3: RISC-V Assembly (*10 pts*, *60 minutes*)
Section 4: SDS (*10 pts*, *60 minutes*)
Section 5: RISC-V Datapath, Control, and Pipelining (*10 pts*, *60 minutes*)

If you believe there are any mistakes, please let staff know ASAP! All diagrams are located at the end of the PDF for reference's sake.\

Q1: Float(*5 pts*)

Consider a `w`-bit floating-point number with the following components (1 sign bit, `e` exponent bits, `m` mantissa bits); i.e. all other properties of IEEE754 apply (bias, denormalized numbers, infinities, NaNs, etc...). The bias is the usual $-(2^{e-1}-1)$.

Part A* — *3 pts
What is the bit representation (in hex) of the floating-point number `n`? **Do NOT include the `0x` prefix when writing your answer.**

Solutions

1. `bias =  - ((1 << (e - 1)) - 1)`
2. `bin_int` = $\log_2(leftofdecimalpoint) - 1$ = num shifts left for floating point needed to represent whole part of number based on FP equation (remember there's an implicit 1)
3. `bias + bin_int` = numerical representation of exponent
4. `bin_first_half` = 0-pad up to `e` length to the left
5. `bin_exp` = calculate binary representation of decimal portion of the floating point value
6. Concatenate `1 + bin_first_half + bin_exp` and 0-pad to the right until the floating point is `w` wide and the mantissa is `m` bits wide.
7. Convert to hex
   - **Note:** if you left your answer in terms of binary, you received 2/3 points; if you sign-extended your answer instead of 0-padding and converted to hex, you received 1.5/3 points. The reason for this was because sign-extension only works if you have the physical bits to work with. In the case of converting between different bases, all we're doing is padding such that we have multiples of whatever base we're in—we don't actually have extra bits to work with.

Part B — 2 pts
How many floats are there in the range `[ low val, high val)`? You can either simplify your answer or leave it in exponent form. If you leave it in exponent form, you must use `**` for exponentiation. For example, if your answer is 8, (i.e. `2**3`), you can put either answer down.

Solutions

1. Calculate the binary floating point representation of the lower bound
2. Calculate the binary floating point representation of the upper bound
3. Get the lower significand bits (`low_sig`) and upper significant bits (`high_sig`)
4. Get the lower exponent bits (`low_exp`) and upper exponent bits (`high_exp`)
5. If your exponent bits are the same, your range is the difference between the significands.
6. If your exponent bits are not the same, calculate the difference between your `high_sig` and 0 if applicable as well as `lower_sig` and `1 << m`. That's your excess. Your overall range are those values plus `(high_sig - low_sig - 1) * (1 << m)`.

Retake Q1: Float (*5 pts*)

Same as original question, different parameters

Q2: Quest Clobber (*10 pts*)

Part A — 3 pts
Help! We have two robots, Alexa and Siri, but they're speaking the wrong languages! Alexa sends sensor data with the bits as `uint8_t`s encoding `ENCODING TYPE 1` which would later be read by Siri. However, Siri can only understand `ENCODING TYPE 2`. Your job is to write a simple function `ENCODING TYPE 1 TO 2` so that Siri can correctly convert messages received from Alexa. (E.g. if Alexa saw "-3" and encoded it as `ENCODING TYPE 1`, you'd need to return a new set of bits such that Siri would interpret those bits in `ENCODING TYPE 2` as "-3" as well.) Note that the range of the sensor is such that it would never produce a number that Siri couldn't handle. If you're ever given the choice between storing +0 or -0, store +0. If you are not able to complete this part, you can still receive full credit on Parts B and C.

Solutions
bias2ones

```
uint8_t bias2ones(uint8_t bias) {
    return (bias >= 0x7F) ? bias - 127 : bias + 128;
}
```

ones2bias

```
uint8_t ones2bias(uint8_t bias) {
    return (ones & 0x80) ? ones - 128 : ones + 127;
}
```

bias2sm

```
uint8_t bias2sm(uint8_t bias) {
    return (bias >= 0x7F) ? bias - 127 : 255 - bias;
}
```

sm2bias

```
uint8_t sm2bias(uint8_t bias) {
    return (sm & 0x80) ? 255 - sm : sm + 127;
}
```

Part B — 4 pts
After much deliberation, the robots have agreed to use unsigned numbers instead. They have stored some data in a binary node structure, but have realized that all their data is one less than the correct value. Complete the code for `TREE *incr_tree(TREE *p)` that returns a duplicate tree in which every number has been incremented by 1. You must use `CS61C_malloc()` instead of `malloc()`. You can assume that every call to `CS61C_malloc()` succeeds.

```
typedef struct node {
    struct node *L;
    struct node *R;
    uint8_t N;
} TREE;
```

Solutions

```
TREE *incr_tree(TREE *p) {
    if (!p) {
        return NULL;
    }
    TREE *q = (TREE *) CS61C_malloc (sizeof(TREE));
    q->N = p->N + 1;
    q->L = incr_tree(p->L);
    q->R = map_tree(p->R);
    return q;
}
```

Part C — 3 pts

Clean up behind yourself! Write a function `void free_tree(TREE *p)` which will free all of the space used by the input tree `p`. You may assume that all of the nodes in `p` were `malloc'd` properly. You must use `CS61C_free()` instead of `free()`. You may edit `main.c` to test your code. Your code should be able to run without modifications in `quest.h`, or the signatures of the three functions. You will only be submitting `quest.c` and only the code in that file will be graded.

Solutions

```
if (p) {
    free_tree(p->L);
    free_tree(p->R);
    CS61C_free(p);
}
```

Retake Q2: Quest Clobber (*10 pts*)

Part A — 3 pts
Question prompt is the same as the original question

Solutions
ones2sm

```
uint8_t ones2sm(uint8_t ones) {
    return (ones & 0x80) ? 128 - ones + 255 : ones;
}
```

sm2ones

```
uint8_t ones2sm(uint8_t sm) {
    return (sm & 0x80) ? 255 - sm + 128 : sm;
}
```

Part B — 4 pts
Consider the binary node structure shown in the Appendix. Complete the code for `TREE num_tree(int n)` that returns a balanced binary tree of height `n` in which the node numbers are listed in the order as shown in the examples. You must use `CS61C_malloc()` instead of `malloc()`. You can assume that every call to `CS61C_malloc()` succeeds.

```
typedef struct node {
    struct node *L;
    struct node *R;
    uint8_t N;
} TREE;
```

Solutions

```
TREE *num_tree(int n) {
    if (!n) {
        return NULL;
    } else {
        TREE *t = (TREE *) CS61C_malloc (sizeof(TREE));
        t->L = num_tree(n-1);
        t->N = count++;
        t->R = num_tree(n-1);
        return t;
    }
}
```

Part C — 3 pts
Question prompt is the same as the original question

Q3: RISC-V (*10 pts*)

Part A — 2 pts
What is the machine code (in hex) of `INSTRUCTION`? *Do NOT prefix your solution with `0x`.* Please pad your answer to a full 4 bytes when submitting if necessary. See Gradescope for your specific instruction.

Solutions
Part B — 8 pts
Write a function in RISC-V that takes a string of only letters (uppercase and lowercase) terminated appropriately and lowercases it, returning the length of the string; call it `FUNCTION NAME`.

Solutions
Note: There were six variations on this question — `UpperSum`, `LowerSum`, `InvertSum`, `UpperLen`, `LowerLen,`, and `InvertLen`. These respectively uppercases, lowercases, or inverts the given strings and then either sums the original character values or gets the length of the string. All versions have similar solutions. Here is our solution for LowerLen; note that it relies on the fact that the capital letter and lowercase letter differ exactly by one bit; for example, `a = 0x61`, and `A = 0x41`:

```
addi t0 x0 0
Loop:
    lb t1, 0(a0)
    beq t1, x0, End
    addi t0, t0, 1 #For Sum, this is add t0, t0, t1
    #For Upper, this next line is andi t1, t1, 0b11011111; for Invert, this is xori t1, t1, 0b00100000
    ori t1, t1, 0b00100000
    sb t1, 0(a0)
    addi a0, a0, 1
    j Loop
End:
    mv a0, t0
    jr ra
```

Retake Q3: RISC-V (*10 pts*)

Part A — 2 pts
Question prompt is the same as the original question

Part B — 8 pts
Write a function in RISC-V that takes a string of only letters (uppercase and lowercase) terminated appropriately and a target character, and upper cases/lower cases/inverts cases the characters before the given target character and returns 1 if the target character was found else 0; call it `FUNCTION NAME`.

Solutions
Note: There were three variations on this question — `UpperSearch`, `LowerSearch`, and `InvertSearch`. These respectively uppercases, lowercases, or inverts the cases of a given string before a given character in the string and then returns 1 if the target character was found and 0 elsewise. All versions have similar solutions. Here is our solution for InverSearch; note that it relies on the fact that the capital letter and lowercase letter differ exactly by one bit; for example, `a = 0x61`, and `A = 0x41`:

```
InvertSearch:
#Reads through the string in a0, and searches for the character in a1.
#Converts the letters before the target character is found to reverse case (Cal -> cAL)
#Returns 1 if the target character is found, and 0 otherwise.
#The input string consists only of alphabetic characters (a-z, A-Z), and is properly formatted.
Loop:
    lb t0 0(a0)
    beq t0 x0 Exit2
    beq t0 a1 Exit
    xori t0 t0 0x20
    sb t0 0(a0)
    addi a0 a0 1
    j Loop
Exit:
    li a0 1
    jr ra
Exit2:
    li a0 0
    jr ra
```

Q4: SDS (*10 pts*)

Truth Table *4 pts*

Fill out the truth table for the circuit shown in the Appendix. For wires that intersect, assume the signal follows a straight path until the wire turns to feed into a logic gate's input.

Solutions

Version 1

x	**y**	**z**	**out**
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

Version 2

x	y	z	out
0	0	0	**1**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

Version 3

x	y	z	out
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

Version 4

x	y	z	out
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**

x	y	z	out
1	1	1	**1**

Version 5

x	y	z	out
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**0**

Version 6

x	y	z	out
0	0	0	**1**
0	0	1	**0**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**

SDS

Part A — 3 pts
The logic implementation of a state machine is shown in the figure in the Appendix. How many reachable states does this state machine have?

Assume that the starting state is `Out0 = 0`, `Out1 = 0`, `Out2 = ?`. Note, `Out2` is variable but all versions have the same answer.

Solutions
Version 1
Starting State: `Out0 = 0`, `Out1 = 0`, `Out2 = 0`
States: 7

Version 2
Starting State: `Out0 = 0`, `Out1 = 0`, `Out2 = 0`
States: 7

Version 3
Starting State: `Out0 = 0`, `Out1 = 0`, `Out2 = 1`
States: 7

Part B — 3 pts
In the figure in the Appendix and from Part A, the flip-flop clk-to-q delay is `CLK` ps, the setup time is `SETUP` ps, the XOR delay is `XOR` ps, and the inverter time if it applies is `INVERTER` ps. What is the minimum cycle of operation?

Solutions
Version 1

```
critical path = CLK + XOR + SETUP
```

Version 2

```
critical path = CLK + XOR + INVERTER + SETUP
```

Version 3

```
critical path = CLK + (XOR + INVERTER) * 2 + SETUP
```

Part C — 2 pts In the above figure, what is the longest hold time for the flip-flop that allows for correct operation?

Solutions
For all versions, the answer is `CLK` ps.

Retake Q4: SDS (*10 pts*)

Truth Table
Question prompt is the same as the original question.

All versions are the same as the original midterm's Q4: SDS (Truth Table) subsection.

SDS *6 pts*

Part A — 3 pts
The logic implementation of a state machine is shown in the figure in the Appendix. How many reachable states does this state machine have?

Assume that the starting state is `Out0 = ?`, `Out1 = ?`, `Out2 = ?`.

See Appendix for Version Diagrams

Solutions
Version 1
Starting State: `Out0 = 0`, `Out1 = 0`, `Out2 = 0`
States: `4`

Version 2
Starting State: `Out0 = 1`, `Out1 = 1`, `Out2 = 1`
States: `4`

Part B — 3 pts
In the figure in the Appendix and from Part A, the flip-flop clk-to-q delay is `CLK` ps, the setup time is `SETUP` ps, the XOR delay is `XOR` ps, and the inverter time if it applies is `INVERTER` ps. What is the minimum cycle of operation?

Solutions
Versions 1 + 2
`critical path = CLK + XOR + INVERTER + SETUP`

Part C — 2 pts
In the above figure, what is the longest hold time for the flip-flop that allows for correct operation?

Solutions
For all versions, the answer is `CLK` ps.

Q5: RISC-V Datapath, Control, and Pipelining (*10 pts*)

Part A — 4 pts
The datapath in the Appendix implements the RV32I instruction set.

In the RISC-V datapath above, marked what is used by `INSTR` instruction.

Possible `INSTR`s: `beq`, `bne`, `addi`, `ori`, `lb`, `sb`

Solutions

`beq`
PC Sel Mux: Input Dependent
ASel Mux: `pc` branch
BSel Mux: `imm` branch
WBSel Mux: `*` (don't care)
Datapath Units: Branch Comp, Imm Gen
Regfile: Read `Reg[rs1]`, Read `Reg[rs2]`

`bne`
PC Sel Mux: Input dependent
ASel Mux: `pc` branch
BSel Mux: `imm` branch
WBSel Mux: `*` (don't care)
Datapath Units: Branch Comp, Imm Gen
Regfile: Read `Reg[rs1]`, Read `Reg[rs2]`

`addi`
PC Sel Mux: `pc + 4` branch
ASel Mux: `Reg[rs1]` branch
BSel Mux: `imm` branch
WBSel Mux: `ALU` branch
Datapath Units: Imm Gen
Regfile: Read `Reg[rs1]`, Write `Reg[rd]`

`ori`
PC Sel Mux: `pc + 4` branch
ASel Mux: `Reg[rs1]` branch
BSel Mux: `imm` branch
WBSel Mux: `ALU` branch
Datapath Units: Imm Gen
Regfile: Read `Reg[rs1]`, Write `Reg[rd]`

`lb`
PC Sel Mux: `pc + 4` branch
ASel Mux: `Reg[rs1]` branch
BSel Mux: `imm` branch
WBSel Mux: `mem` branch
Datapath Units: Imm Gen, Load Extend
Regfile: Read `Reg[rs1]`, Write `Reg[rd]`

`sb`
PC Sel Mux: `pc + 4` branch
ASel Mux: `Reg[rs1]` branch
BSel Mux: `imm` branch
WBSel Mux: `*` (don't care)

Datapath Units: Imm Gen
Regfile: Read `Reg[rs1]`, Read`Reg[rs2]`

Part B — 3 pts
Specify whether the following proposed instructions can be implemented using this datapath without modifications. If the instruction can be implemented, specify an expression for the listed control signals, by following the example below. If the instruction is not implementable, write "No" in the implementable column and "N/A" in the Control Signals column.

Possible Instructions: Load word with add (`lwadd rd, rs1, rs2, imm`), `beq` with writeback (`beq rd, rs1, rs2, imm`), PC-relative load (`lwpc rd, imm`), Register offset load (`lwreg rd, rs1, rs2`), Jump to zero (`jzero rs1, rs2`), Negate (`neg rd, rs1`)

Solutions
Instruction: implementable, ASel, BSel
`lwadd rd, rs1, rs2, imm`: No, N/A, N/A
`beq rd, rs1, rs2, imm`: No, N/A, N/A
`lwpc rd, imm`: Yes, `pc`, `imm` (and synonymous signals)
`lwreg rd rs1, rs2`: Yes, `reg`, `reg` (and synonymous signals)
`jzero rs1, rs2`: Yes, `*`, `imm` (and synonymous signals)

- **Alternate Answer:** Yes, `reg`, `reg` (and synonymous signals); this would only work in the instance that `Reg[rs1] == Reg[rs2]` purposefully to get `Reg[rs1] - Reg[rs2] = 0` `neg rd, rs1`: Yes, `reg`, `imm` (and synonymous signals)
- **Alternate Answer:** Not Implementable (must give reasoning in regrade request)

Part C — 3 pts
Consider the 5-stage pipelining presented in lecture with the combinational-read IMEM and DMEM and the forwarding paths as drawn in the pipelined diagram in the Appendix.

Write the number of NOPs needed between each instruction, and list the hazard that causes the stall. If no hazard occurs, select "None", and list the number of NOPs as 0. Assume you can write to and read from the same address in the register file (Reg [ ]) in the same cycle. If there is a branch, assume that it is not taken, and there is no branch prediction. Consider two cases:
Case 1: Forwarding is not implemented (the diagram as seen in lecture)
Case 2: The forwarding muxes in the diagram are driven correctly by the forwarding logic. Note: If a hazard is resolved by forwarding and no other hazard is present, select "None" for the hazard

There are 6 versions.

NOTE: there are alternative answers depending on what version of the diagram you used and which clarifications you received. Please submit a regrade request detailing which diagram you used and your reasoning for your answer if you believe your answer is correct.

Solutions
Instruction Set
Instruction Pair: Case 1 Hazard, Case 1 nops, Case 2 Hazard, Case 2 nops

Version 1

```
1         addi x1, x0, 0xFF
2         ori x2, x1, 0x7FF
3         bge x1, x2, label
4         xori x2, x2, 1
```

1-2: Data, 2, None, 0
2-3: Data, 2, Data, 2
3-4: Control, 2, Control, 3

Version 2

```
1         or x3, x2, x1
2         lw x5, 0(x6)
3         sw x5, 4(x6)
4         xori x2, x5, 1
```

1-2: None, 0, None, 0
2-3: Data, 2, None, 0
3-4: None, 0, None, 0

Version 3

```
1         andi x2, x1, 0xF
2         bge x1, x2, label
3         xori x2, x2, 1
4     label:  or x3, x2, x1
```

1-2: Data, 2, Data, 2
2-3: Control, 2, Control, 3
3-4: Data, 2, None, 0

Version 4

```
1         andi x2, x1, 0xF
2         bge x1, x2, label
3         xori x2, x2, 1
4     label:  ori x3, x2, x1
```

1-2: Data, 2, Data, 2
2-3: Control, 2, Control, 3
3-4: Data, 2, None, 0

Version 5

```
1         addi x1, x0, 0xFF
2         andi x2, x1, 0xF
3         bge x1, x2, label
4         xori x2, x2, 1
```

1-2: None, 0, None, 0
2-3: Data, 2, None, 0
3-4: None, 0, None, 0

Version 6

```
1         ori x3, x2, x1
2         lw x5, 0(x6)
3         sw x5, 4(x6)
4         xori x2, x5, 1
```

1-2: Data, 2, None, 0
2-3: Data, 2, Data, 2
3-4: Control, 2, Control, 3

Retake Q5: RISC-V Datapath, Control, and Pipelining (*10 pts*)

Part A — 4 pts

The datapath in the Appendix implements the RV32I instruction set.

In the RISC-V datapath above, marked what is used by `INSTR` instruction.

Possible `INSTR`s: `blt`, `bne`, `addi`, `xori`, `lbu`, `sb`

Solutions

`blt`
PC Sel Mux: Input Dependent
ASel Mux: `pc` branch
BSel Mux: `imm` branch
WBSel Mux: `*` (don't care)
Immediate Type: `sb`
Datapath Units: Branch Comp, Imm Gen
Regfile: Read `Reg[rs1]`, Read `Reg[rs2]`

`bne`
PC Sel Mux: Input dependent
ASel Mux: `pc` branch
BSel Mux: `imm` branch
WBSel Mux: `*` (don't care)
Immediate Type: `sb`
Datapath Units: Branch Comp, Imm Gen
Regfile: Read `Reg[rs1]`, Read `Reg[rs2]`

`addi`
PC Sel Mux: `pc + 4` branch
ASel Mux: `Reg[rs1]` branch
BSel Mux: `imm` branch
WBSel Mux: `ALU` branch
Immediate Type: `i`
Datapath Units: Imm Gen
Regfile: Read `Reg[rs1]`, Write `Reg[rd]`

`xori`
PC Sel Mux: `pc + 4` branch
ASel Mux: `Reg[rs1]` branch
BSel Mux: `imm` branch
WBSel Mux: `ALU` branch
Immediate Type: `i`
Datapath Units: Imm Gen
Regfile: Read `Reg[rs1]`, Write `Reg[rd]`

`lbu`
PC Sel Mux: `pc + 4` branch
ASel Mux: `Reg[rs1]` branch
BSel Mux: `imm` branch
WBSel Mux: `mem` branch
Immediate Type: `i`
Datapath Units: Imm Gen, Load Extend
Regfile: Read `Reg[rs1]`, Write `Reg[rd]`

`sb`
PC Sel Mux: `pc + 4` branch
ASel Mux: `Reg[rs1]` branch
BSel Mux: `imm` branch
WBSel Mux: `*` (don't care)
Immediate Type: `s`
Datapath Units: Imm Gen
Regfile: Read `Reg[rs1]`, Read`Reg[rs2]`

Part B — 3 pts
Question and solutions same as original midterm versions

Part C — 3 pts
Consider the 5-stage pipelining presented in lecture with the combinational-read IMEM and DMEM and the forwarding paths as drawn in the pipelined diagram in the Appendix.

Write the number of NOPs needed between each instruction, and list the hazard that causes the stall. If no hazard occurs, select "None", and list the number of NOPs as 0. Assume you can write to and read from the same address in the register file (Reg [ ]) in the same cycle. If there is a branch, assume that it is not taken, and there is no branch prediction. Consider two cases:
Case 1: Forwarding is not implemented (the diagram as seen in lecture)
Case 2: The forwarding muxes in the diagram are driven correctly by the forwarding logic. Note: If a hazard is resolved by forwarding and no other hazard is present, select "None" for the hazard

There are 4 versions.

NOTE: there are alternative answers depending on what version of the diagram you used and which clarifications you received. Please submit a regrade request detailing which diagram you used and your reasoning for your answer if you believe your answer is correct.

Solutions
Instruction Set
Instruction Pair: Case 1 Hazard, Case 1 nops, Case 2 Hazard, Case 2 nops

Version 1

```
1         addi x1, x0, 0xFF
2         bge x1, x2, label
3          ori x2, x1, 0x7FF
4         xori x2, x2, 1
```

1-2: Data, 2, Data, 2
2-3: Control, 2, Control, 3
3-4: Data, 2, None, 0

Version 2

```
1         addi x1, x0, 0xFF
2         ori x2, x1, 0x7FF
3          bge x1, x2, label
4         xori x2, x2, 1
```

1-2: Data, 2, None, 0
2-3: Data, 2, Data, 2
3-4: Control, 2, Control, 3

Version 3

```
1         or x3, x2, x1
2           lw x5, 0(x6)
3         sw x5, 4(x6)
4         xori x2, x5, 1
```

1-2: None, 0, None, 0
2-3: Data, 2, None, 0
3-4: None, 0, None, 0

Version 4

```
1         andi x2, x1, 0xF
2         bge x1, x2, label
3         xori x2, x2, 1
4         label:  or x3, x2, x1
```

1-2: Data, 2, Data, 2
2-3: Control, 2, Control, 3
3-4: Data, 2, None, 0

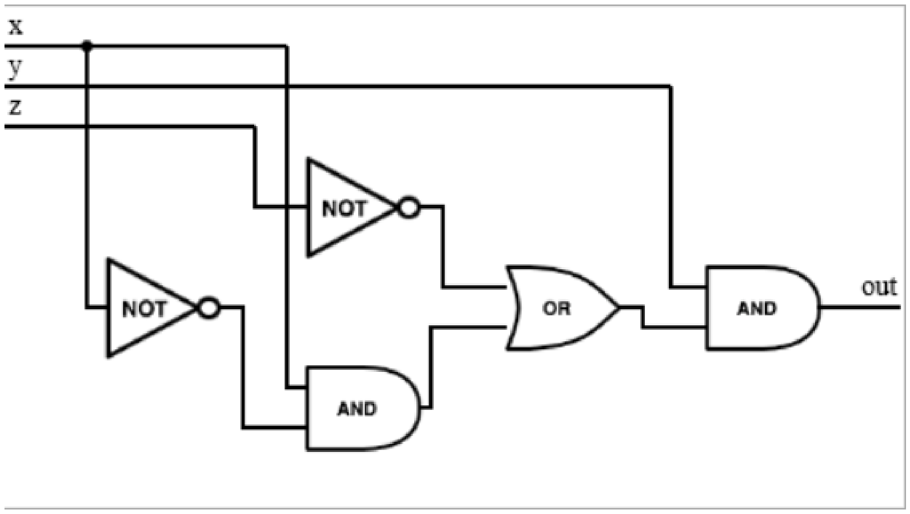
Appendix: Relevant Diagrams

Retake Q2: Quest Clobber

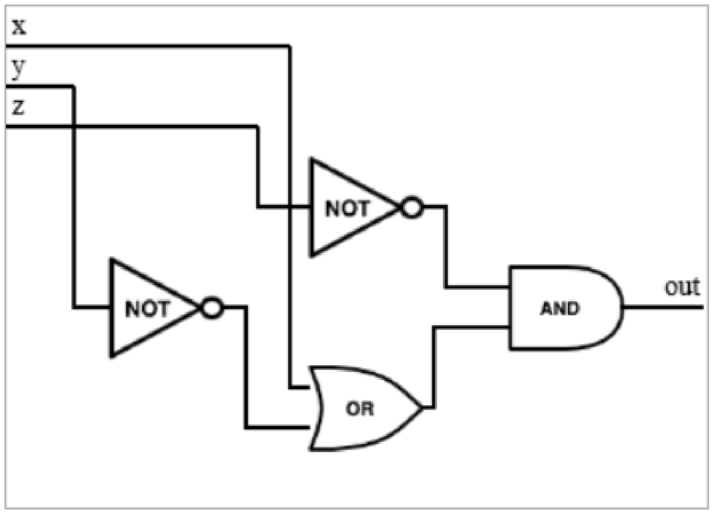
Binary Node Structure Reference

Q4: Truth Table

Version 1 Truth Table



Version 2 Truth Table



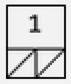
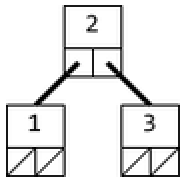
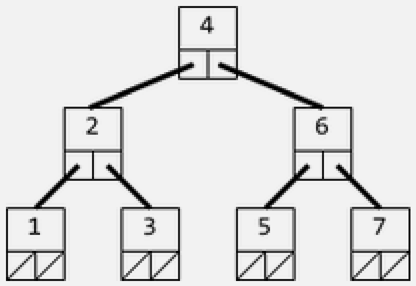
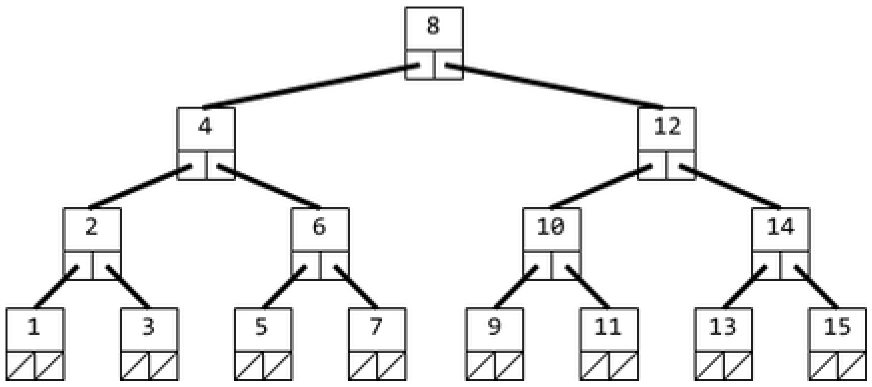
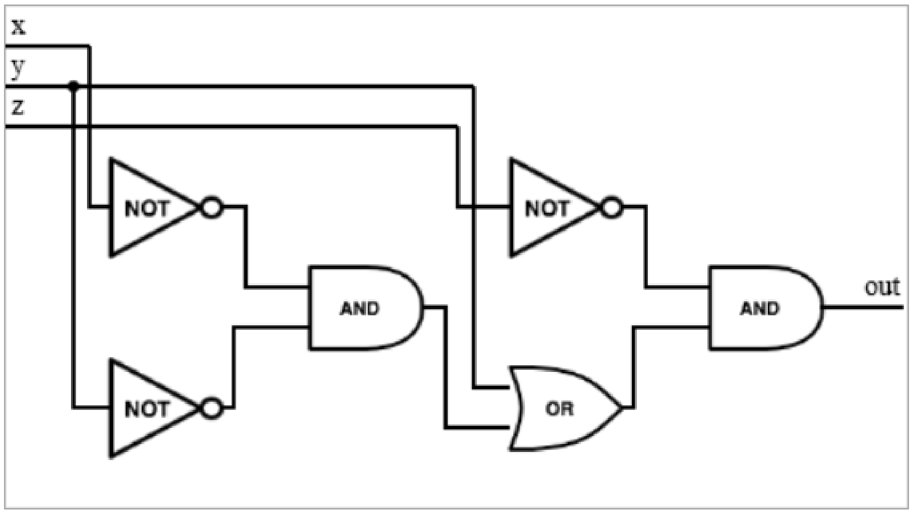
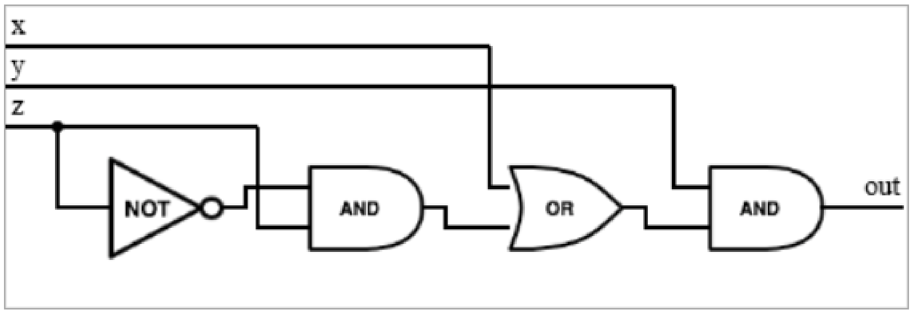
Call to `num_tree`	**Tree**
`num_tree(0)`	
`num_tree(1)`	
`num_tree(2)`	
`num_tree(3)`	
`num_tree(4)`	 8 4 12 2 6 10 14 1 3 5 7 9 11 13 15
...etc...	**...etc...**

Figure 1: retake-quest-clobber-bin-node-struct

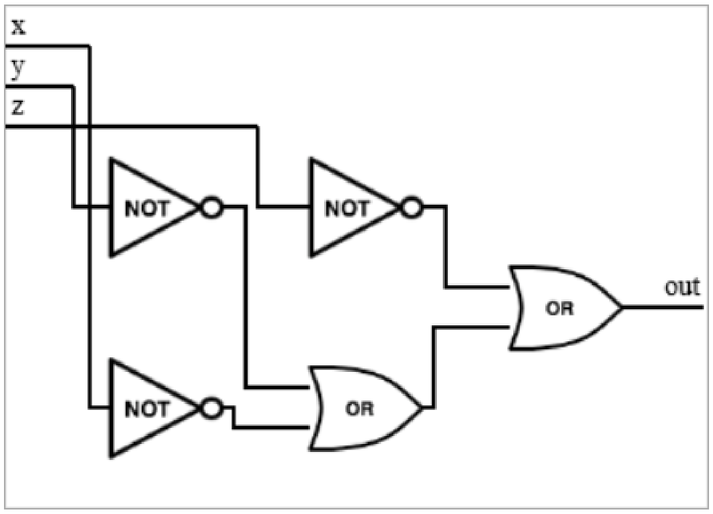
Version 3 Truth Table



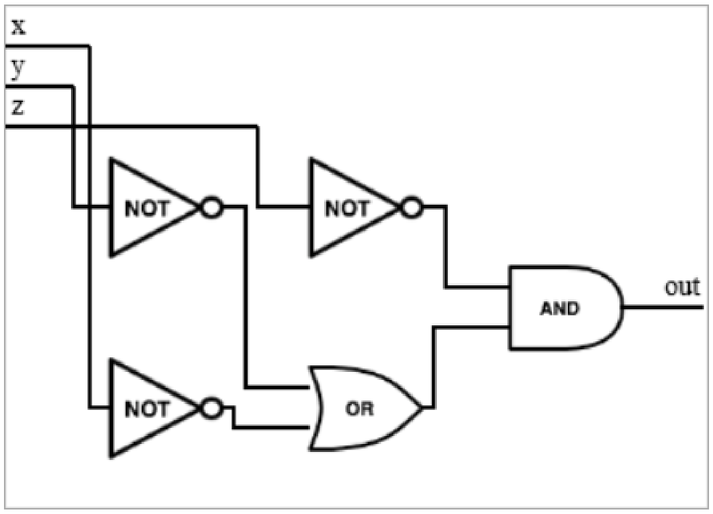
Version 4 Truth Table



Version 5 Truth Table

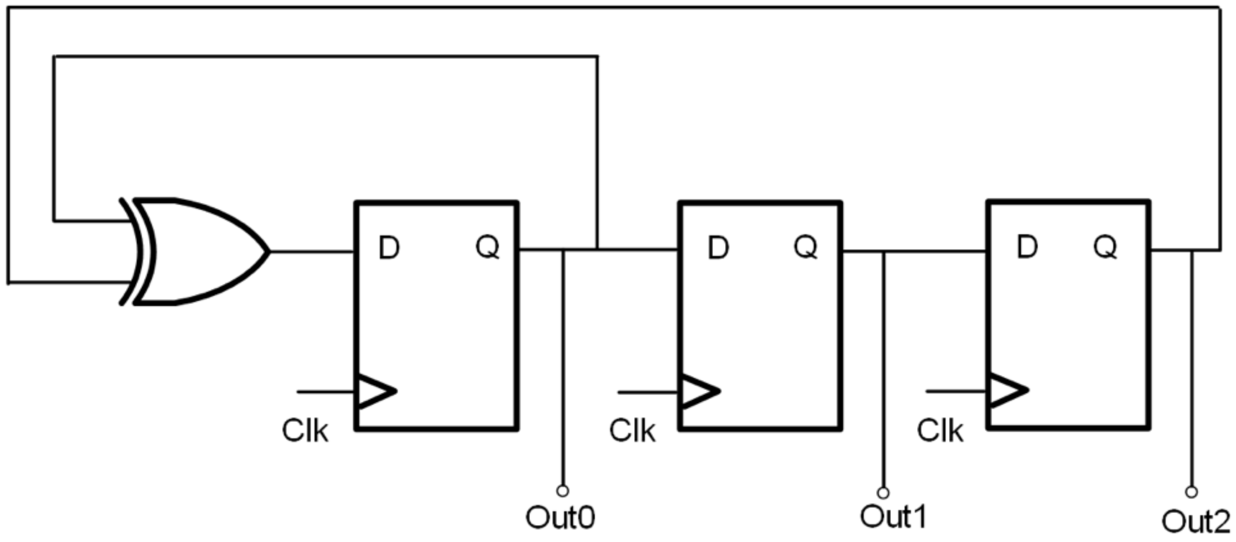


Version 6 Truth Table

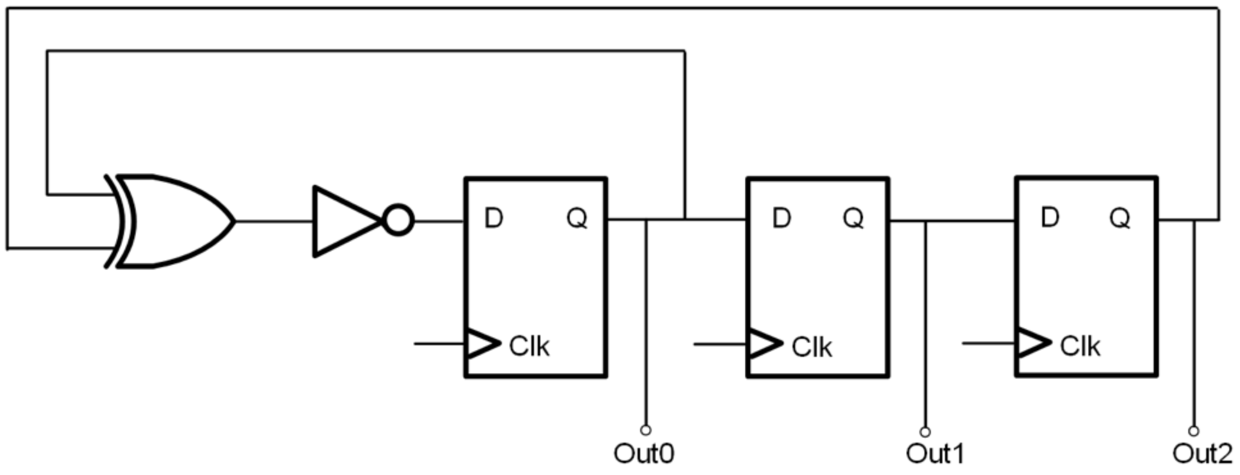


Q4: SDS

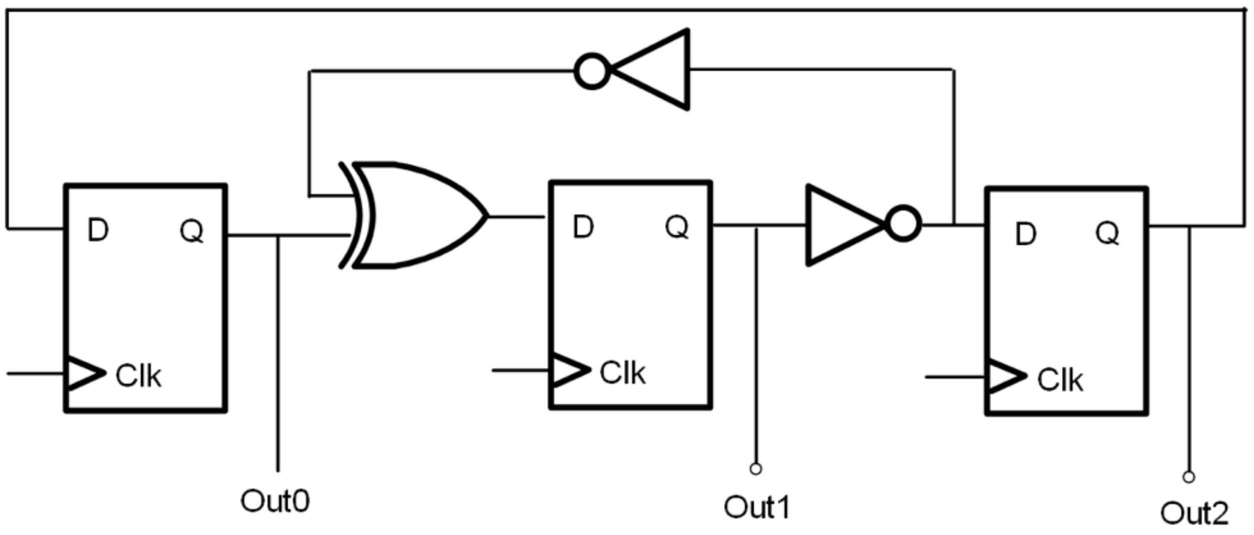
Version 1 SDS



Version 2 SDS

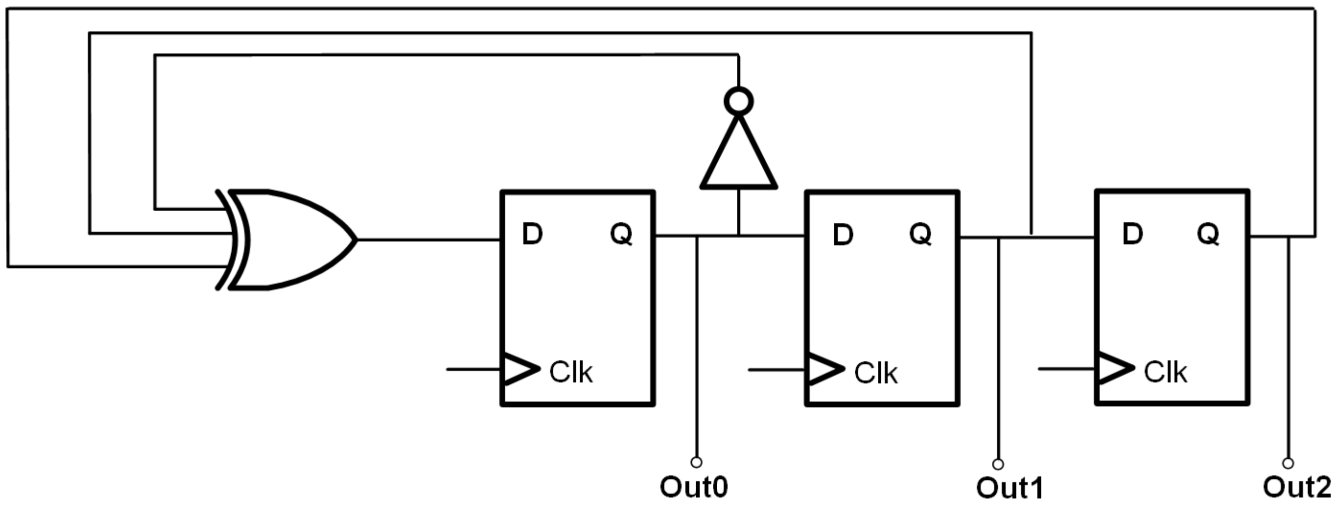


Version 3 SDS



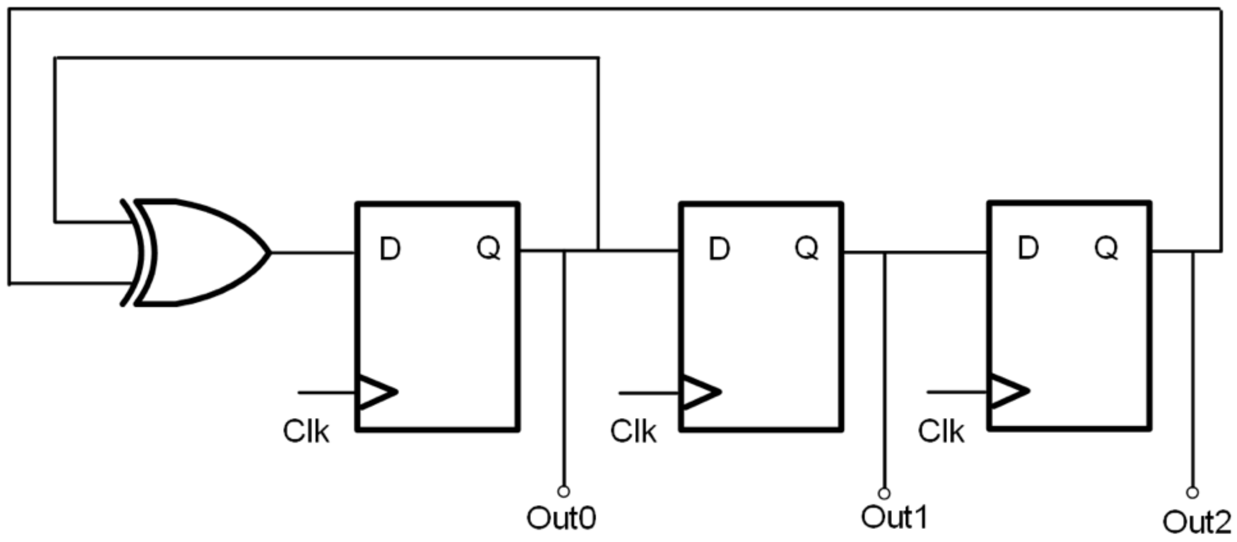
Retake Q4: SDS

Version 1 + 2 SDS (Same Diagram)

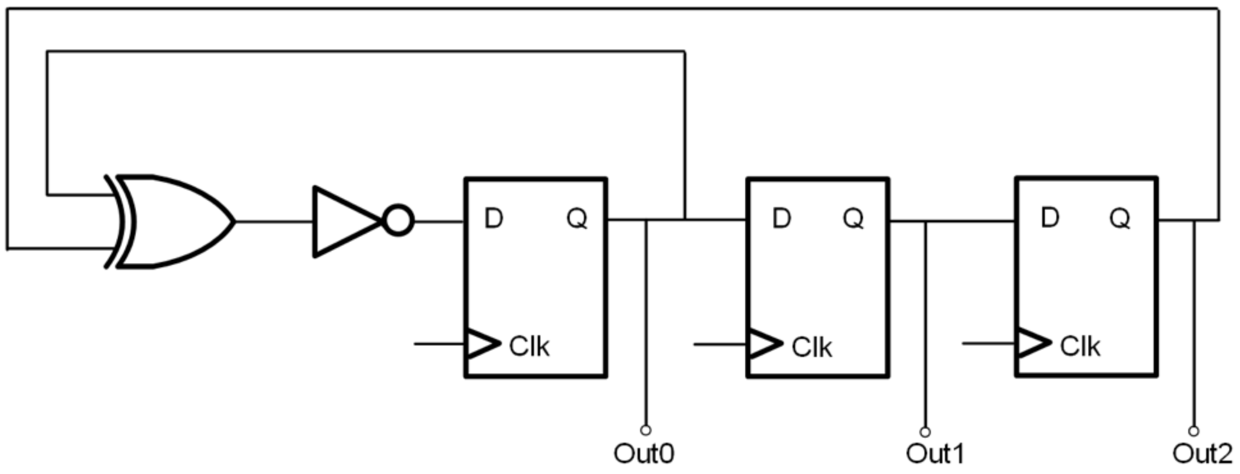


Q5: RISC-V Datapath, Pipelining, and Controls

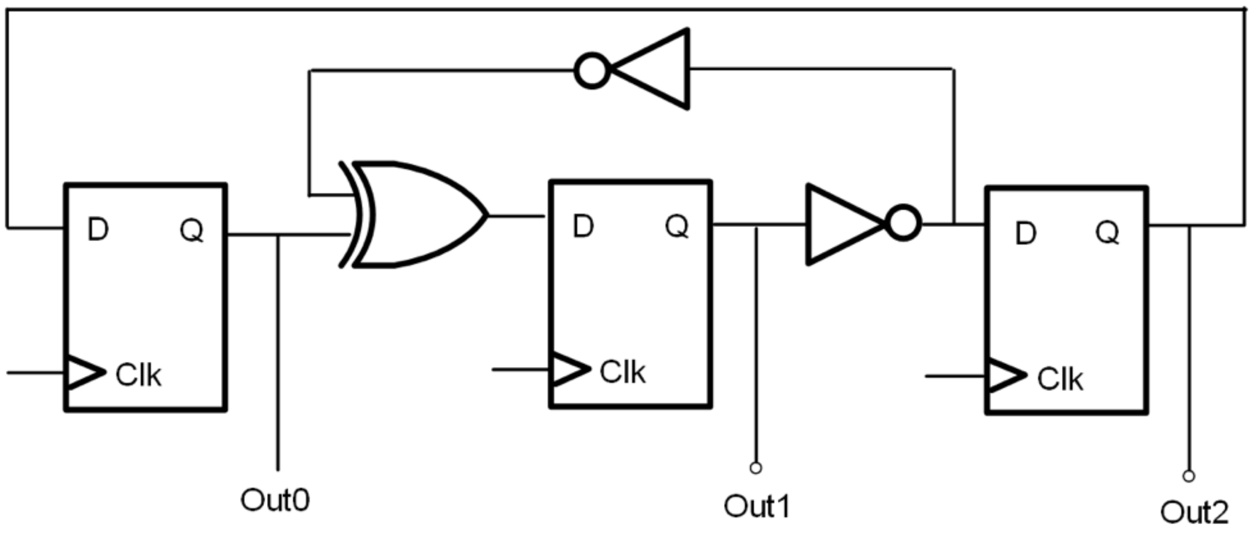
Version 1 Diagram



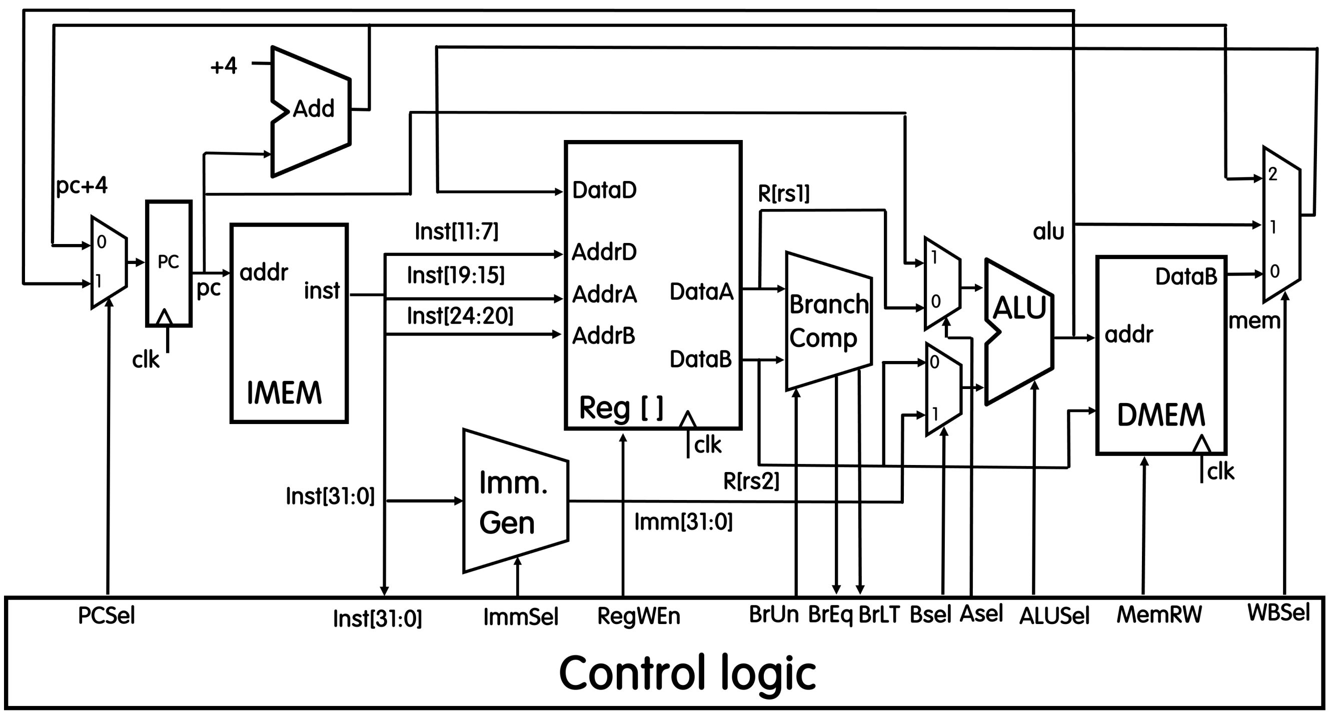
Version 2 Diagram



Version 3 Diagram



Datapath Updated



Pipelined Datapath

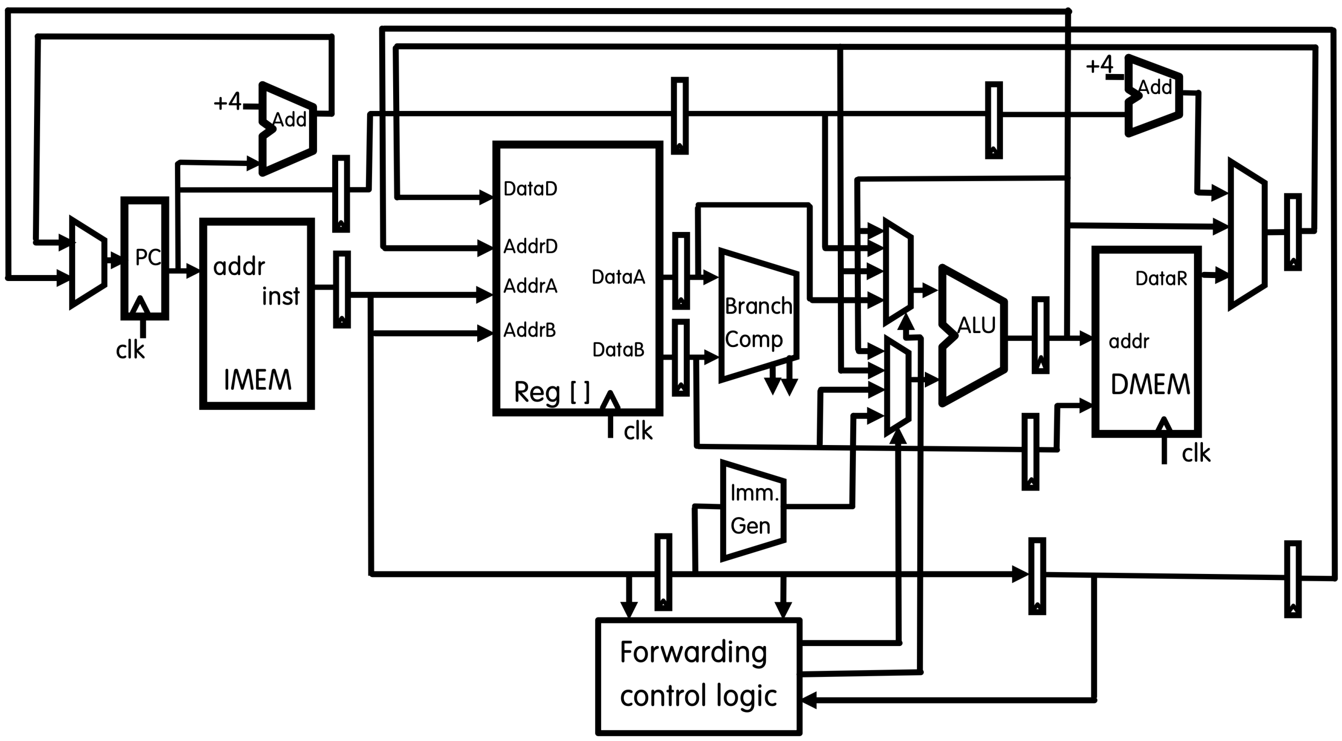


Figure 2: datapath-pipelined-updated