Quiz 4

Note: Cannot bypass a MEM stage if hw/sw

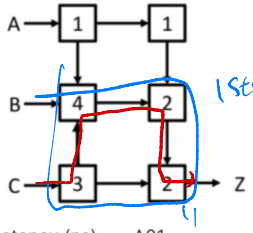
2020

Question A

Note the throughput part

For each of the questions below, create a valid N-stage pipeline of the given circuit. Each component in the circuit is annotated with its propagation delay. Give the latency and throughput of each design, assuming ideal registers (t_{PD} = 0, t_{SETUP} = 0). Remember that our convention is to place a pipeline register on each output.

(1) Show the maximum-throughput 1-stage pipeline.



1 stage

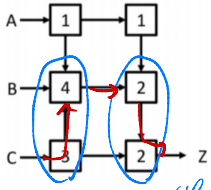
3+4+2+2 = 11

Latency (ns): __ A01 __

Throughput (ns^{-1}): __ A02 __ $\frac{1}{11}$

- A01 = ?
- A02 = ?

(2) Show the maximum-throughput 2-stage pipeline using a minimal number of registers.



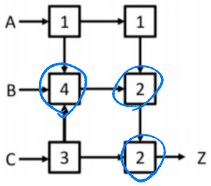
(4+3)·2 = 14

Latency (ns): __ A03 __ 14

Throughput (ns^{-1}): __ A04 __ $\frac{1}{7}$

- A03 = ?
- A04 = ?

(3) Show the maximum-throughput pipeline using a minimal number of registers.



(A & 4th reg.)

6×3 = 12

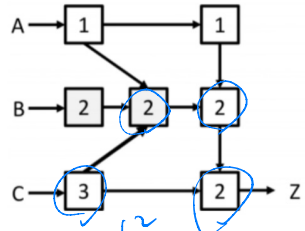
3 stage

Latency (ns): __ A05 __ 12

Throughput (ns^{-1}): __ A06 __ $\frac{1}{4}$

- A05 = ?
- A06 = ?

(4) You manage to reimplement the slowest combinational component in the previous circuit (the one with a propagation delay of 4 ns) using two components with propagation delays of 2 ns, as shown below. Show the maximum-throughput pipeline using a minimal number of registers.



3 x 4 = 12

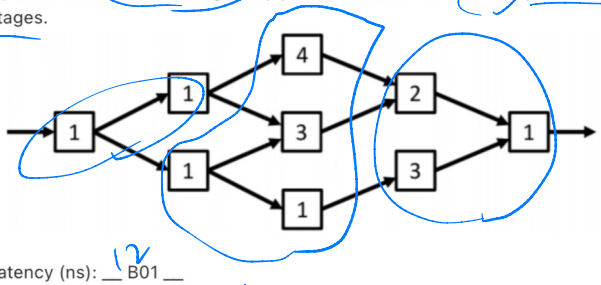
Latency (ns): __ A07 __ 12
Throughput (ns^{-1}): __ A08 __ 1/3

- A07 = ?
- A08 = ?

Question B

For each of the questions below, create a valid N-stage pipeline of the given circuit. Each component in the circuit is annotated with its propagation delay in nanoseconds. Give the latency and throughput of each design, assuming ideal registers (t_{PD} = 0, t_{SETUP} = 0). Remember that our convention is to place a pipeline register on each output.

(1) Show a maximum-throughput pipeline that uses the smallest possible number of pipeline stages.

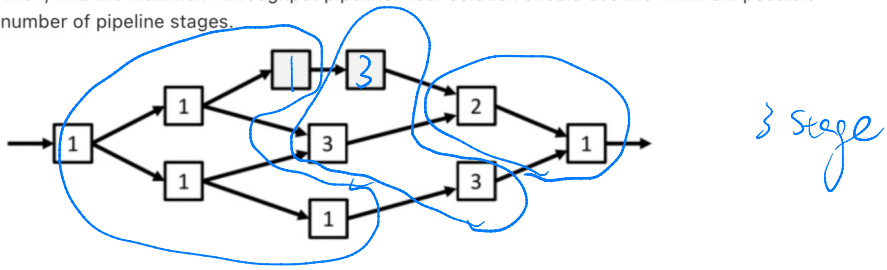


3 stage

Latency (ns): __ B01 __ 12
Throughput (ns^{-1}): __ B02 __ 1/4

- B01 = ?
- B02 = ?

(2) You reimplement the 4 ns combinational component in the previous circuit using two faster components connected in series, shown in **grey** below. You can choose the propagation delays of these two components, as long as their delays add to 4 ns (e.g., they could be 3 ns + 1 ns, 2 ns + 2 ns, etc.) Choose the propagation delays of both components in a way that lets you pipeline the circuit for maximum throughput while minimizing the number of pipeline stages. Then, find the maximum-throughput pipeline. Your solution should use the minimum possible number of pipeline stages.



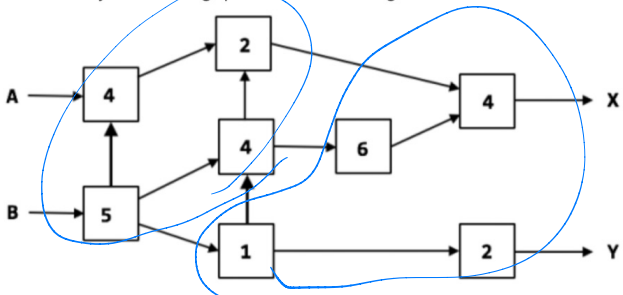
Latency (ns): __ B03 __
Throughput (ns^{-1}): __ B04 __

- B03 = ?
- B04 = ?

Question C

For each of the questions below, create a valid N-stage pipeline of the given circuit. Each component in the circuit is annotated with its propagation delay in nanoseconds. Give the latency and throughput of each design, assuming ideal registers (t_{PD} = 0, t_{SETUP} = 0). Remember that our convention is to place a pipeline register on each output.

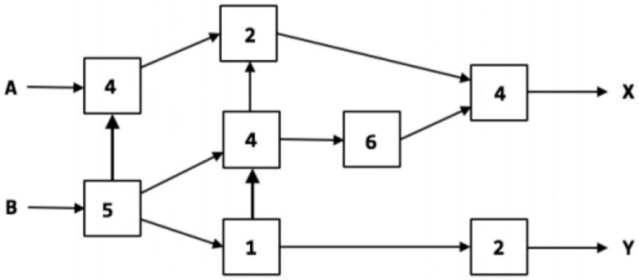
(1) Show the maximum-throughput 2-stage pipeline using a minimal number of registers. What are the latency and throughput of the resulting circuit?



Latency (ns): __ C01 __
Throughput (ns^{-1}): __ C02 __

- C01 = ?
- C02 = ?

(2) Show the maximum-throughput pipeline using a minimal number of registers. What are the latency and throughput of the resulting circuit?

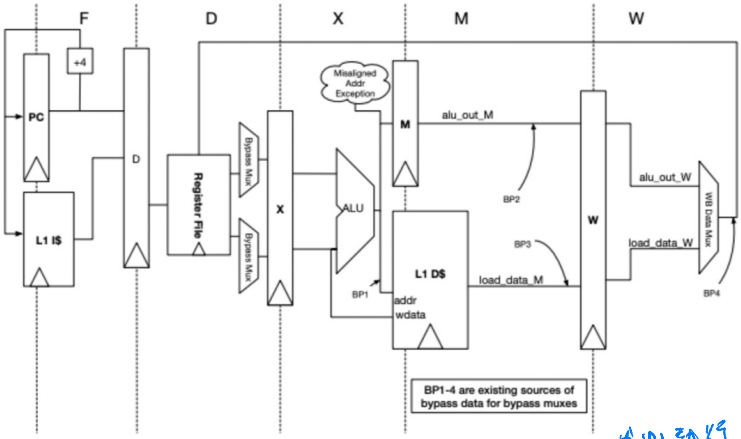


Latency (ns): __ C03 __ 24
Throughput (ns^{-1}): __ C04 __ 1/6

- C03 = ?
- C04 = ?

Question D

Consider the below 5-Stage Pipeline processor.



(1) Even a simple in-order pipelined processor would make use of speculative execution. For the 5-stage pipeline above, assume that there is no virtual memory, and that misaligned accesses are checked in the Execute stage. For the instruction sequence below, complete the execution diagram and specify the cycles in which the second add is being executed speculatively. For example, if one stage F is being executed speculatively, use the notation `F (S)`.

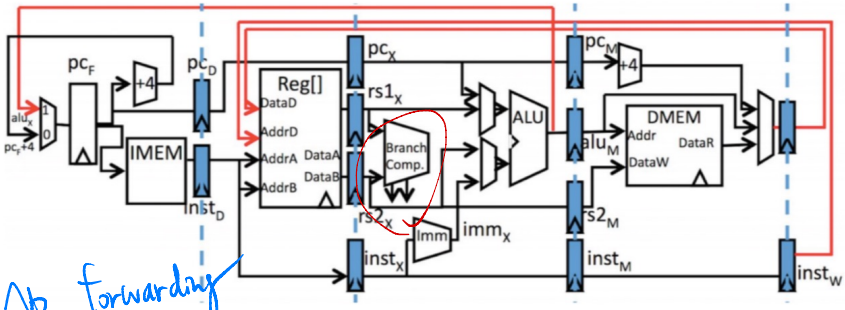
Clock Cycle	0	1	2	3	4	5	6	7	8	9
`add x1, x2, x0`	F	D	X	M	W	-	-	-	-	-
`lw x3, 0(x2)`	-	D01	D02	D03	D04	D05	-	-	-	-
`add x3, x4, x5`	-	-	D06	D07	D08	D09	D10	-	-	-

- D01 = ?
- D02 = ?
- D03 = ?
- D04 = ?
- D05 = ?
- D06 = ?
- D07 = ?
- D08 = ?
- D09 = ?
- D10 = ?

(2) Given the 5-stage pipeline above, how long is the load-use delay? Answer in terms of how many bubbles must be added between a load and a dependent register-register instruction that is fetched right after the load.
How many bubble(s) must be added? __ D11

Question E

Consider the 5-Stage pipeline RISC-V processor.



(1) How many cycles does it take to run each iteration of the following loop on a standard 5-stage pipelined RISC-V processor **P1**?

```
loop: lw   x10, 0x100(x0)
      beqz x10, loop
      add  x12, x10, x11
      sub  x13, x12, x1
```

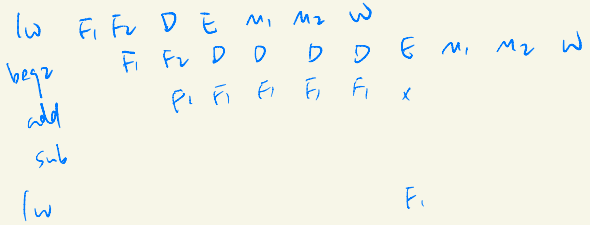
Number of cycles per loop iteration: __ E01

(2) Consider a modified processor, **P2**, which has extra hardware for the special case of checking if a register is equal to zero or not in the decode stage. What would be the number of cycles per loop iteration in this case?
Number of cycles per loop iteration on processor P2: __ E02 __

- E02 = ?

(3) Consider a third processor, **P3**, whose instruction and data memories are pipelined and take 2 clock cycles to respond. Assume that P3 also has the extra hardware for checking if a register is equal to zero or not in the decode stage. What would be the number of cycles per loop iteration using P3?
Number of cycles per loop iteration on processor P3: __ E03 __

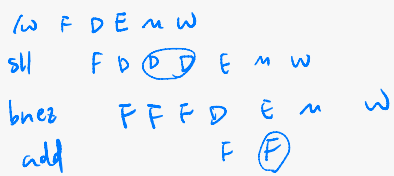
- E03 = ?



Question F

You have been given a 5-stage pipelined RISC-V processor. Unfortunately, the processor you hve been given is defective: it has no bypass paths, annulment of instructions in branch delay slots, or pipeline stalls.

```
      nop
      nop
      nop
      nop
Loop: lw   x10, 0x0(x10)
AAA:
      sll  x14, x10, x11
BBB:
      bnez x10, loop
CCC:
      add  x13, x10, x13
      nop
      nop
      nop
      nop
```



You undertake to convert some existing code, designed to run on an unpipelined RISC-V, to run on your defective pipelined processor. The code snip on above is a sample of the program to be converted. It does not make much sense to you, but you are to add the minimum number of NOP instructions at the various tagged points in this code to make it give the same results on your defective pipelined RISC-V as it gives on a normal, unpipelined RISC-V.

Note that the code snip begins and ends with sequences of NOPs; thus, you do not need to worry about pipeline hazards involving interactions with instructions outside of the region shown.

(1) Specify the minimal number of NOP instructions (defined as `add x0, x0, x0`) to be added at each of the labeled points in the above program.

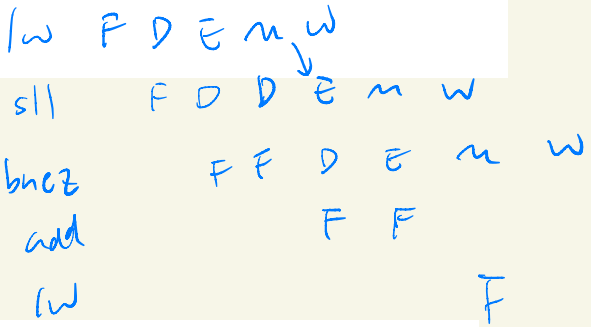
NOPs at Loop: __ F01 __
NOPs at AAA: __ F02 __
NOPs at BBB: __ F03 __
NOPs at CCC: __ F04 __

- F01 = ?
- F02 = ?
- F03 = ?
- F04 = ?

(2) On a fully functional 5-stage pipeline (with working bypass, annul, and stall logic), the above code will run fine with no added NOPs. How many clock cycles of execution time are required by the fully functional 5-stage pipelined RISC-V for each iteration through the loop?

Clocks per loop iteration: __ F05 __

- F05 = ?

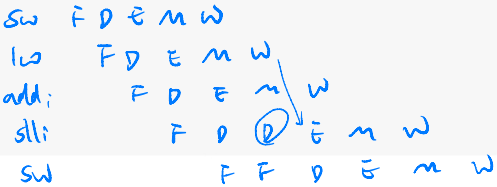


Question G

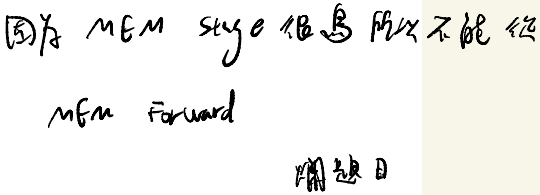
The following programs are being executed on the 5-stage pipelined RISC-V processor with full bypassing. For each code listing, the pipeline diagram shows the state of the pipeline for cycle `1000` of execution. Please fill in the diagram for cycle `1001`; use `UNKNOWN` if you cannot tell what opcode to write into a stage.

(1) Program below

```
    ...
sw   x1, 0(x0)
lw   x17, 0xC(x1)
addi x2, x2, -4
slli x11, x17, 2
sw   x11, 0(x2)
jal  ra, factorial
    ...
```



Cycle	1000	1001
IF	sw	G01
ID	slli	G02
IE	addi	G03
MEM	lw	G04
WB	sw	G05



(2) Program below

```
    ...
xor  x11, x11, x12
slli x12, x12, 3
sub  x13, x12, x11
and  x12, x13, x11
add  x13, x12, x13
sw   x13, 0x100(x0)
    ...
```

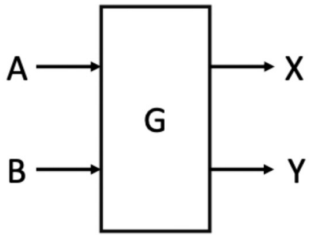
Cycle	1000	1001
IF	`add`	G06
ID	`and`	G07
IE	`sub`	G08
MEM	`slli`	G09
WB	`xor`	G10

- G06 = ?
- G07 = ?
- G08 = ?
- G09 = ?
- G10 = ?

2021

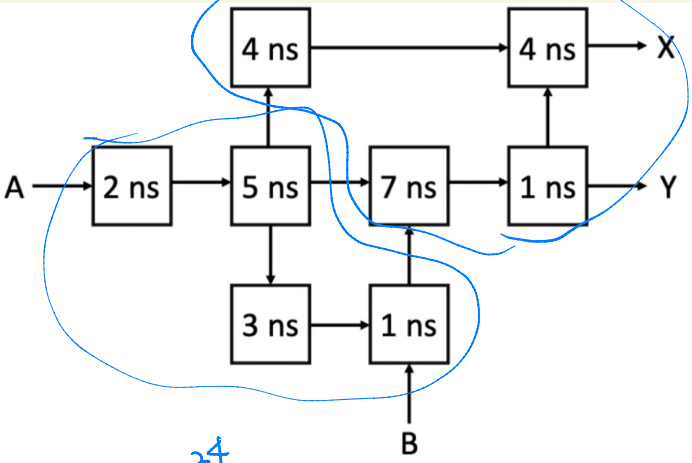
Problem A

Consider the G circuit, depicted on the following.



Since the G circuit is currently only combinational, we know exactly what to improve: Pipeline it.

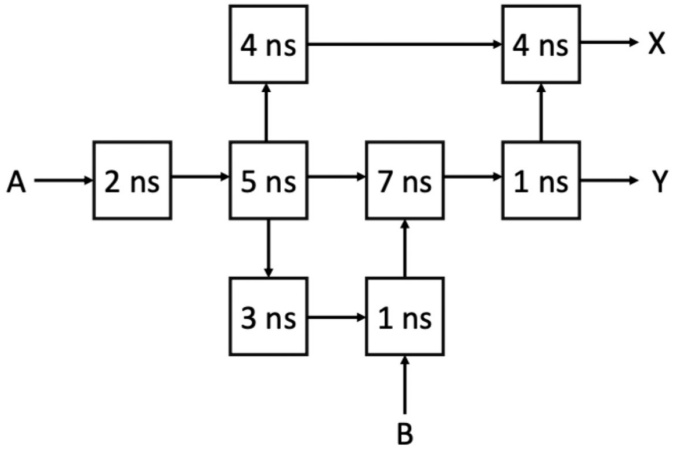
1. We decide to start small with a **two-stage pipeline**. Using the diagram below, please show a two-stage pipeline with maximal throughput using a minimum number of registers. Calculate the overall throughput and latency of your circuit.



- Latency (ns): __ A01 __ 24
- Throughput ($\frac{1}{ns}$): __ A02 __ $\frac{1}{10}$

- A01 = ?
- A02 = ?

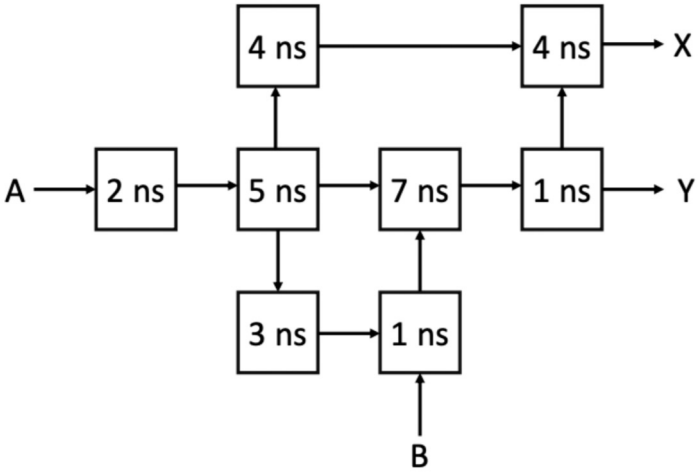
2. Now, we decide to add an additional stage. Using the diagram below, please show a **three-stage pipeline** with maximal throughput using a minimum number of registers. Calculate the overall throughput and latency of your circuit.



- Latency (ns): __ A03 __
- Throughput ($\frac{1}{ns}$): __ A04 __

> - A03 = ?
> - A04 = ?

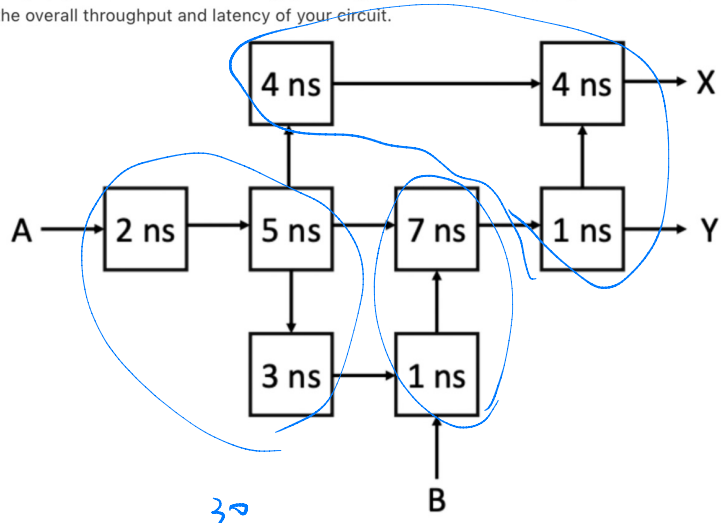
3. Then, think of the maximalthroughput pipeline. Using the diagram below, please show a pipeline that maximizes the throughput using a minimum number of registers. Calculate the latency and throughput of our circuit. You should use the smallest number of pipeline stages required to achieve maximum throughput.



- Latency (ns): __ A05 __
- Throughput ($\frac{1}{ns}$): __ A06 __

- A05 = ?
- A06 = ?

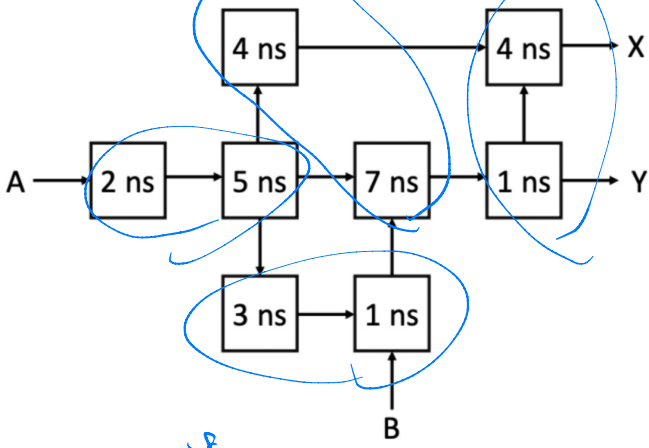
2. Now, we decide to add an additional stage. Using the diagram below, please show a **three-stage pipeline** with maximal throughput using a minimum number of registers. Calculate the overall throughput and latency of your circuit.



- Latency (ns): __ A03 __ 30
- Throughput ($\frac{1}{ns}$): __ A04 __ $\frac{1}{10}$

- A03 = ?
- A04 = ?

3. Then, think of the maximalthroughput pipeline. Using the diagram below, please show a pipeline that maximizes the throughput using a minimum number of registers. Calculate the latency and throughput of our circuit. You should use the smallest number of pipeline stages required to achieve maximum throughput.



- Latency (ns): __ A05 __ 28
- Throughput ($\frac{1}{ns}$): __ A06 __ $\frac{1}{7}$

- A05 = ?
- A06 = ?

Problem B

Assume that we are trying to build a RISC-V processor. For the reference purpose, there are two 5-stage (`IF` , `DEC` , `EXE` , `MEM` , `WB`) bypassing processors. However, they are a little broken. The processors are as follows:

- **Processor 1**: A defective 5 stage bypassing processor which does not annul instructions following branches.
- **Processor 2**: A defective 5 stage bypassing processor that reads all values bypassed back as 0 but reads correctly from the register file.

Known properties:

- Both processors always predict that branches are not taken (they always fetch from `PC + 4`).
- Both processors determine the direction of the branch in the `EXE` stage.

We try this simple RISC-V looping code on both of these processors. Assume at the start of the code that `x2` is set to some number greater than `0` and that all the other registers are set to `0` .

```
L1: addi x1, x2, -4
    bnez x1, L1
    slli x3, x1, 1
    lw x2, 0x100(x0)
. = 0x100
    .word 0x4
```

1. If we had a fully working 5 stage bypassing processor, which always fetches from `PC + 4` and annuls instructions following taken branches, how many cycles would it take for one iteration of the loop? Assume that the `bnez` is taken. Provide the number of cycles per loop iteration.
   - You may use the pipeline diagrams below to help you answer the question, but you are not required to fill them out.

cycle	i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8
IF	addi								
DEC									
EXE									
MEM									
WB									

   - number of cycles per loop iteration = __ B01 __

- B01 = ?

2. If we pass the number `0x10` into `x2`, by the time the second iteration's `addi` reaches `EXE`, what value will be passed to `x1` for each of the two defective processors? If the loop does not make it to a second iteration's `addi`, write `N/A`.
   - You may use the pipeline diagrams below to help you answer the question, but you are not required to fill them out.

cycle	i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8
IF	addi								
DEC									
EXE									
MEM									
WB									

cycle	i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8
IF	addi								
DEC									
EXE									
MEM									
WB									

   - Processor 1's `x1` : __ B02 0
   - Processor 2's `x1` : __ B03 N/A

Processor 1 : 不會經過 instruction even if branch

4 - 4 = 0

processor 2 : bypassed value 都沒有 0

add: F D E M W

next F D E M W

F D E M W

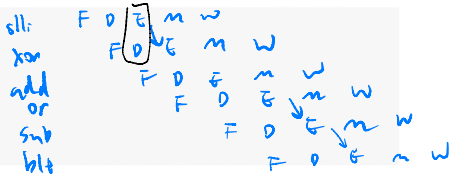
F D E M W

NT branch (sll: (W

N/A

Since having full bypassing can be very costly, we attempt to reorder the instructions in this loop so that we can minimize the total number of cycles per loop iteration while using a single bypass path.

```
loop:
    slli a4, a3,5
    or a5, a0, a7
    xor a3, a4, a2
    add a1, a2, a3
    sub a6, a5, a2
    blt a6, a2, loop
```



We can change the order of the instructions in the program as long as it does not change the final result. In what order would you execute the 6 instructions in the loop so that you only need a single bypass path?

1. Here, we select `EXE` → `DEC` bypass path. Please list the opcodes in order of execution: __ C01 __ (split in comma)

   - C01 = ?

2. Now, suppose we are offered another processor. This processor is identical to the processor used above with full bypassing, except that it can make branch decisions in the decode stage rather than the execute stage. The disadvantage is that this increases the clock period from `400` ps (before) to `450` ps (with the branch calculated in decode).
   - Which processor is faster (old: branch decision in `EXE` and t_{CLK} = 400ps, new: branch decision in `DEC` and t_{CLK} = 450ps)? __ C02 __

   - C02 = ?

Problem D

Consider the following C code:

```
int price[6] = {7, 5, 8, 10, 15, 7};
int maximum = 10, c = 3, t = 5;
for (int i = 0; i < 6; i++) {
   if (price[i] > maximum)
       maximum = price[i];
}
int total_cost = maximum + c + t;
```

Then, the translated RISC-V assembly:

```
// x4 = 0x24 - length of price in bytes
// x5 = 0x3 - c
// x6 = 0x5 - t
// x7 = 0xA - maximum
// x1 = 0x400 - address of price[0]
start: addi x2, x0, 0
       slli x2, x2, 2
loop:  add x8, x2, x1
       lw x3, 0(x8)
       bge x7, x3, skip
       mv x7, x3
       ori x7, x7, 0
skip:  addi x2, x2, 4
       blt x2, x4, loop
       add x7, x7, x5
       add x7, x7, x6
```

Assume the registers are initialized to the values specified in the assembly code comments.

In the following five-stage pipelined RISC-V processor (`IF` , `DEC` , `EXE` , `MEM` , `WB`):

- All branches are predicted not-taken. (Always fetch from `PC + 4`).
- Branch decisions are made in the `EXE` stage.
- The pipeline has full bypassing.
- The processor annuls instructions following taken branches.
- Assume that in the first iteration of the loop both branches are taken.

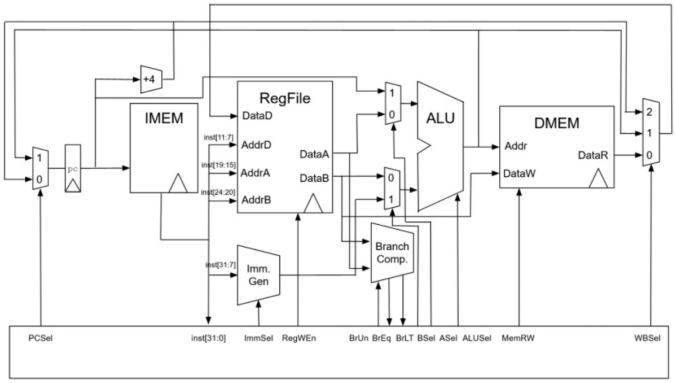
1. How many cycles did it take to execute the first loop iteration on this processor? Make sure not to include the first two instructions at label start in your cycle count.
   - Cycles to execute first iteration of the loop on this processor: __ D01 __
     - D01 = ?

2. If you could modify your fetch stage to always fetch the correct next instruction instead of predicting all branches not taken, how many cycles will it now take to execute the first iteration of the loop on this modified processor? __ D02 __
   - D02 = ?

3. Now, let's change this processor **without** bypassing. That is, all data hazards are resolved by stalling. How many cycles did it take to execute the first loop on this processor? __ D03 __
   - D03 = ?

Problem E

Given the standard RISC-V datapath, determine if the following is implementable or not without any additional functional units? Assume the instruction is not a pseudoinstruction encoding.



1. Consider the new instruction `isnull rd, rs1` which check if an input given through `rs1` is considered `NULL` or not by C standard. The result is returned through `rd` as a bit. Is it Implementable? __ E01 __ (Answer with `Yes` or `No`)
   - E01 = ? Yes

2. What changes would you need to make in order for the instruction to be able to execute correctly? Assume all modifications and additions are done on top of the existing single cycle datapath. Select all that apply. __ E02 __ (Anwser in the following items and use comma to split) c e h
   - `a`. Modify Branch Comparator logic.
   - `b`. Modify the control signals to the `ALU`.
   - `c`. Modify the control logic for the Branch Comparator.
   - `d`. Modify `ALU` buses.
   - `e`. Modify the control logic for `WBSel`.
   - `f`. Add additional control signals for the writeback mux.
   - `g`. Modify control logic for `ALU` / `ALUSel`.
   - `h`. Modify the control logic for parsing `instr[31:0]`.
   - `i`. Add an additional comparator.
   - `j`. None of the above.
   - E02 = ?

3. `isnull rd, rs1` is not in a standard RISC-V instruction format; as we are attempting to reduce the number of hardware changes in our datapath. We instead choose to implement our instruction as a pseudoinstruction in the following format. Which of the following statements is true? Assume earlier changes propagate. Select all that apply __ E03 __ (Anwser in the items and use comma to split)

    - `a` . We need to provide a second argument `x0` when calling the instruction and modify the control signals.
    - `b` . We need to provide a second argument `x0` as a comparator for all branch comparisons.
    - `c` . It is impossible to represent as an R-Type instruction
    - `d` . We need to wire `x0` as a comparator for all branch comparisons.
    - `e` . We need to wire `x0` as `rs2` and modify the control signals.
        - E03 = ?

e

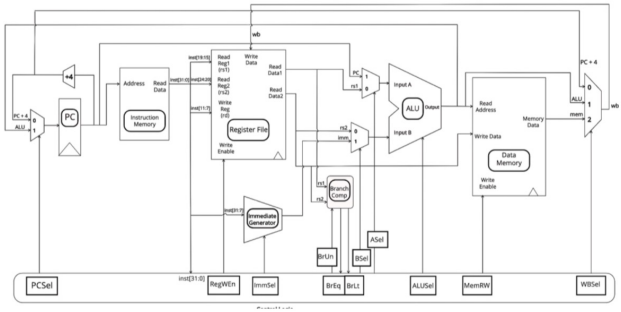
4. We plan to add a new R-Type signed compare instruction called `comp`, into the RISC-V single-cycle datapath, to compare `R[rs1]` and `R[rs2]` and set `R[rd]` appropriately. The RTL for it is shown below.

```
// comp rd, rs1, rs2
if R[rs1] > R[rs2]: R[rd] = 1
elif R[rs1] == R[rs2]: R[rd] = 0
else: do nothing
```

We want to change the datapath to make this work. We start by adding two more inputs (`0x00000000` and `0x00000001`) to the rightmost `WBSel` MUX. What else is required to make this instruction work? Select all that apply __ E04 __ (Anwser in the items and use comma to split)

- `a` . Modify Branch Comp
- `b` . Modify Imm. Gen.
- `c` . Modify the `ALU` and `ALUSel` control signals
- `d` . Modify the control logic for `RegWEn`
- `e` . Modify the control logic for `MemWEn`
    - E04 = ?

and `0` otherwise. Using the datapath below, fill in the following table with the rest of the control bits for this instruction. If the control bit can be set to `*`, please draw an `X` in the table below.



IArrN	PCSel	RegWEn	MemRW	WBSel	BrUn	ALUSel
1	F01	F02	F03	F04	F05	ADD

- F01 = ?
- F02 = ?
- F03 = ?
- F04 = ?
- F05 = ?

2022

Problem A

Assume that we are building a JIT compiler, which generates RISC-V instructions at runtime. Because a JIT must render and execute a native binary image at runtime, true machine-code JITs necessitate platforms that allow for data to be executed at runtime – code is data. Here is a simplified implementation based on C and RV32I: (filename: `jit.c`)

```
1  #include <stdint.h>
2  #include <stdio.h>
3
4  typedef int func_t(void);
5  int main(int argc, char *argv[])
6  {
7      uint32_t instructions[] = {
8          [0] = A01,   /* addi a0, zero, 56 */
9          [1] = 0x00008067,   /* A02 */
10     };
11
12     /* Reinterpret the array address as a function */
13     func_t *jit = (func_t *) instructions;
14     printf("%d\n", jit());
15
16     return 0;
17 }
```

Compile via the command below:

```
$ riscv-none-elf-gcc -O0 -march=rv32i -mabi=ilp32 -o jit.elf jit.c
```

Run it with rv32emu.

```
$ rv32emu jit.elf
```

We get the corresponding output.

```
56
inferior exit code 0
```

Reference: Porting a JIT compiler to RISC-V: Challenges and Opportunities

1. Given instruction `addi a0, zero, ?` as shown in Line 8, what is the HEX value of A01?
   - A01 = ?
2. What is the dissassmbly of `0x00008067` (shown in Line 9) ? __ A02 __
   - A02 = ?
3. Explain why the above program works. __ A03 __
   - A03 = ?

Reinterpret the array address as function

Problem B

The Python code for the recursive function `count8`, which counts the number of 8's (base 16) in the input, is shown below. `count8(0x988) = 2` and `count8(0x81) = 1`, for instance. A RISC-V assembly-based implementation of the function is shown below.

- [ ] Python code

```
def count8(in):
    if in < 8:
        return 0
    end = in & 0xF
    current_8 = 0
    if end == 8:
        current_8 = 1
    return current_8 + count8(in >> 4)
```

- [ ] The corresponding RISC-V assembly

```
count8:
    li a1, 8
    bgt a1, a0, base
    addi sp, sp, -12
    sw ra, 0(sp)
    sw s1, 4(sp)
    andi a2, a0, 0xF # a2 is end ; a0 is in
    li s1, 0
L1:
    bne __ B01 __      # if end == 8
    addi s1, s1, 1
continue:
    srli a0, a0, 4
    sw a0, 8(sp)
    call count8
    add a0, a0, s1
L2:
    lw ra, 0(sp)
    lw s1, 4(sp)
    addi sp, sp, 12
    j done
base:
    li a0, 0
done:
    ret
```

0x9D8

0x9EC

1. What should be `B01` to make the assembly implementation match the Python code?
   - B01 = ? bne a2, a1, continue
2. How many words will be written to the stack before the program makes each recursive call to the function `count8` ? __ B02 __ 3

The instruction call `count8` makes the first call to the function `count8` in the program outside of the function definition. The program is interrupted right before the execution of `lw ra, 0(sp)` at label `L2` during a recursive call to `count8`. The layout of a memory area is shown in the diagram below. The values of all addresses and data are shown in hex. The sp register's current value, `0xEAC`, points to the location indicated in the diagram.

-	Address	Data
	0xE94	0x9E4
	0xE98	0x0
	0xE9C	0x0
	0xEA0	__ B03 __
	0xEA4	__ B04 __
	0xEA8	__ B05 __
sp→	0xEAC	0x9E4
	0xEB0	0x1
	0xEB4	0x82
	0xEB8	0xC8
	0xEBC	0x73
	0xEC0	0x828
	0xEC4	0xC14
	0xEC8	0x82

3. Fill B03, B04, and B05 for the stack associated with `0xEA0`, `0xEA4`, and `0xEA8` (answer in HEX)
    - B03 = ?
    - B04 = ?
    - B05 = ?

4. What is the initial input in at the initial call to count8? __ B06 __ (answer in HEX)
    - B06 = ?

5. What are the values in the following registers right when the execution of `count8` is interrupted? (answer in HEX)
    - ra = B07
    - a2 = B08
    - s1 = B09
    - B07 = ?
    - B08 = ?
    - B09 = ?

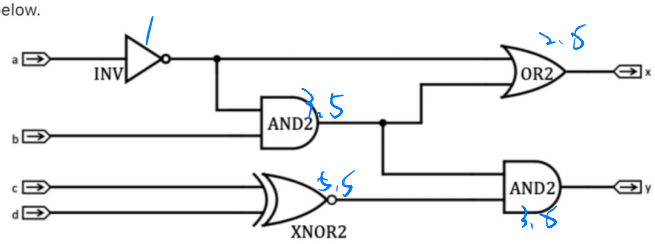
6. What is the value in register `a0` right when the execution of `count8` is interrupted? (answer in HEX) __ B10 __
    - B10 = ?

7. What is the hex address of the `call f` instruction that made the initial call to `count8`? __ B11 __
    - B11 = ?

8. What is the hex address of the `continue` label? __ B12 __
    - B12 = ?

Problem C

Take a look at the logic diagram below, which computes two outputs {x, y} from four inputs {a, b, c, d}. Calculate the circuit's t_{PD} using the t_{PD} details for the gate components listed in the table below.



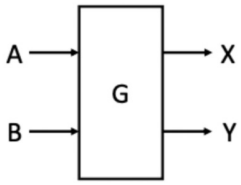
Gate	t_{PD}
XNOR2	5.5ns
AND2	3.5ns
OR2	2.5ns
INV	1.0ns

t_{PD} = __ C01 __ ns

- C01 = ?

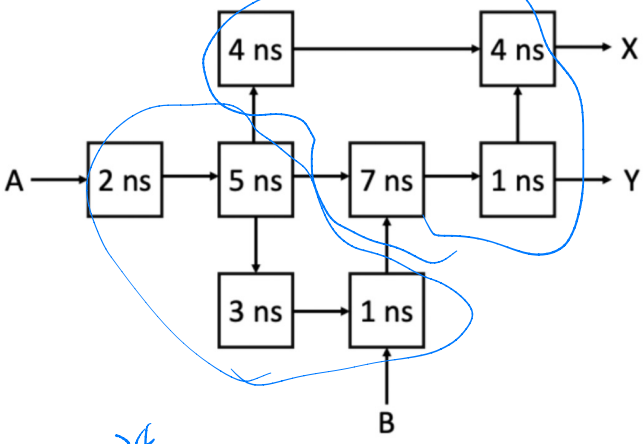
Problem D

Assume that you are creating `G` circuit, as seen below. Everything has been running smoothly until you discover that the G circuit's throughput is too low one day. Your biggest client needs a throughput at least twice as high as what you can currently offer in order to include the `G` circuit into their new product. You know precisely what to do: pipeline it as the `G` circuit is now only combinational.



D.

1. You choose to start small with a two-stage pipeline for your initial iteration. Calculate your circuit's overall throughput and latency.



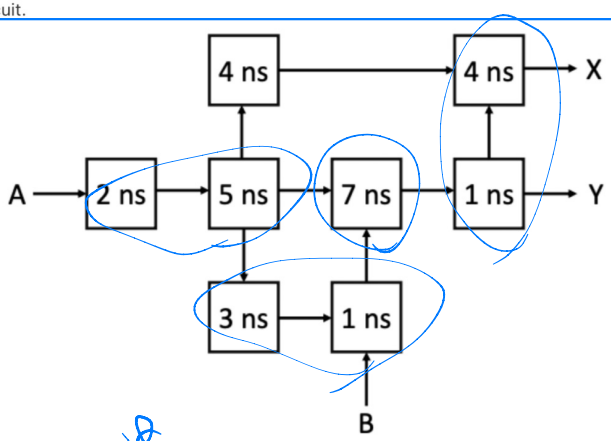
24

- Latency: __ D01 __ ns
- Throughput: __ D02 __ ns^{-1}
  - D01 = ?
  - D02 = ?

1/12

You decide to add an additional level after completing the first iteration because you are beginning to feel quite at ease with the `G` circuit. Let's implement the maximal throughput pipeline. Kindly explain a pipeline using the following design that maximizes throughput while utilizing the fewest possible registers. Calculate your circuit's throughput and latency.

NOTE: Use the fewest possible pipeline stages to maximize throughput in your pipelined circuit.

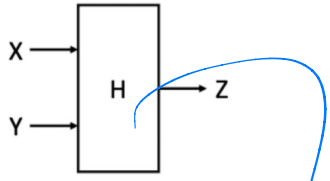


28

- Latency: __ D03 __ ns
- Throughput: __ D04 __ ns^{-1}

1/7

3. Assume that we want to integrate another pipelined circuit, Circuit H, shown below. However, this circuit has multiple different timing specifications! In the below table, you have timing specifications for two different implementations of H: Version A, and Version B. Using your maximal-throughput G circuit from part C, select the version of H that minimizes your latency of the combined G-H circuit. Additionally, calculate the latency and throughput of the combined G-H circuit.



-	Version A	Version B
clock period	7 ns	8 ns
stages	2	1

- Which one can minimize your latency of the combined G-H circuit? (Answer in `Version A` or `Version B`) __ D05 __
- Overall latency = __ D06 __ ns
- Overall throughput = __ D07 __ ns^{-1}
  - D05 = ?
  - D06 = ?
  - D07 = ?

Problem E

We are considering the following RISC-V processor design. For this processor `P`, the code and the starting state of the necessary registers in the register file are shown below. Keep in mind that while `x6` is in hexadecimal, registers `x1` through `x5` are given their values in decimal.

```
start:
    lw x1, 0(x6)
    addi x2, x0, 5
    blt x1, x2, end
    addi x3, x2, 11
    sub x4, x3, x1
    xori x5, x6, 0x1
end:
    sub x4, x3, x2
    addi x3, x4, 7
    addi x3, x1, 3
. = 0x400
.word 0x1
```

Register	Value
x1	~~5~~ 1
x2	~~11~~ 5
x3	~~30~~ 32
x4	~~19~~ 25
x5	20
x6	0x400

Processor `P` : 5-stage pipelined RISC-V processor, which is fully bypassed and has branch annulment. Branch decisions are made in the `EXE` stage and branches are always predicted not taken. What are the values of registers `x3` and `x4` at the start of cycle 12 (in decimal):

- x3: __ E01 __
- x4: __ E02 __
  - E01 = ?
  - E02 = ?

Problem F

The P_1 and the P_2 are two RISC-V based designs whose performance you are expected to evaluate. You have made the choice to compare the performance of the two CPUs using the next construction step.

```
L1:
    slli x11, x10, 2
    lw x12, 0x80(x11)
    add x13, x13, x12
    addi x10, x10, 1
    blt x10, x14, L1
    sub x14, x14, x15
    xori x15, x14, 0x1
```

The P_1 is a 4-stage pipelined processor with full bypassing. It attempts to improve performance by decreasing the number of cycles it takes to execute each instruction. The `EXE` and `MEM` stages are combined. Load requests are sent in the `EXE` stage and received in the `WB` stage one cycle after. The `IF` stage speculatively fetches the instruction at PC+4 and annuls incorrectly fetched instructions after a branch misprediction. Branches are resolved in the `EXE` stage.

The P_2 is a 6-stage pipelined processor with full bypassing. It attempts to improve performance by raising the clock speed. The `EXE` stage is split into two stages to break up the critical path through the ALU. The `IF` stage always fetches the instruction at PC+4 and annuls incorrectly fetched instructions after a taken branch. ALU and Branch ALU results are available only in `EXE2` .

1. How many cycles does it take the P_1 to execute one iteration of the loop? __ F01 __
   - F01 = ?

2. How many cycles does it take the P_2 to execute one iteration of the loop? __ F02 __
   - F02 = ?

3. Assume that we can build the P_1 with a clock period of 6ns. What is the clock period that the P_2 must achieve to attain the same performance on this assembly sequence? __ F03 __ ns
   - F03 = ?

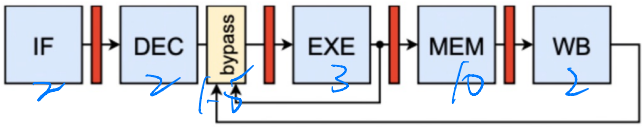
Problem G

Assume that while our data memory utilizes a little more realistic model that includes a DRAM, our instruction memory continues to respond in one cycle. A memory address, a read/write signal, and optional write data are all inputs to the memory. The CPU first uses the cache to fulfill read or write requests. The data can be quickly sent back to the pipeline if the access succeeds in the cache. It will take significantly longer for the CPU to reach the DRAM, and then retrieve the needed data from it.

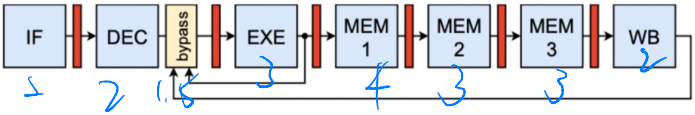
Assume that each of the 2 processors has extra hardware that enables them to foresee when branches will be taken and to get the branch's destination instruction immediately following the branch instruction (without requiring any stalls).

Consider the following 2 RISC-V processor designs.

- [ ] P_1: Basic 5-stage pipeline with bypass paths from EXE to DEC and from WB to DEC



- [ ] P_2: Since the memory access can take a long time in the worst case, we use pipelining to improve the processor performance. The `MEM` stage is further divided into three pipeline stages.



1. Assume the 2 processors have ideal registers (t_{SETUP} = t_{HOLD} = t_{PD} = t_{CD} = 0ns) between different pipeline stages. Please derive the minimum clock cycle given the following propagation delays for the logic blocks.

Logic Block	Propagation Delay
IF	2 ns
DEC	2 ns
EXE	3 ns
MEM	10 ns
WB	2 ns
bypass	1.5 ns
MEM1	4 ns
MEM2	3 ns
MEM3	3 ns

   - Clock period for P_1: __ G01 __ ns
   - Clock period for P_2: __ G02 __ ns
     - G01 = ?
     - G02 = ?

2. The code segment below copies elements from array A to array B, where Array A starts at address 0x100, and Array B starts at address 0x500. Assume that this loop has been running for a long time, and that the processor always predicts that the branch will be taken.

```
    add x1, x0, x0
    add x2, x0, x0
Next:
    lw x3, 0x100(x1)
    addi x1, x1, 4
    sw x3, 0x500(x2)
    addi x2, x2, 4
    blt x1, x4, Next // x4=1000
```

How many cycles does this loop take to execute in P_1? __ G03 __

- G03 = ?

3. Assume the memory can only serve one read or write operation at a time. In P_2, since `MEM1`, `MEM2` and `MEM3` stages all use the memory, we now have structural hazard. Structural hazard happens if multiple instructions try to use the same hardware resource. To resolve the structural hazard, we can only allow one memory access instruction exist in the `MEM1`, `MEM2`, and MEM3 stages. Note that it is ok to have multiple other types of instructions, e.g., arithmetic instructions, as long as they do not need to use the memory. Specifically, we need some mechanism to stall memory access instructions.
The detailed stall mechanism is explained below. We will inject NOPs to the `EXE` stage when the `IF` and `DEC` stages are stalled. When the `DEC` has a memory access instruction, we stall for the following 2 cases:

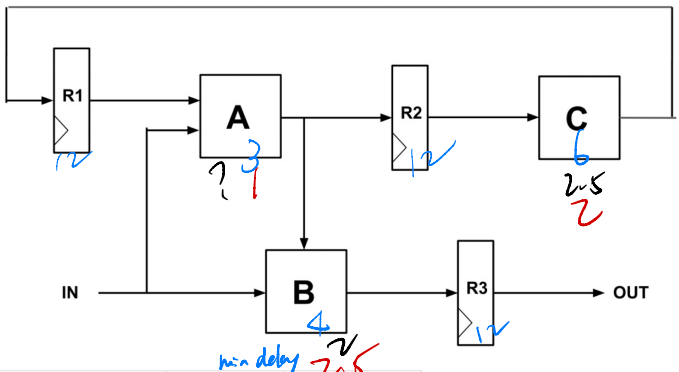
   - If the `EXE` stage also has a memory access instruction, stall `IF` and `DEC` for 2 cycles
   - If the `MEM1` stage also has a memory access instruction, stall `IF` and `DEC` for 1 cycle

These stalls are in addition to any stalls required to deal with data hazards. How many cycles does this loop take to execute in P_2? __ G04

- G04 = ?

Quiz4 of Computer Architecture (2023 Fall)

HackMD



2023

-	t_{PD}	t_{CD}	t_{SETUP}	t_{HOLD}
Register (R1, R2, R3)	5ns	1ns	7ns	2ns
Circuit A	3ns	?	-	-
Circuit B	4ns	2ns	-	-
Circuit C	6ns	2.5ns	-	-

1. What is t_{PD} of this sequential circuit? __ B01 __ ns
   - B01 = ?

上午9:15 11月14日 週二
91%
hackmd.io
Quiz4 of Computer Architecture (2023 Fall)
HackMD
Quiz4 of Compute...
Problem A
Problem B
Problem C
Problem D
Problem E
Problem F
Problem G
全部展開
回到頂部
移至底部

2. What is t_{CD} of this sequential circuit? __ B02 __ ns
   - B02 = ?
3. What is the minimum value of t_{CD} for Circuit A that will ensure proper operation of this circuit? __ B03 __ ns
   - B03 = ?
4. What is the minimum clock period that can be employed to clock all the registers within the circuit? __ B04 __ ns
   - B04 = ?
5. Available from the supplier are alternative circuits for A, B, and C, each with the following specifications.

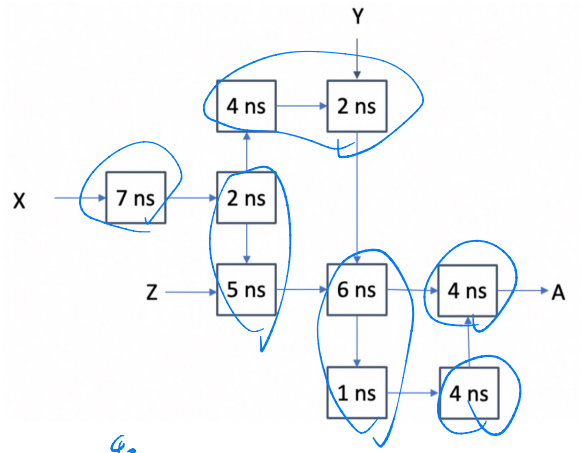
-	t_{PD}	t_{CD}	t_{SETUP}	t_{HOLD}
A-New	1ns	0.5ns	-	-
B-New	2.5ns	2ns	-	-
C-New	2ns	1ns	-	-

Replacement of these new circuits is cost-sensitive, allowing for the substitution of only one combinational circuit to achieve a minimized clock period. Please specify which combinational circuit you intend to replace and the resulting minimum clock period.

- Combinational circuit replaced: __ B05 __
- t_{CLK} of circuit: __ B06 __ ns

each output.

1. Consider a 2-stage pipeline optimized for maximum throughput while utilizing the fewest possible registers. Determine the latency and throughput of the resulting circuit, taking careful note of the direction of each arrow.



- Latency: __ C01 __ ns
- Throughput: __ C02 __ ns^{-1}
  - C01 = ?
  - C02 = ?

- Latency: __ C01 __ ns
- Throughput: __ C02 __ ns^{-1}
  - C01 = ?
  - C02 = ?

2. Consider a maximum-throughput 4-stage pipeline using a minimal number of registers. What are the latency and throughput of the resulting circuit?

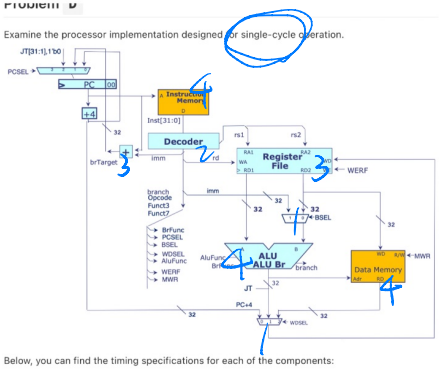
- Latency: __ C03 __ ns
- Throughput: __ C04 __ ns^{-1}
  - C03 = ?
  - C04 = ?

3. Consider a maximum-throughput pipeline that uses a minimum number of pipeline stages. What are the latency and throughput of the resulting circuit?

- Latency: __ C05 __ ns
- Throughput: __ C06 __ ns^{-1}
  - C05 = ?
  - C06 = ?

Problem D

Examine the processor implementation designed for single-cycle operation.



Below, you can find the timing specifications for each of the components:

Component	Propagation delay (t_{PD})
Register	1ns
Decoder	2ns
RegFile read	3ns
MUX	1ns
ALU	4ns
Adder	3ns
Memory read (instruction or data)	4ns

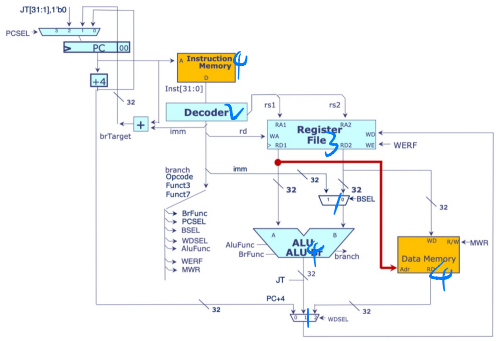
Setup/hold times for clocked inputs (registers and writes to RegFile and data memory)

- Setup time (t_{SETUP}): 2 ns
- Hold time (t_{HOLD}): 0 ns

1. What is the shortest clock cycle duration achievable for this processor? Minimum t_{CLK}: __ D01 __ ns

上午9:43 11月14日 週二
83%
hackmd.io
Quiz4 of Computer Architecture (2023 Fall)
HackMD
Quiz4 of Compute...
Problem A
Problem B
Problem C
Problem D
Problem E
Problem F
Problem G
全部展開
回到頂部
移至底部

2. The current processor's performance is suboptimal. We intend to introduce an alternative data path (while keeping the control logic unchanged) in which the data memory's Adr input is sourced differently. What is the minimum clock cycle time of the new processor? Minimum t_{CLK}: __ D02 __ ns



- D02 = ?

3. Now, the new processor executes certain instructions incorrectly as per the RISC-V ISA

Problem E

Take a look at the loop provided below, which calculates the sum of values in a linked list. We execute this code on a typical 5-stage RISC-V processor equipped with complete bypassing. Assume that branches are consistently predicted as **not taken**, and that branch determinations are made during the EX stage. Also, consider that the loop undergoes numerous iterations and is presently in the middle of its execution.

```
loop:
    lw a1, 0(a0)
    lw a0, 4(a0)
    add a2, a2, a1
    bnez a0, loop
    mv a0, a2
    addi sp, sp, 8
    ret
```

1. Assume that at cycle 100 the `lw a1, 0(a0)` instruction is fetched. Fill in the following:
   - Number of cycles per loop iteration: __ E01 __
   - Number of cycles per loop iteration wasted due to stalls: __ E02 __
   - Number of cycles per loop iteration wasted due to annulments: __ E03 __
     - E01 = ?
     - E02 = ?
     - E03 = ?

We aim to enhance the performance of this processor. Upon careful examination, we have identified an opportunity to save cycles by introducing additional bypass paths. Contrary to its label, "full bypassing" does not fully exploit the potential for bypassing. It primarily bypasses