CS 61C Spring 2021 Sample Exam.

MIDTERM

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `cs61c@berkeley.edu`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- ◯ You must choose either this option
- ◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- ☐ You could select this choice.
- ☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

Please complete and submit these questions before the exam starts.

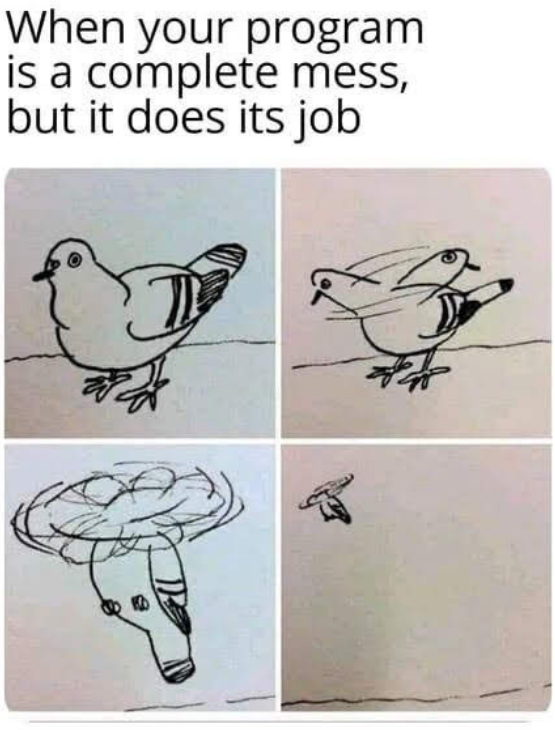
(a) What is your full name?

(b) What is your student ID number?

(c) If an answer requires hex input, make sure you only use capitalized letters! For example, 0xDEADBEEF instead of 0xdeadbeef. You will be graded incorrectly otherwise! Please always add the hex (0x) and binary (0b) prefix to your answers or you will receive 0 points. For all other bases, do not add the suffix or prefixes.

Do not add units unless the problem explicitly tells you to!

Some of the questions may use images to describe a problem. If the image is too small, you can click and drag the image to a new tab to see the full image. You can also right click the image and download it or copy its address to view it better. You can use the image below to try this. You can also click the star by the question if you would like to go back to it (it will show up on the side bar). In addition, you are able see a check mark for questions you have fully entered in the sidebar. Questions will auto submit about 5 seconds after you click off of them, though we still recommend you click the save button.



Good luck!

1. Number Rep

(a) **(1.0 pt)** Which of the following representation systems have two representations for the decimal number zero?

- ☐ Bias notation
- ☐ Signed-magnitude
- ☐ Floating Point
- ☐ Two's complement
- ☐ Unsigned

(b) **(1.0 pt)** What is the range of numbers that can be represented in biased notation where N is the number of bits in the binary representation and B is the bias (B is a negative number)?

- ◯ [-2(n-1) + B, 2(n-1) - 1 + B]
- ◯ [B, 2(n) - 1 + B]
- ◯ [-2(n-1) - B, 2(n-1) - 1 - B]
- ◯ [-2(n-1) - 1 + B, 2(n-1) - 1 + B]
- ◯ [-B, 2(n) - 1 - B]
- ◯ [-2(n-1) - 1 - B, 2(n-1) - 1 - B]
- ◯ [-B, 2(n-1) - 1 - B]
- ◯ [B, 2(n-1) - 1 + B]

Translate the following decimal numbers to their correct representation in binary using **7 bits**. Answers that contain more or less than **7 bits** will receive no credit. If the number cannot be represented, enter N/A **(be sure to use all caps)**. **REMEMBER to include the binary prefix!**.

(c) **i.** **(1.0 pt)** 64 in 2's complement

ii. **(1.0 pt)** 64 in unsigned notation

iii. **(1.0 pt)** 64 in bias notation (with an added bias of -63)

iv. **(1.0 pt)** -25 in 2's complement

v. (1.0 pt) -25 in unsigned notation

vi. (1.0 pt) -25 in bias notation (with an added bias of -63)

2. (5.0 points) CALL

(a) (2.0 points) General

i. (1.0 pt) Which of the following statements must be true about compilers?

- ☐ Compiled code generally is only able to run on one ISA.
- ☐ Compilers produce larger code than interpreters but do it faster.
- ☐ The code produced is always more efficient and higher performance than that produced by interpreters.
- ☐ There is only one compiler per language.
- ☐ Compilers are always more difficult to write than interpreters.
- ☐ The easiest step of CALL is compilation; the harder parts are assembling, linking, and loading.

(b) (1.0 pt) Which of the following statements must be true about translators?

☐ They can produce interpreted code.

☐ They can produce compiled code.

(c) (3.0 points) CALL

For each of the following questions, select whether it's true or false. Assume all questions are in the context of 32-bit systems.

i. (0.5 pt) After compilation, the outputted assembly file will only ever have instructions reflected by the RISC-V 32-bit base instruction set and its corresponding pseudoinstructions.

◯ True

◯ False

ii. (0.5 pt) Object files are distinctly segmented into components and will tell the linker exactly how many bytes and where each component is located.

◯ True

◯ False

iii. (0.5 pt) The linker is in charge of stitching together files such that the resulting file includes three distinct segments.

◯ True

◯ False

iv. (0.5 pt) In the outputted assembly file, the instructions can be directly translated to machine code and be understood by the machine.

◯ True

◯ False

v. (0.5 pt) The symbol and relocation table can be synonymous in what they do during assembly.

◯ True

◯ False

vi. (0.5 pt) External libraries referenced in the original C code will be downloaded during compilation to allow for the assembler to properly reference the necessary labels.

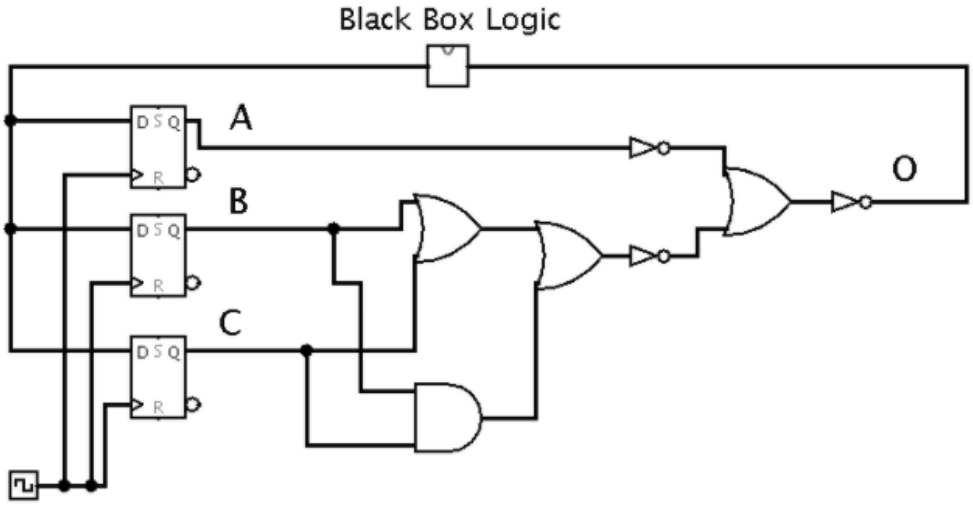
◯ True

◯ False

3. SDS

For the following question, do NOT include units in your answer!

In the following circuit, the registers have a clk-to-q delay of 6ns and setup times of 5ns. `NOT` gates have a delay of 3ns, `AND` and `OR` gates have a delay of 7ns, and the "Black Box" logic component has a delay of 9ns.



Circuit

(a) (2.5 pt) What is the maximum allowable hold time of the registers?

(b) (2.5 pt) What is the minimum acceptable clock period for this circuit?

4. RISC-V

(a) (18.0 pt) Note: This coding question will be autograded. If your code fails to compile, you will initially receive 0 points when scores are reported. You WILL be given an opportunity to request a regrade for compilation errors, at -1 point per line that needs to be changed. You are allowed to use Venus for this question, and thus are expected to test your code. We have provided the following helper function for use in testing your code. You should NOT use this function in your actual solution:

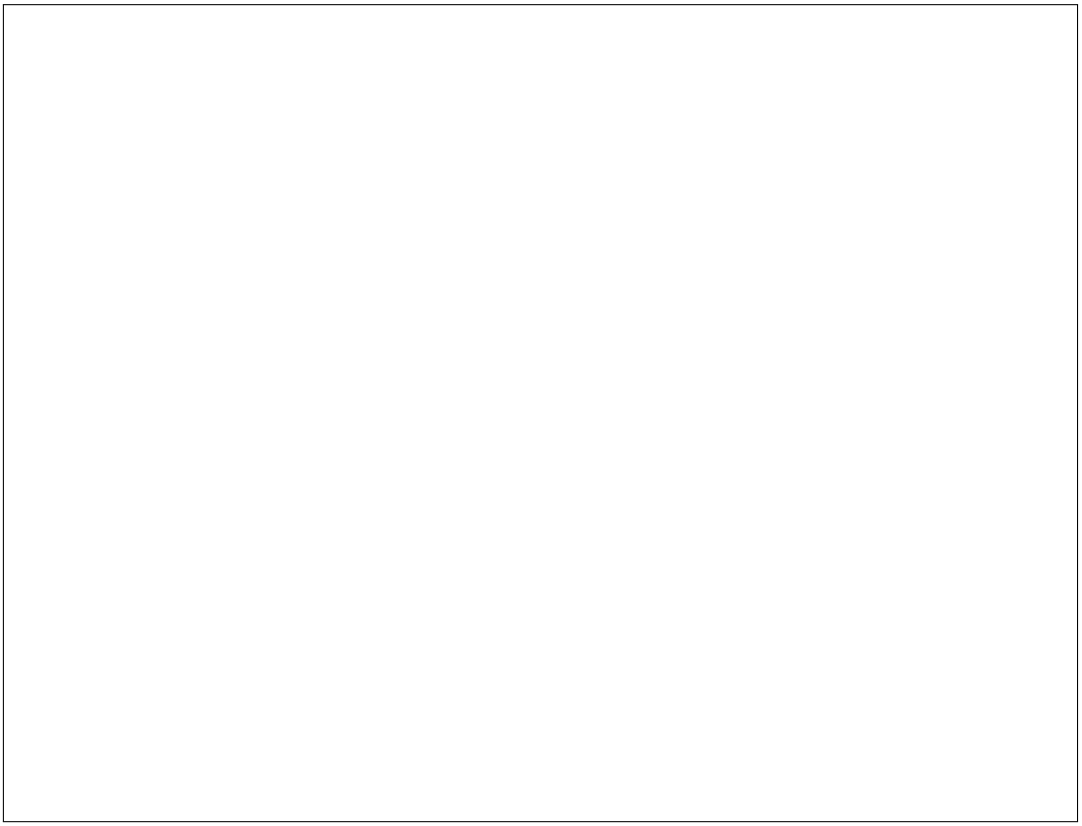
```
#Input: a1 contains a buffer to a string
#Output: Prints the string. Returns nothing.
print_str:
li a0 4
ecall
ret
```

The function `stringtriple` receives as input a null-terminated string, and writes a string where every letter is repeated three times. For example, if we called `stringtriple` on the input string "`Hello World!`", we would get the output string "`HHHeeellllllooo  WWWooorrrllldddd!!!`". More specifically: `stringtriple` has the following inputs: `a0` stores a null-terminated string. It is guaranteed that the string is well-formatted. Let the string have `strlen` of n. `a1` contains a pointer to a buffer of length at least $3n + 1$ chars, potentially containing garbage data. You may assume that the buffer does not overlap with the string stored in `a0`.

`stringtriple` outputs the following: `a0` should return the pointer to the buffer we initially received in `a1`. The buffer should contain the string originally provided in `a0`, with every character tripled. You must properly null-terminate the string, and for full credit must NOT replicate the null terminator at the end.

For full credit, you must follow proper calling convention. You do NOT need to include the `stringtriple:` label.

`stringtriple:`



5. DATAPATH PIPELINING

(a) (3.0 points) Control Signals

Given the standard single-cycle RISC-V datapath (Figure 1) for the following instruction, what control signals are provided for the datapath? Use 0, 1, or * for signals and N/A if the given signal is not provided by the control logic; for the ALU, assume 0 refers to the add operation and 1 to the sub operation. For the immediate selector, use I for immediate-type, S for store-type, B for branch-type, J for jump-type, and U for upper-immediate type. **Use the signals without any leading or trailing whitespace otherwise it may be marked incorrect. All answers should be exactly one signal.**

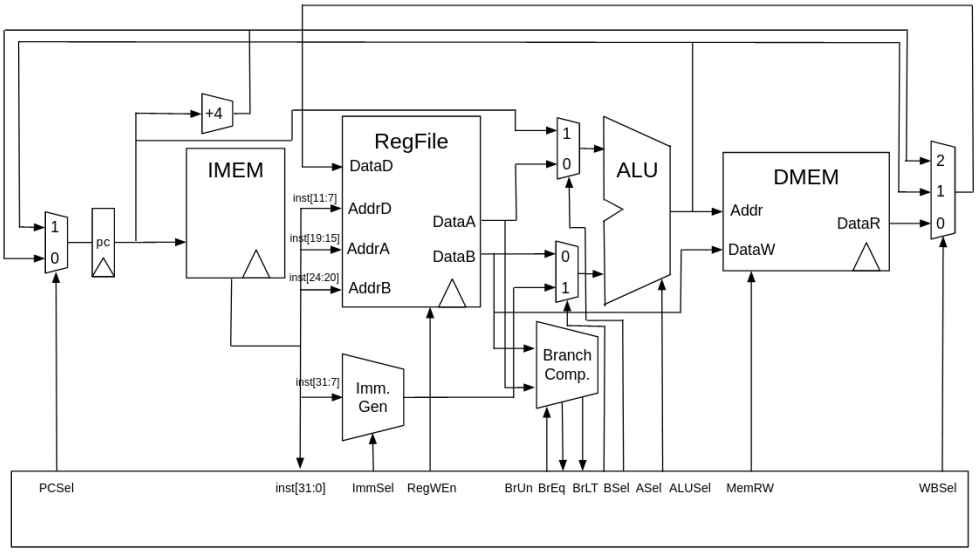


Figure 1

`blt` (not taken)

i. (0.25 pt) `PCSel`

ii. (0.25 pt) `inst[31:0]`

iii. (0.25 pt) `ImmSel`

iv. (0.25 pt) `RegWEn`

v. (0.25 pt) `BrUn`

vi. (0.25 pt) `BrEq`

vii. (0.25 pt) `BrLt`

viii. (0.25 pt) `BSel`

ix. (0.25 pt) `ASel`

x. (0.25 pt) `ALUSel`

xi. (0.25 pt) `MemRW`

xii. (0.25 pt) `WBSel`

(b) (4.0 points) Datapath

Say we alter our datapath such that we split data memory into two serial steps, i.e. instead of MEM we have MEM1 and MEM2 (see Figure 2) which are respectively the first and second half of the original memory. The way this new datapath operates is that any memory access first must go through MEM1; if it finds the data it needs there, then the datapath bypasses the second memory access and does not attempt to make an access; the value obtained from MEM1 is then passed to the writeback mux through the MEM slot. However, if the address is not in MEM1, then the datapath signals to control logic that the ALU needs to pass the address to MEM2 in order to make an access there, blocking any subsequent ALU operations for that instruction, and the result is fed into the MEM input in the writeback mux. Accessing MEM1 and MEM2 individually takes the same amount of delay.

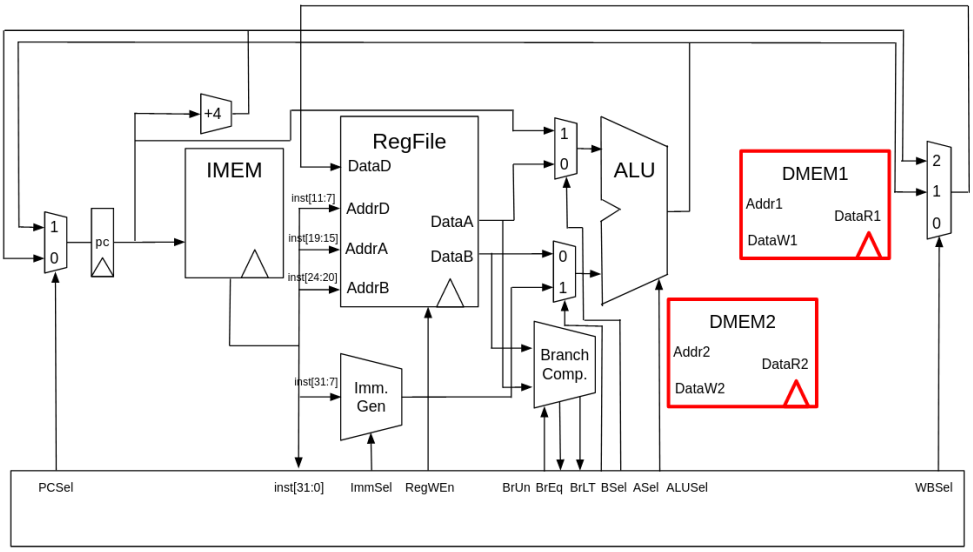


Figure 2

NOTE: certain portions of the datapath are disconnected; this is intentional and it is up to you to decide which portions to connect.

i. (4.0 pt) What is the minimum amount of additional control logic and hardware may be needed to implement this in the datapath for it to be functional? Select all that apply.

- ☐ Path from MEM1 output to MEM2 input.
- ☐ Path from ALU to MEM1 input.
- ☐ Add additional state element.
- ☐ Path from control logic to additional mux.
- ☐ Path from ALU to MEM2 input.
- ☐ Path from control logic to MEM1.
- ☐ Path from MEM2 out to WB mux.
- ☐ Path from control logic to MEM2.
- ☐ Path from MEM2 to control logic.
- ☐ Path from MEM1 to control logic.
- ☐ Add additional mux.
- ☐ Path from control logic to additional state element.
- ☐ Path from MEM1 to WB mux.

(c) (1.5 points) Pipelining

Assume we've added pipeline registers to create a 6-stage pipeline with our updated datapath, as seen below (Figure 3). This pipelined datapath acts similarly to our standard RISC-V 5-stage pipeline with the additional change of the memory section being separated into two sections. Assume no forwarding logic has been implemented, no branch prediction, and one register operation per cycle.

For the given code, what hazards might exist and due to which lines? Assume all registers have been initialised and that all labels are defined and that all branches are taken. **HINT: having a table open will be useful for scratch work.**

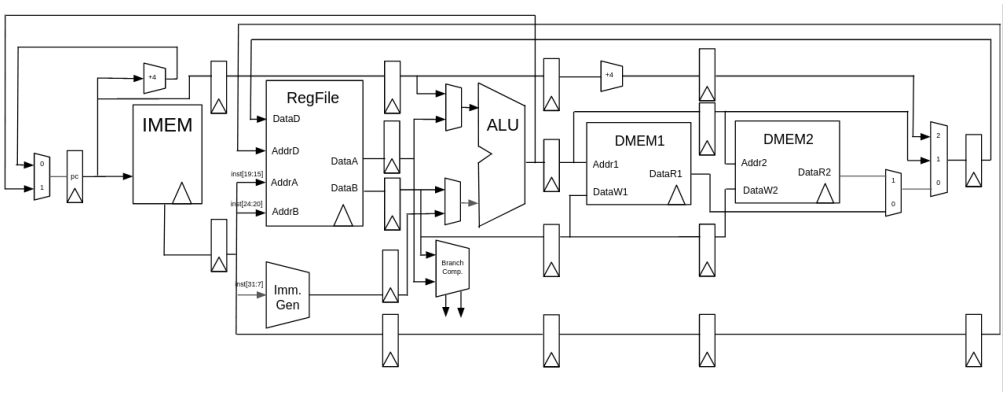


Figure 3

```
1 bne x0, t0, next
2 addi t1, t0, 1
3 lb s0, 0(t1)
4 shw s0, 4(t0)
```

i. (0.5 pt) Lines 1/2

- ◯ Structural
- ◯ None
- ◯ Control
- ◯ Data

ii. (0.5 pt) Lines 2/3

- ◯ Control
- ◯ None
- ◯ Structural
- ◯ Data

iii. (0.5 pt) Lines 3/4

- ◯ None
- ◯ Structural
- ◯ Data
- ◯ Control

iv. (2.5 pt) What is the minimum number of NOPs needed to ensure the above code will run as expected?

(d) (4.0 points) Forwarding

We now add some forwarding paths to our datapath from the previous questions as seen below (Figure 4). For our code above, how many NOPs/stalls will we need now for our code to operate correctly? What about after we add forwarding paths from the end of DMEM1 to the beginning of DMEM1 and from the end of DMEM2 to the beginning of DMEM2? Give your answers in order. Note that not all forwarding logic is depicted in the diagram given; only the important forwarding paths are given. Refer to the key in the diagram for forwarding path types.

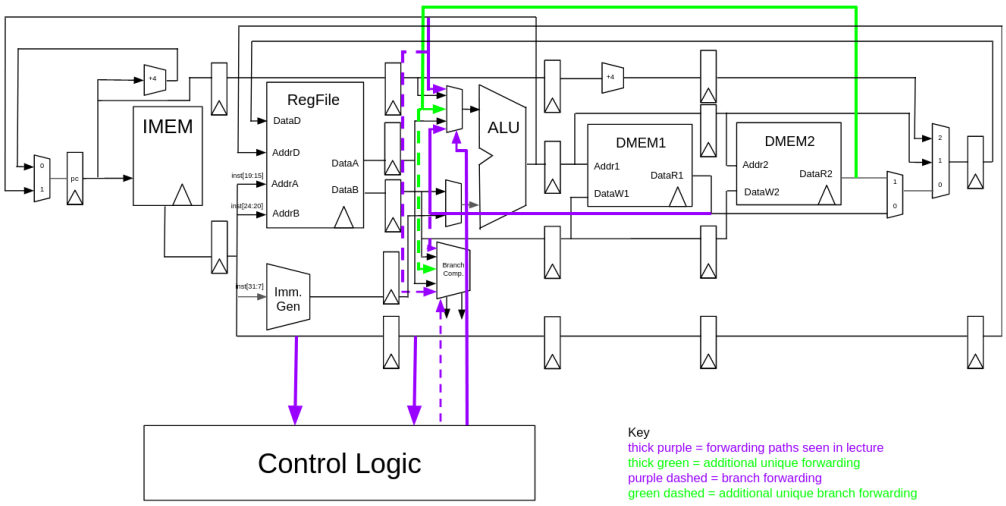


Figure 4

i. (1.5 pt) No memory forwarding paths NOPs/stalls count:

ii. (1.5 pt) With additional memory forwarding paths NOPs/stalls count:

iii. (1.0 pt) Looking at our datapath, what kind of hazards still remain given our pipeline? Select all that apply.

- ☐ Control
- ☐ Structural
- ☐ Data
- ☐ None

(e) (3.0 points) 5-Stage

We see in our previous pipelined datapath that there are several issues. We decide to alternatively switch to the following design where DMEM1 and DMEM2 fall in the same pipeline stage (Figure 5). Answer your following question with the 6-stage pipeline with **only** the marked forwarding paths in Figure 4 as the original design and the updated 5-stage pipeline as the alternative design.

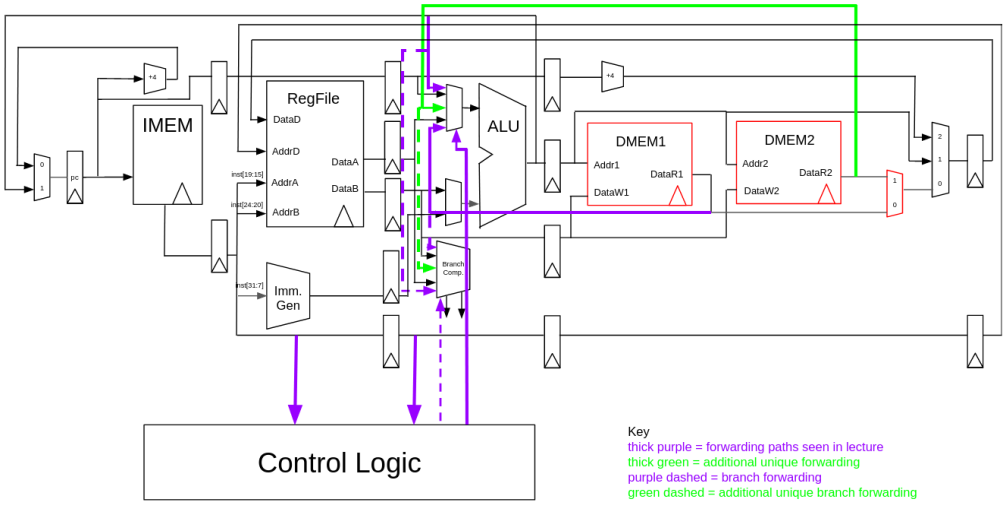


Figure 5

i. (3.0 pt) Our goal for this alternative design is to allow for concurrent memory accesses in consecutive memory operations upon failure in MEM1 to optimise the memory section time whilst maintaining correctness of data as it takes significantly longer than other pipeline sections. Assuming the MEM section in our alternative design allows for 2 instructions to occupy the stage simultaneously, with our given updated pipelined datapath, do we accomplish this?

- ◯ Yes; by allowing two instructions into a stage, we utilise both memory arrays without any structural hazards.
- ◯ Yes; by merging the two stages into one, we avoid structural hazards as only one output will ever be assumed at the end of the MEM stage.
- ◯ Yes; it lowers the overall time required to be used in MEM as two memory operations can be performed concurrently without issue.
- ◯ No; by allowing two instructions into a stage, there is the opportunity for register instability in the outputs.
- ◯ No; having two instructions in stage means that stage will always take two times as long to complete and negates any potential benefits from having two half-sized memory arrays.
- ◯ No; we reintroduce a structural hazard as this causes the ALU output register to need to store two unique values thus causing an incorrect output.
- ◯ No; we create additional control hazards as the datapath is dependent on non-existent control signals to determine whether instruction will access DMEM1 or DMEM2 first.
- ◯ None of the above.

6. FSM

The logic you used to read a dictionary (replacement set) in `Philphix` can be formalized as a finite state machine. The machine reads from its input one character at a time and keeps track of both its current state and a buffer (a string of characters). Instead of issuing an output, the machine takes an action when a transition occurs. Transitions can be taken when a specific character is seen (e.g., `'\n'`) or when a character belonging to a specific type is seen. The machine can use the following character types:

- Letters or Digits (`alphaNumeric`)
- Whitespace Characters (`whitespace`)
- Printable Characters (`printable`)

The possible actions for the machine are:

- Consume a character but ignore it (`consume`)
- Consume a character and append it to the buffer (`append`)
- Store the current contents of the buffer as a key (`storeKey`)
- Store the current contents of the buffer as a value, pair this value with the most recently stored key, and add this pair to the dictionary (`storeValue`)

For example, if the machine appends the most recently read character to its buffer if it is a whitespace character, then it should have the transition annotated as `whitespace/append`. If the machine consumes a newline character when it is seen, then it should have the transition `'\n'/consume`

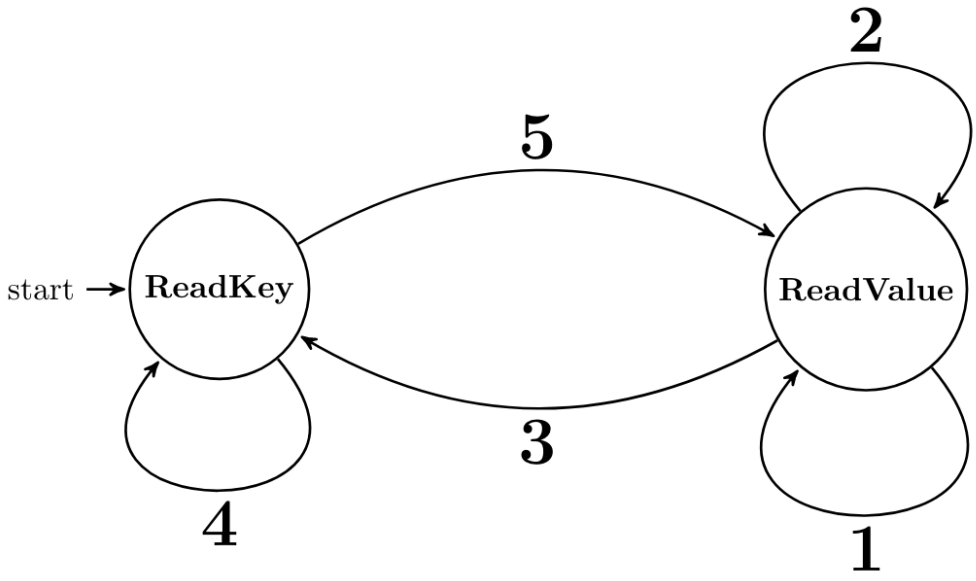
Two actions may be taken in sequence during a transition. For example: `whitespace/consume,storeKey` indicates that the state machine consumes and ignores the latest character and stores the contents of its buffer as a key.

Here are the instructions from the Project 1 spec for processing a dictionary:

> The replacement set consists of pairs of "words" (separated by an arbitrary number of tabs and spaces), each pair on its own line. The key is the first element of the pair, and the value is the second element. The key only consists of alphanumeric characters, but the value can include any non-whitespace printable character.

You may additionally assume that the last replacement pair in the dictionary input ends in a newline character (`\n`).

A state machine for reading a `Philphix` dictionary is shown below. Your task is to fill in the logic for each transition (Labeled **1** through **5**).



State Machine

(a) (1.0 pt) Specify the input and action for **Transition 1**:

- ◯ `whitespace/append`
- ◯ `whitespace/append,storeValue`
- ◯ `whitespace/consume`
- ◯ `whitespace/consume,storeValue`

(b) (1.0 pt) Specify the input and action for **Transition 2**:

- ◯ `alphaNum/consume`
- ◯ `printable/append`
- ◯ `alphaNum/consume,storeValue`
- ◯ `printable/append,storeValue`

(c) (1.0 pt) Specify the input and action for **Transition 3**:

- ◯ `'\n'/append`
- ◯ `'\n'/consume,storeValue`
- ◯ `'\n'/consume`
- ◯ `'\n'/append,storeValue`

(d) (1.0 pt) Specify the input and action for **Transition 4**:

- ◯ `printable/append`
- ◯ `alphaNum/append`
- ◯ `alphaNum/consume`
- ◯ `whitespace/append`

(e) (1.0 pt) Specify the input and action for **Transition 5**:

- ◯ `whitespace/consume,storeKey`
- ◯ `printable/consume,storeKey`
- ◯ `whitespace/storeKey`
- ◯ `printable/append`

7. C Programming

(a) Consider the following structure definition. Assume we are using a 32-bit machine.

```
struct foo {
  char a;
  char *b;
}
```

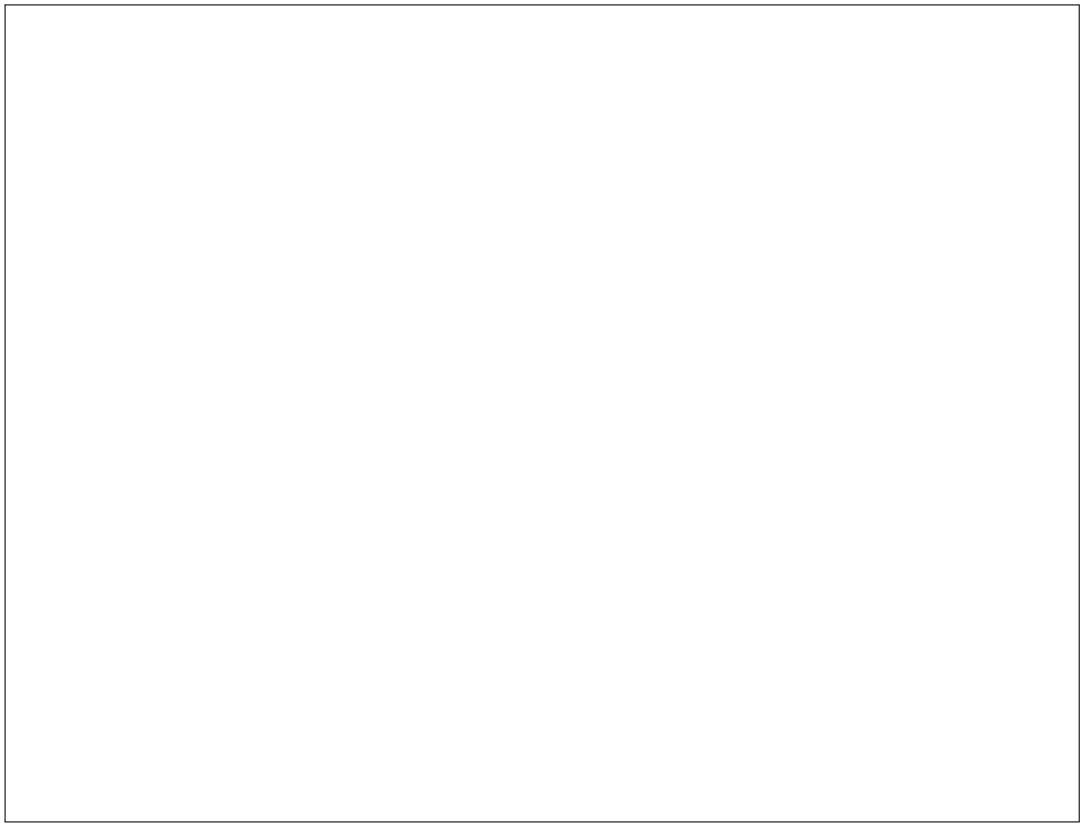
And the following C code

```
void bar(struct foo *f){
  int i;
  ....
  for(i = 0; i < 5, ++i){
    baz(f[i].b);
  }
  ....
}
```

i. (2.0 pt) What is `sizeof(struct foo)`?

ii. (2.0 pt) What is `sizeof(f)`?

iii. (4.0 pt) Translate the line baz(f[i].b) into RISC-V assembly. Assume that f is in S5 and i is in S6. You should use only 4 instructions and you can only use a0 as a temporary. **You may NOT use mul, div, or rem!**



(b) A very cool data-structure is called a "Bloom Filter" which implements a probabilistic set (a test to see if a given element is present or not, but there is a chance of an error where it says something is in the set when it isn't).

To set an element in a Bloom filter we repeatedly hash the element. This hash function takes both the input and an index number and returns a 64 bit value that both "looks random" (take 161 for details on how we can do that). Simply adding 1 to the index number means the hash for the next index is apparently random and unrelated from the previous hash.

So for a fixed number of iterations we hash the element and for each one set the corresponding bit in a giant bitfield to 1 to add an element to the set and check that all such bits are set to see if an element is in the set.

```
struct BloomFilter {
   uint64_t (*hashFunc) (void *, uint32_t);
   uint16_t iters;
   uint32_t size; /* The size in BITS of the bloom filter */
   uint8_t * data; /* Size of uint8_t is always 1 */
};
```

Fill in the following code:

```
/* Be sure to initialize the data */
struct BloomFilter *bfalloc(uint64_t (*hashFunc) (void *, uint32_t), int size, int iters) {
   struct BloomFilter *ret = malloc(sizeof(struct BloomFilter));
   ret->hashFunc = hashFunc;
   ret->size = size;
   ret->iters = iters;
   ret->data = _________;  /* Fill in A */
}

void bfset(struct BloomFilter *bf; void *element){
  uint32_t i = 0;
  uint32_t n;
  for(i = 0; i < b->iters; ++i){
      /* n = hash function called on the element modulo the # of BITS in the bloom filter */
      n = _____________________; /* Fill in B */
      /* Now we set the n'th BIT in the bloom filter to 1 */
      bf->data[n/8] = (______) | bf->data[n/8]; /* Fill in C */
  }

}
```

i. (3.0 pt) What should be line A? You don't need a sizeof() because unit8_t is always defined as one byte.

ii. (3.0 pt) What should be line B?

iii. (3.0 pt) What should be line C?

8. FLOATING POINT

For the following floating point questions, please use ^ as the power operator in your answer. Do NOT put parentheses around the exponent. For example, 2^-12.

(a) (3.0 pt) We define floating-point standard A to have 1 sign bit, 10 exponent bits, and 21 mantissa bits and floating-point standard B to have 1 sign bit, 18 exponent bits, and 45 mantissa bits. All other rules of IEEE 754 apply to standard A and B. How many more non-zero positive values can standard B represent compared to standard A? Please format your answer as additions and subtractions of 2's powers.

(b) For the following parts, use a floating point standard with 1 sign bit, 9 exponent bits, and 22 mantissa bits.

i. In discussion 3, we defined the step size of x to be the distance between x and the smallest value larger than x that can be completely represented. Now consider all floating-point numbers in the range $[2^{-120} + 2^{-110}, 80]$.

A. (2.0 pt) What is the largest step size?

ii. Conversions: convert the following floating-point representation into their decimal values or vice versa. If the conversion is impossible, write N/A. Please specify infinities as +inf or -inf, not a number as NaN, hex numbers as 0xddddddddd where each d is in [0, f]. If your answer is a decimal number, DO NOT round it.

A. (2.0 pt) $-2^{-100} \cdot 0.625$

B. (2.0 pt) 0xC07C0000

No more questions.