CS61C F20 Midterm Blank

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*)

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.**

Your Answer: `here`

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.

Your Answer: `here`

Retake Q1: Float (*5 pts*)

Same as original question, different parameters

Q2: Quest Clobber (*10 pts*)

Skeleton Code

Type Conversions

`quest.c` repeated per version

```
TREE *incr_tree(TREE *p) {
    return p; // <-- replace this with your code
}

void free_tree(TREE *p) {
    return; // <-- replace this with your code
}
```

bias2ones

```
#include "quest.h"

uint8_t bias2ones(uint8_t bias) {
    return bias + 1; // <-- replace this with your converter
}
```

bias2sm

```
#include "quest.h"

uint8_t bias2sm(uint8_t bias) {
    return bias + 1; // <-- replace this with your converter
}
```

ones2bias

```
#include "quest.h"

uint8_t ones2bias(uint8_t ones) {
    return ones + 1; // <-- replace this with your converter
}
```

sm2bias

```
#include "quest.h"

uint8_t sm2bias(uint8_t sm) {
    return sm + 1; // <-- replace this with your converter
}
```

Helper Files

`quest.h`

```
#include <inttypes.h>
#include <stdlib.h>
#include <stdio.h>

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

void *CS61C_malloc(size_t size);
void CS61C_free(void *ptr);
```

```
uint8_t <VERSION FUNCTION>(uint8_t <VERSION INPUT>);
TREE *map_tree(TREE *p);
void free_tree(TREE *p);

void print_tree(TREE *p);
```

print_tree.c

```
#include "quest.h"

void print_tree(TREE *p) {
    if (p) {
        printf("(");
        print_tree(p->L);
        printf(" %02X ", p->N);
        print_tree(p->R);
        printf(")");
    } else {
        printf(".");
    }
}
```

main.c

```
#include "quest.h"

void *CS61C_malloc(size_t size) {
    void *ptr = malloc(size);
    printf("malloc: %p\n",ptr);
    return ptr;
}

void CS61C_free(void *ptr) {
    printf("free  : %p\n",ptr);
    free(ptr);
}

int main(int argc, char *argv[])  {
    TREE a,b,c,d,e,*p;

    b.N = 0x0; // <-- PUT VALUES HERE to test your converter
    a.N = 0x1; //  a    <-- this is the tester tree we provide
    c.N = 0x2; // b c   <-- print_tree prints it like this
    e.N = 0x3; //    d  <-- ((. b .) a (. c ((. e .) d .)))
    d.N = 0x4; //   e   <-- in which each # is 2-digit Hex

    // Set up the tester tree for you, don't worry about this
    a.L = &b; a.R = &c;  c.R = &d;  d.L = &e;
    b.L = b.R = c.L = d.R = e.L = e.R = NULL;

    // Print the tree before map_tree, call map_tree and free it
    print_tree(&a); printf("\n");
    print_tree(p = map_tree(&a)); printf("\n");
    free_tree(p);
    return 0;
}
```

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.

```
uint8_t <VERSION FUNCTION>(uint8_t <VERSION INPUT>) {
    //YOUR ANSWER
}
```

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;

TREE *incr_tree(TREE *p) {
    //YOUR ANSWER
}
```

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.

```
void free_tree(TREE *p) {
    //YOUR ANSWER
}
```

Retake Q2: Quest Clobber (*10 pts*)

Skeleton Code

Type Conversions

`quest.c` repeated per version

```
#include "quest.h"

int count = 1; // a global variable for the node number

TREE *num_tree(int n) {
    TREE *t;
    return t; // <-- replace this with your code
}

void free_tree(TREE *p) {
    return; // <-- replace this with your code
}
```

ones2sm

```
uint8_t ones2sm(uint8_t ones) {
    return ones + 1; // <-- replace this with your converter
}
```

sm2ones

```
uint8_t sm2ones(uint8_t sm) {
    return sm + 1; // <-- replace this with your converter
}
```

Helper Files

`quest.h`

```
#include <inttypes.h>
#include <stdlib.h>
#include <stdio.h>

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

void *CS61C_malloc(size_t size);
void CS61C_free(void *ptr);

uint8_t <VERSION_NAME>(uint8_t <VERSION_INPUT>);
TREE *num_tree(int n);
void free_tree(TREE *p);

void print_tree(TREE *p);
```

`print_tree.c`

```
#include "quest.h"

void print_tree(TREE *p) {
    if (p) {
        printf("(");
```

```
        print_tree(p->L);
        printf(" %d ", p->N);
        print_tree(p->R);
        printf(")");
    } else {
        printf(".");
    }
}
```

main.c

```
#include "quest.h"

void *CS61C_malloc(size_t size) {
    void *ptr = malloc(size);
    printf("malloc: %p\n",ptr);
    return ptr;
}

void CS61C_free(void *ptr) {
    printf("free  : %p\n",ptr);
    free(ptr);
}

int main(int argc, char *argv[])  {
    TREE *hardcoded_num_tree_3, *t;
    TREE *n1,*n2,*n3,*n4,*n5,*n6,*n7;

    // Part A tests:
    printf("<VERSION_NAME>(0) = %u\n", <VERSION_NAME>(0));
    printf("<VERSION_NAME>(1) = %u\n", <VERSION_NAME>(1));
    printf("<VERSION_NAME>(2) = %d\n", <VERSION_NAME>(2));
    printf("<VERSION_NAME>(3) = %d\n", <VERSION_NAME>(3));
    printf("<VERSION_NAME>(4) = %d\n", <VERSION_NAME>(4));

    // Hardcoded num_tree(3) for you, use it to compare
    // with your own num_tree, and to test free_tree.
    n1 = (TREE *) CS61C_malloc (sizeof (TREE));
    n2 = (TREE *) CS61C_malloc (sizeof (TREE));
    n3 = (TREE *) CS61C_malloc (sizeof (TREE));
    n4 = (TREE *) CS61C_malloc (sizeof (TREE));
    n5 = (TREE *) CS61C_malloc (sizeof (TREE));
    n6 = (TREE *) CS61C_malloc (sizeof (TREE));
    n7 = (TREE *) CS61C_malloc (sizeof (TREE));
    n1->N = 1; n2->N = 2; n3->N = 3; n4->N = 4; n5->N = 5; n6->N = 6; n7->N = 7;
    n1->L = n1->R = n3->L = n3->R = n5->L = n5->R = n7->L = n7->R = NULL;
    n2->L = n1; n2->R = n3; n6->L = n5; n6->R = n7; n4->L = n2; n4->R = n6;
    hardcoded_num_tree_3 = n4;

    // Print the tree
    print_tree(hardcoded_num_tree_3); printf("\n");

    // Part B
    // When you have authored num_tree, comment out the line above
    // (and the 11-line hardcoded tree above that) and uncomment the line below
    // ...oh, by the way you should make sure it works for values other than 3...
    // print_tree(t = num_tree(3)); printf("\n");
```

```
    // Part C test...free it
    free_tree(hardcoded_num_tree_3);

    return 0;
}
```

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

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;
```

```
TREE *num_tree(int n) {
    //YOUR ANSWER
}
```

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.

```
void free_tree(TREE *p) {
    //YOUR ANSWER
}
```

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.**

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

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.

Provided Files

Unique `AfterString` and `ReturnValueString` Per Version

InvertLen.s
```
AfterString:   .asciiz "After calling, it should be 'cAL'. It is: "
ReturnValueString:   .asciiz "Return Value should be 3. It is: "
```

UpperSum.s
```
AfterString:   .asciiz "After calling, it should be 'CAL'. It is: "
ReturnValueString:   .asciiz "Return Value should be 272. It is: "
```

UpperLen.s
```
AfterString:   .asciiz "After calling, it should beCAL. It is: "
ReturnValueString:   .asciiz "Return Value should be 3. It is: "
```

LowerSum.s
```
AfterString:   .asciiz "After calling, it should be 'cal'. It is: "
ReturnValueString:   .asciiz "Return Value should be 272. It is: "
```

LowerLen.s
```
AfterString:   .asciiz "After calling, it should be 'cal'. It is: "
ReturnValueString:   .asciiz "Return Value should be 3. It is: "
```

InvertSum.s
```
AfterString:   .asciiz "After calling, it should be 'cAL'. It is: "
ReturnValueString:   .asciiz "Return Value should be 272. It is: "
```

<VERSION_NAME.s>

```
.data

InputString:   .asciiz "Cal"

BeforeString:   .asciiz "Input should be 'Cal', and it is: "

AfterString:   .asciiz "After calling, it should be <SEE ABOVE>. It is: "

ReturnValueString:   .asciiz "Return Value should be <SEE ABOVE>. It is: "

.text

main:
    #Feel free to edit this for your own tests, but this part will not be graded
    #Print String before running code
    la a0 BeforeString
    jal print_str
    la a0 InputString
    jal print_str
```

```
    jal print_newline
    #Run <VERSION_NAME>
    la a0 InputString
    jal <VERSION_NAME>
    mv s0 a0
    #Print String after running code
    la a0 AfterString
    jal print_str
    la a0 InputString
    jal print_str
    jal print_newline
    #Print return value
    la a0 ReturnValueString
    jal print_str
    mv a0 s0
    jal print_int
    jal print_newline
    #Exits the program
    li a0 10
    ecall



<VERSION_NAME>:
    #Reads through the string, and converts the entire string to <VERSION PARAMETERS> case
    #Returns the length/sum of the string for version
    #YOUR CODE HERE
    li a0 0
    jr ra


###########################################
#Utility Functions
###########################################
print_int:
    mv a1, a0
    li a0, 1
    ecall
    jr    ra

print_str:
    mv a1, a0
    li a0, 4
    ecall
    jr    ra

print_newline:
    li a1, '\n'
    li a0, 11
    ecall
    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`.

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.

Provided Files

Unique `AfterString` and `ReturnValueString` Per Version

LowerSearch.s
```
AfterString:   .asciiz "After calling, it should be 'ucberkeleyGObears'. It is: "
```

InvertSearch.s
```
AfterString:   .asciiz "After calling, it should be 'ucbErkeleyGObears'. It is: "
```

UpperSearch.s
```
BeforeString:   .asciiz "Input should be 'UCBerkeleyGObears', and it is: "
```

<VERSION_NAME.s>

```
.data

InputString:   .asciiz "UCBerkeleyGObears"

InputChar: .asciiz "r"

BeforeString:   .asciiz "Input should be 'UCBerkeleyGObears', and it is: "

AfterString:   .asciiz "After calling, it should be <SEE ABOVE>. It is: "

ReturnValueString:   .asciiz "Return Value should be 1. It is: "

.text

main:
    #Feel free to edit this for your own tests, but this part will not be graded
    #Print String before running code
    la a0 BeforeString
    jal print_str
    la a0 InputString
    jal print_str
    jal print_newline
    #Run <VERSION NAME>
    la a0 InputString
    la a1 InputChar
    lb a1 0(a1)
    jal <VERSION NAME>
    mv s0 a0
    #Print String after running code
    la a0 AfterString
    jal print_str
    la a0 InputString
    jal print_str
    jal print_newline
```

```
    #Print return value
    la a0 ReturnValueString
    jal print_str
    mv a0 s0
    jal print_int
    jal print_newline
    #Exits the program
    li a0 10
    ecall



<VERSION NAME>:
    #Reads through the string in a0, and searches for the character in a1.
    #<DOES THE INTENDED OPERATION>
    #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.
    #YOUR CODE HERE
    li a0 0
    jr ra


###########################################
#Utility Functions
###########################################
print_int:
    mv a1, a0
    li a0, 1
    ecall
    jr    ra

print_str:
    mv a1, a0
    li a0, 4
    ecall
    jr    ra

print_newline:
    li a1, '\n'
    li a0, 11
    ecall
    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.

See Appendix for Version Diagrams

All Version Inputs

x	y	z	out
0	0	0	?
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

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.

See Appendix for Version Diagrams

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

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

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

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?

Your Answer: `here`

See Appendix for Version Diagrams

Part C* — *2 pts In the version diagram in the Appendix's figure, what is the longest hold time for the flip-flop that allows for correct operation?

Your Answer: `here`

Retake Q4: SDS (*10 pts*)

Truth Table ***4 pts***

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

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

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

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?

Your Answer: `here`

See Appendix for Version Diagrams

Part C — 2 pts In the version diagram in the Appendix's figure, what is the longest hold time for the flip-flop that allows for correct operation?

Your Answer: `here`

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 marked in the Appendix, marked what is used by `INSTR` instruction.

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

Datapath Components

PC Sel Mux

- `ALU` Branch
- `*` (don't care)
- Input Dependent
- `pc + 4` branch

ASel Mux

- `Reg[rs1]` branch
- `pc` branch
- `*` (don't care)

BSel Mux

- `Reg[rs2]` branch
- `imm` branch
- `*` (don't care)

WBSel Mux

- `ALU` Branch
- `*` (don't care)
- `mem` branch
- `pc + 4` branch

Select all that apply for the following

Datapath Units

- Load Extend
- Branch Comp
- Imm. Gen

Regfile

- Write `Reg[rd]`
- 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`
  - `R[rd] = M[R[rs1] + imm] + R[rs2]`
- `beq` with writeback
  - `beq rd, rs1, rs2, imm`
  - `R[rd] = R[rs1] + R[rs2] if (R[rs1] == R[rs2]) else PC = PC + {imm, 1'b0}`
- PC-relative load
  - `lwpc rd, imm`

  - R[rd] = M[PC + imm]
- Register offset load
  - lwreg rd, rs1, rs2
  - R[rd] = M[R[rs1] + R[rs2]]
- Jump to zero
  - jzero rs1, rs2
  - if (R[rs1] == R[rs2]) then PC = 0
- Negate
  - neg rd, rs1
  - R[rd] = -R[rs1]

Instruction	Description	Implementable?	Control Signals
<Instruction Formatting>	**<Instruction Description>**	YOUR ANSWER	**ASel**: YOUR ANSWER, **BSel**: YOUR ANSWER

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

General Answer Space

Instruction	Case 1 Hazard	Case 1 Nops	Case 2 Hazards	Case 2 Nops
1-2	Data, Control, Structural, None	YOUR ANSWER	Data, Control, Structural, None	YOUR ANSWER
2-3	Data, Control, Structural, None	YOUR ANSWER	Data, Control, Structural, None	YOUR ANSWER
3-4	Data, Control, Structural, None	YOUR ANSWER	Data, Control, Structural, None	YOUR ANSWER

Versioning

Version 1

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

Version 2

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

Version 3

```
1         andi x2, x1, 0xF
```

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

Version 4

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

Version 5

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

Version 6

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

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 marked in the Appendix, marked what is used by `INSTR` instruction.

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

Datapath Components

Same as original midterm selections

Additional Question

Which type of immediate is used by this instruction?

- `i` —immediate instruction immediate
- `s`—store instruction immediate
  - Correction: certain versions should be `sb` (branch type instruction immediate)
- `u` —upper immediate instruction immediate
- `j` —jump instruction immediate

Part B — 3 pts All versions of this subsection are identical to the original midterm

Part C — 3 pts Question setup is identical to original midterm; versioning has changed

Versioning

Version 1

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

Version 2

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

Version 3

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

Version 4

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

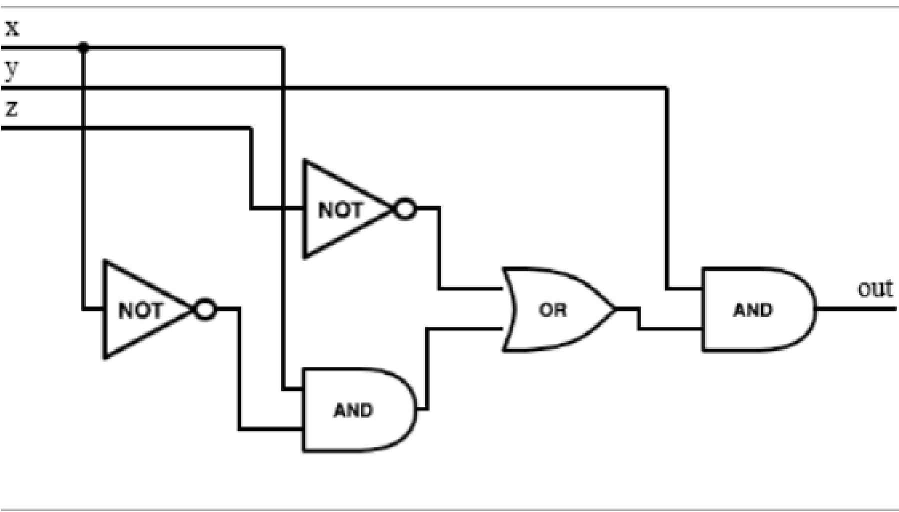
Appendix: Relevant Diagrams

Retake Q2: Quest Clobber

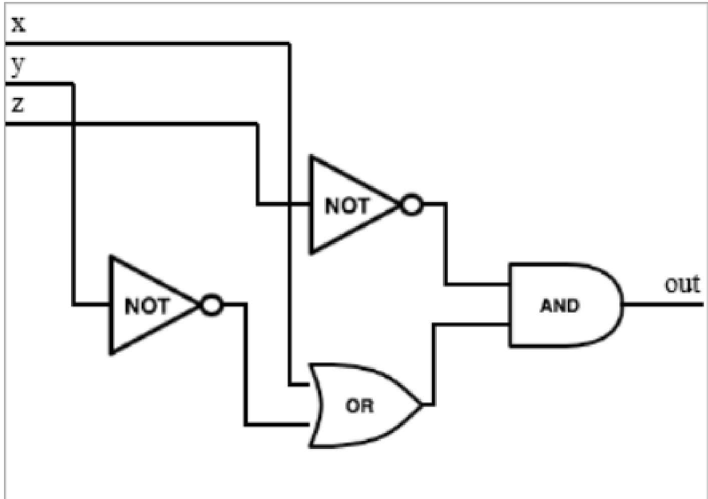
Binary Node Structure Reference

Q4: Truth Table

Version 1 Truth Table



Version 2 Truth Table



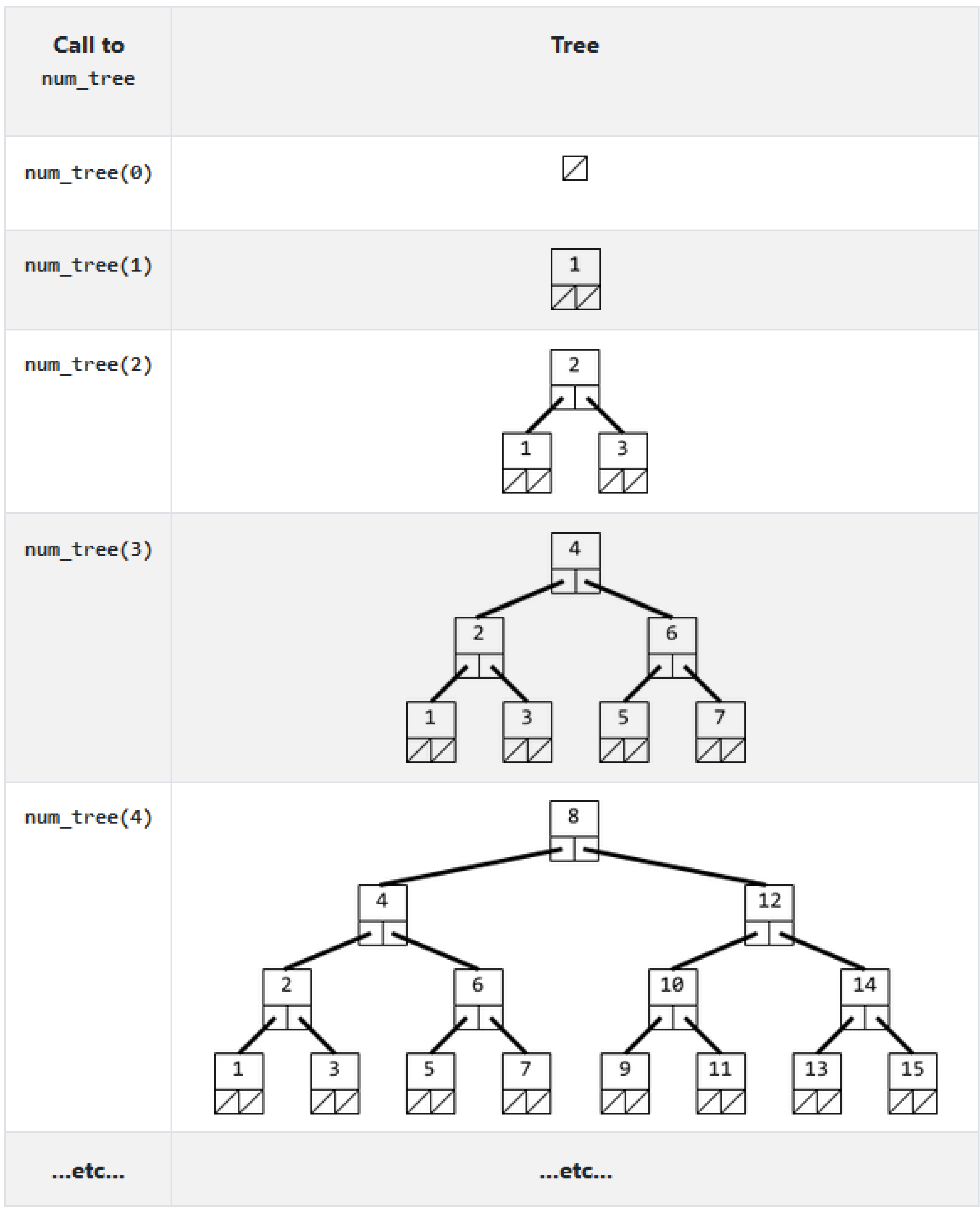
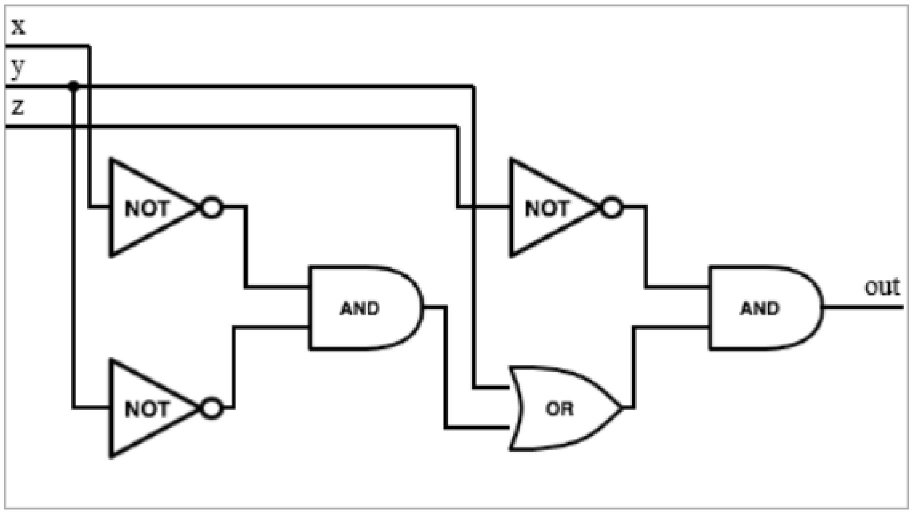
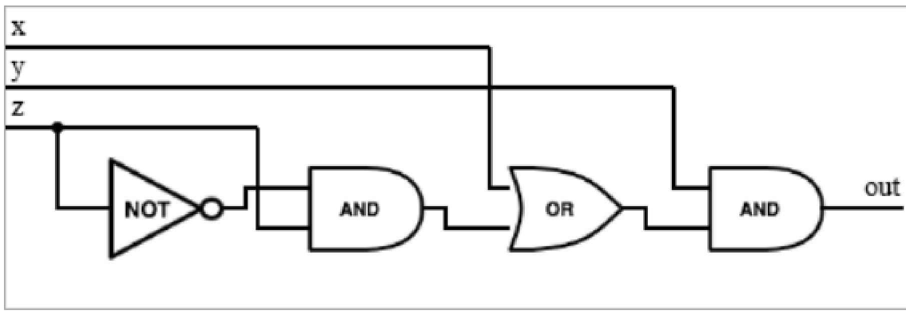


Figure 1: retake-quest-clobber-bin-node-struct

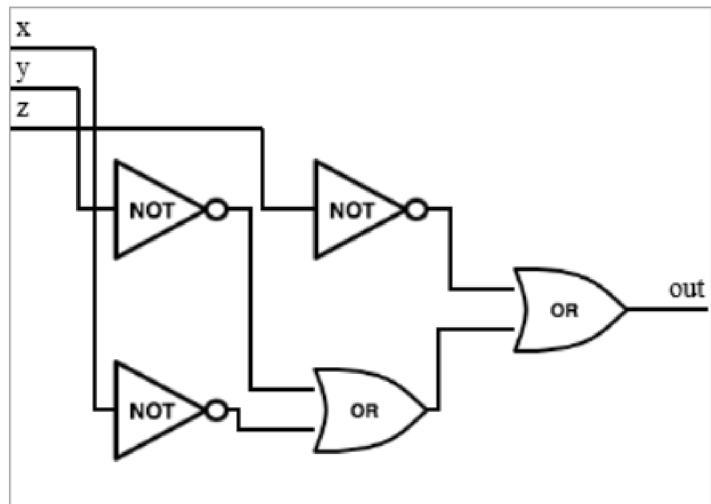
Version 3 Truth Table



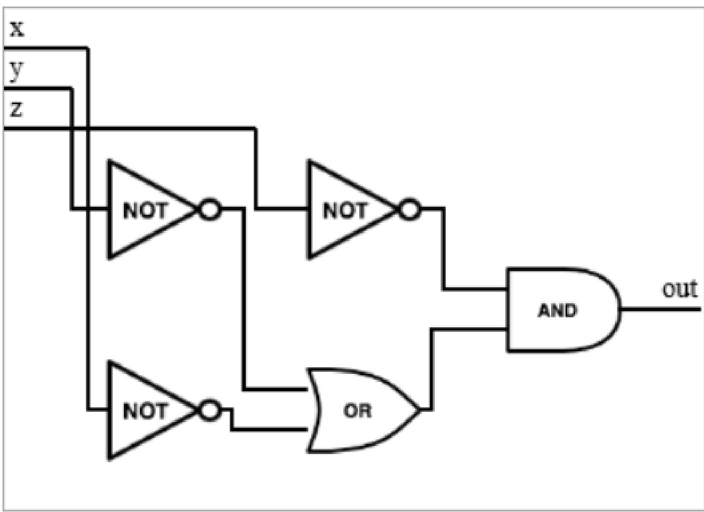
Version 4 Truth Table



Version 5 Truth Table

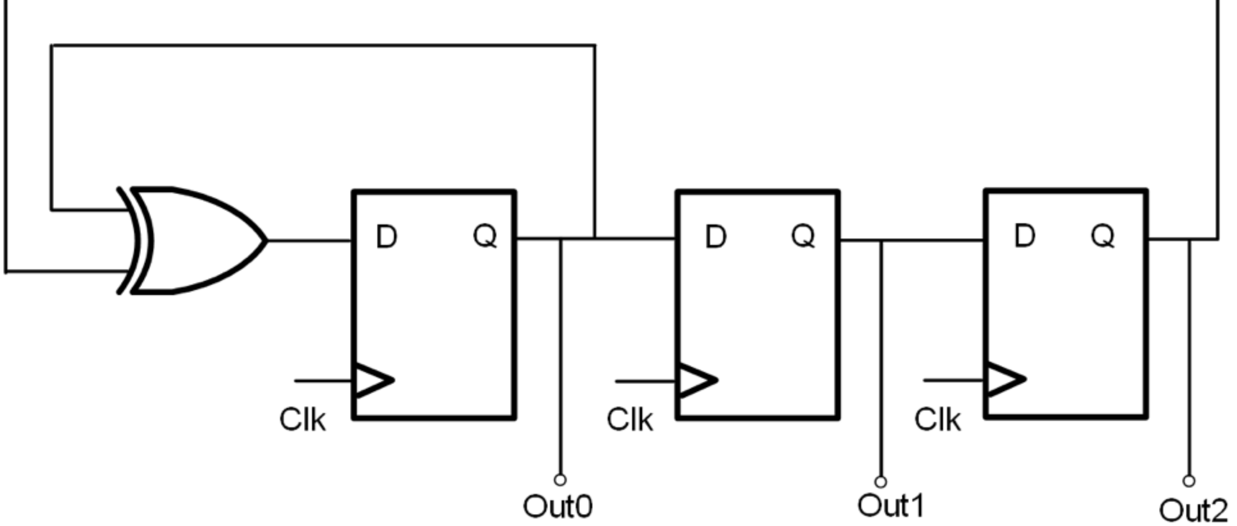


Version 6 Truth Table

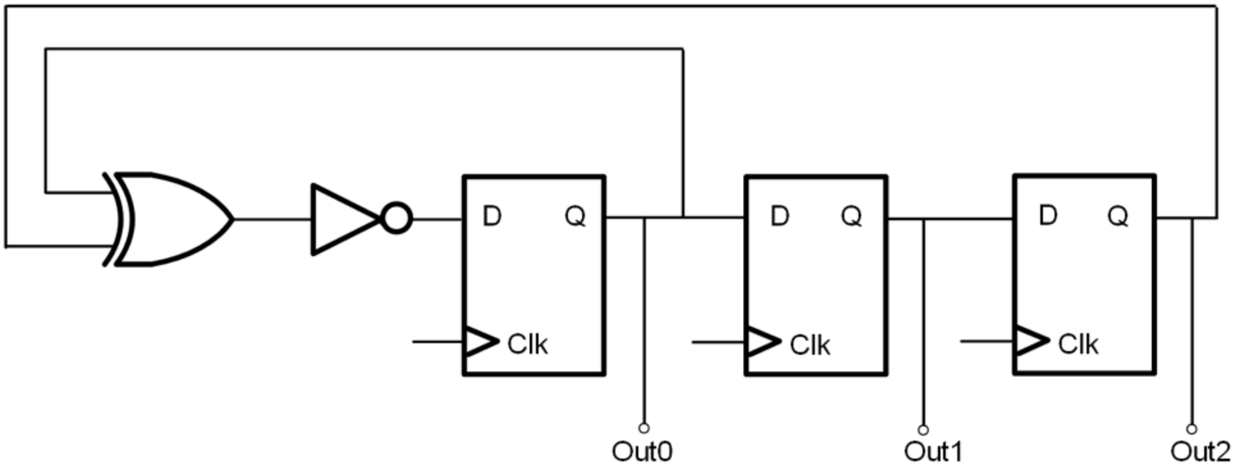


Q4: SDS

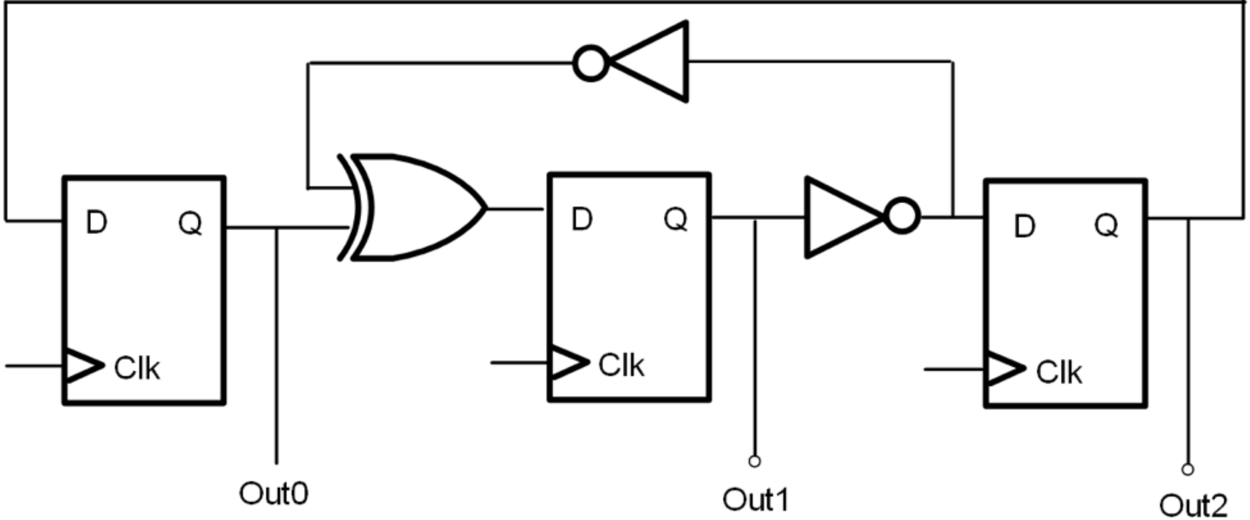
Version 1 SDS



Version 2 SDS

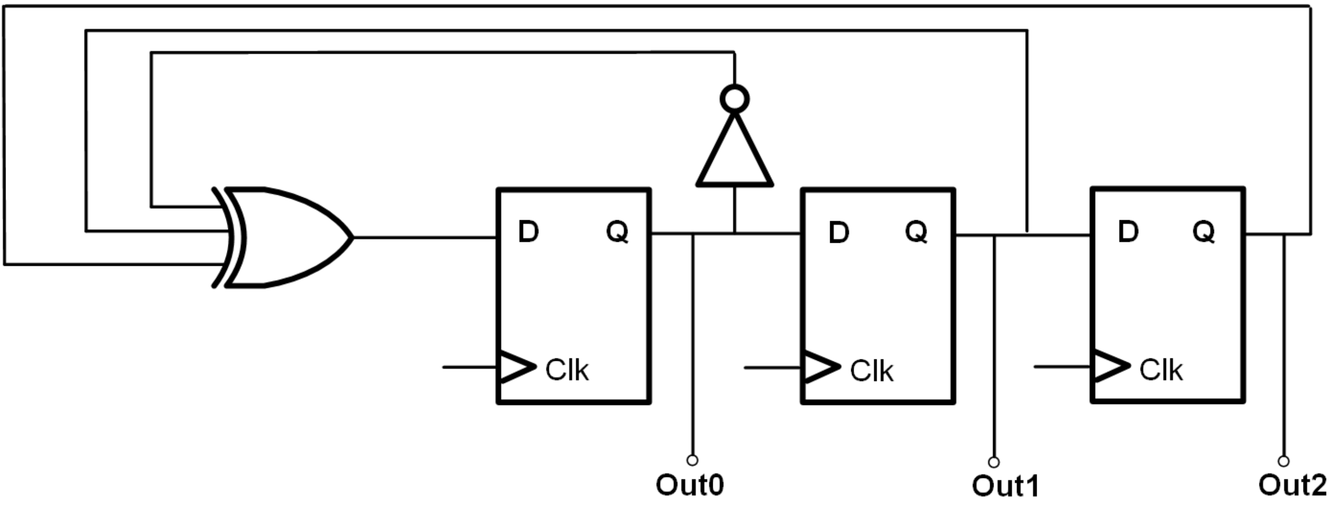


Version 3 SDS



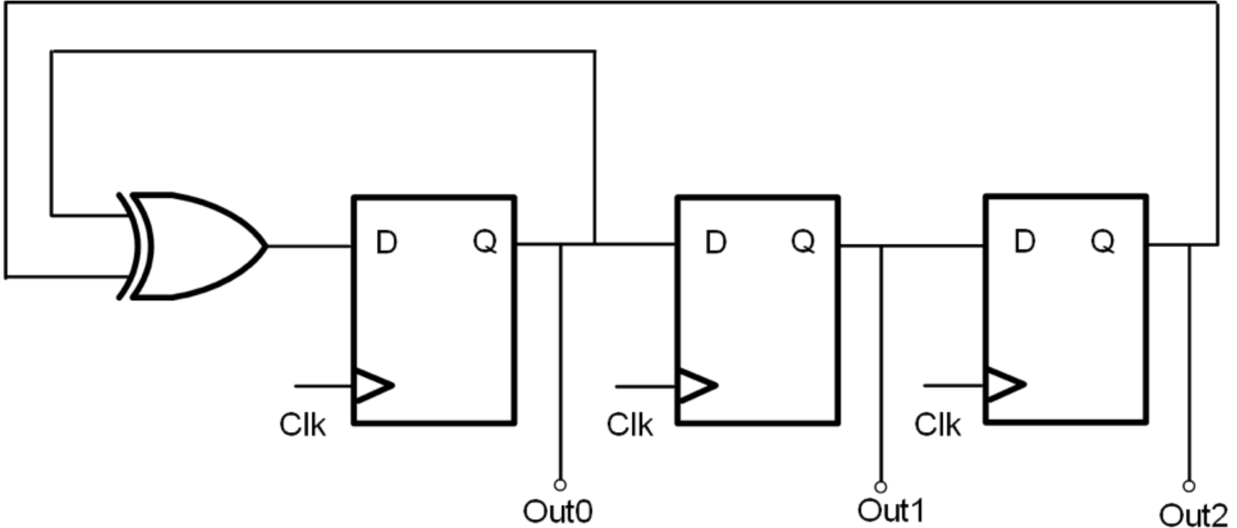
Retake Q4: SDS

Version 1 + 2 SDS (Same Diagram)

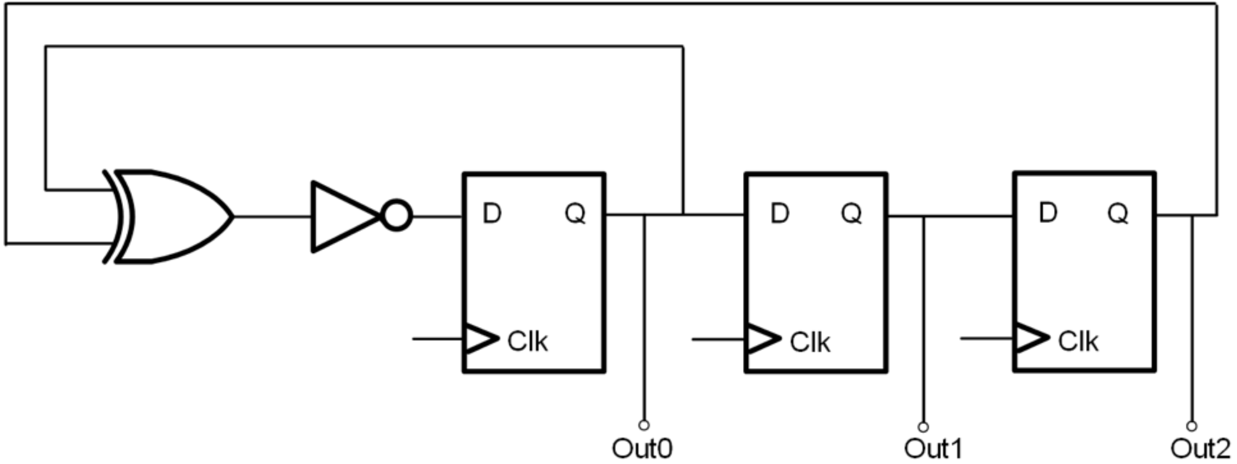


Q5: RISC-V Datapath, Pipelining, and Controls

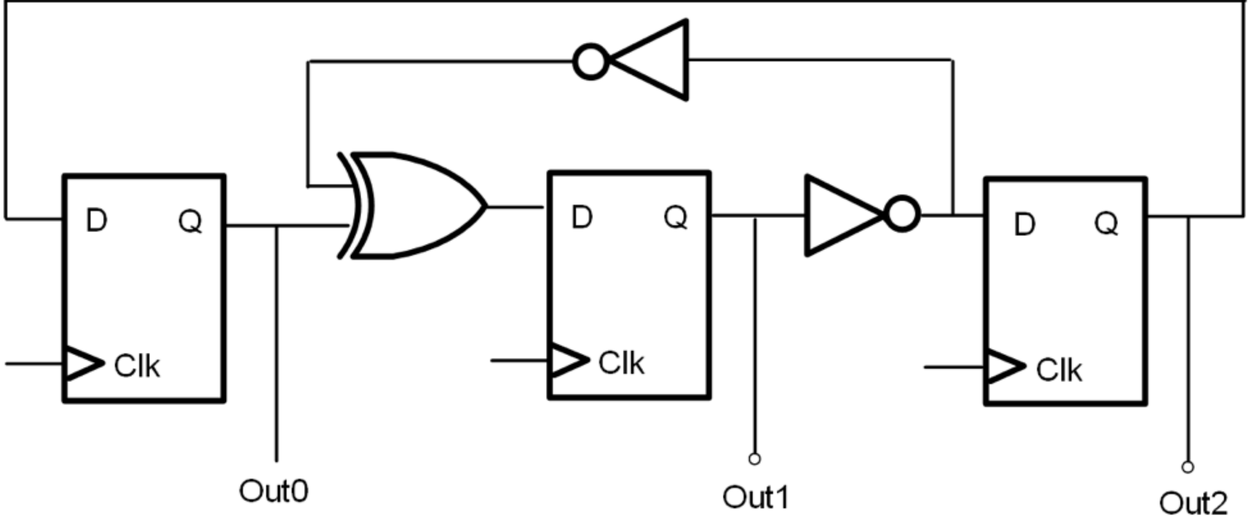
Version 1 Diagram



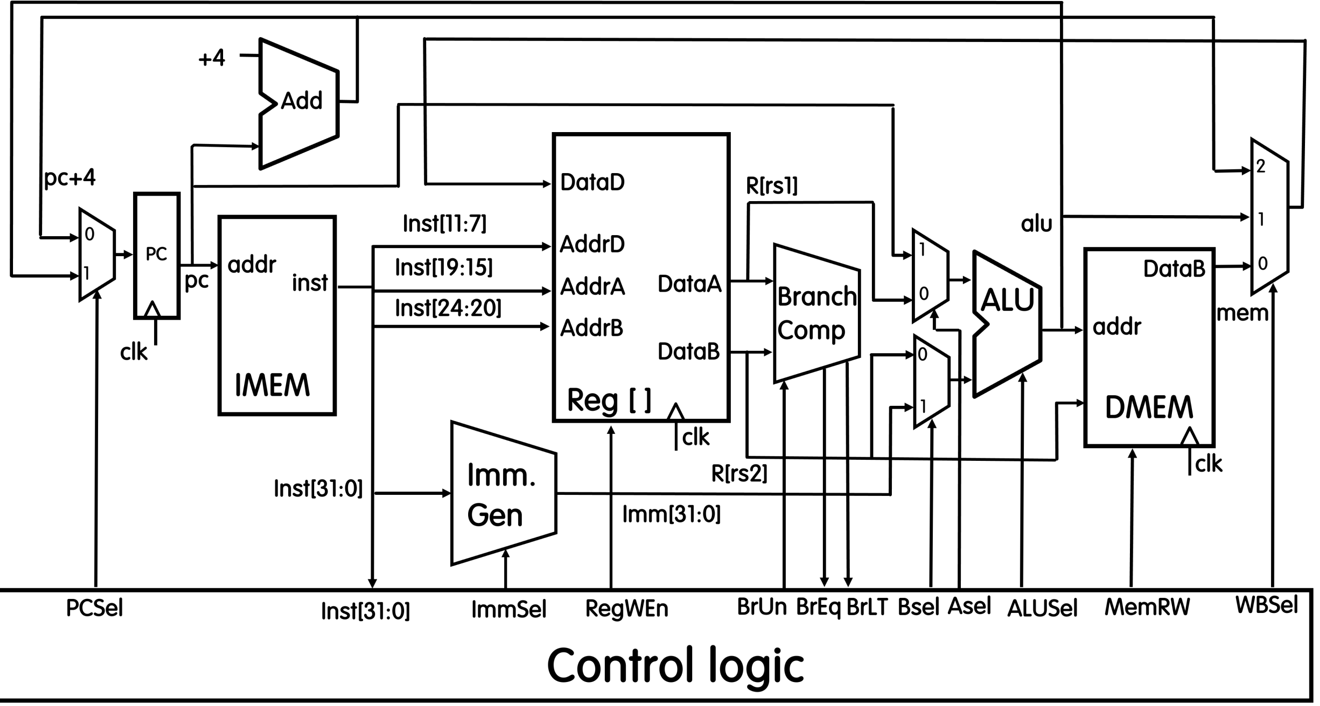
Version 2 Diagram



Version 3 Diagram



Datapath Updated



Pipelined Datapath

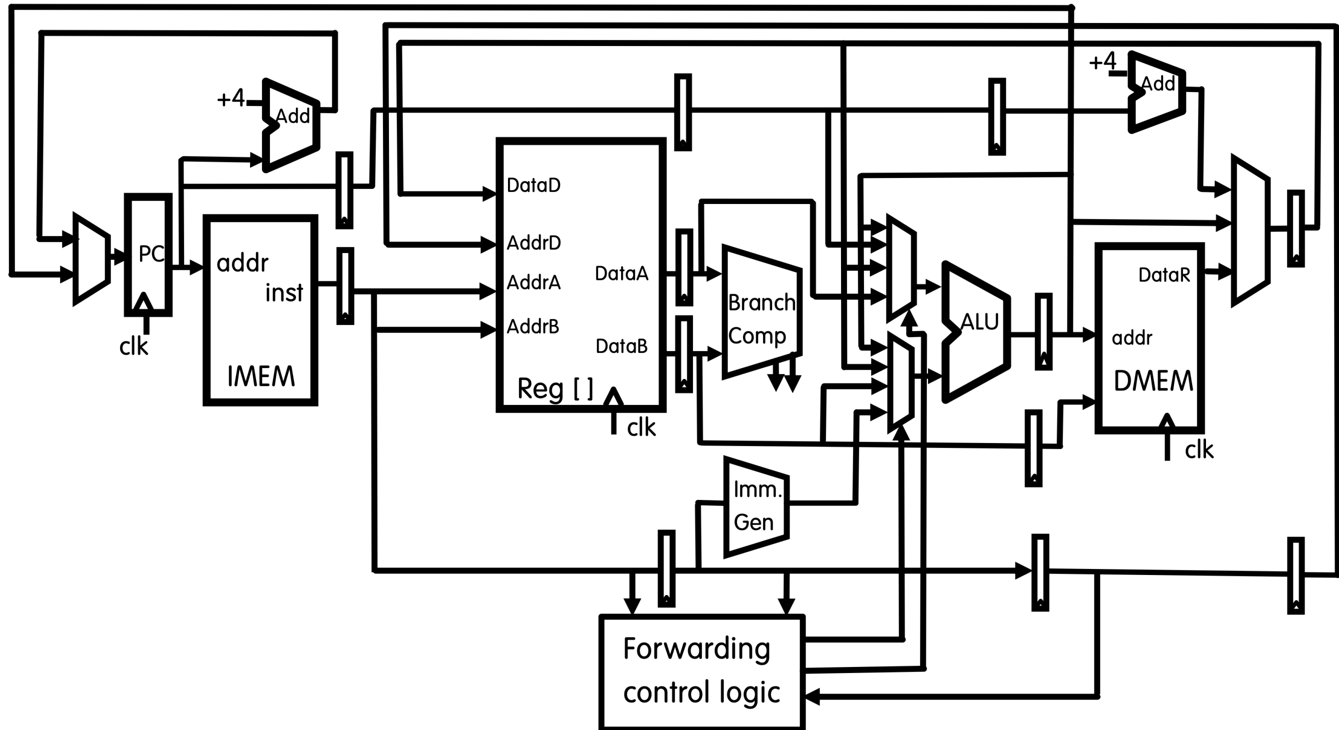


Figure 2: datapath-pipelined-updated