Final

Due Dec 18 at 11:59pm **Points** 100 **Questions** 17
Available after Dec 18 at 12am **Time Limit** 125 Minutes

Instructions

Regulations: https://www.seas.upenn.edu/~ese532/fall2020/final_details.pdf

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	125 minutes	13 out of 100 *

* Some questions not yet graded

ⓘ Correct answers are hidden.

Score for this quiz: **13** out of 100 *
Submitted Dec 18 at 11:53pm
This attempt took 125 minutes.

Question 1 — 1 / 1 pts

I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity and the exam regulations https://www.seas.upenn.edu/~ese532/fall2020/final_details.pdf (https://www.seas.upenn.edu/~ese532/fall2020/midterm_details.pdf) in completing this exam.

- (●) True
- ( ) False

Consider the following application in answering the questions on this exam:

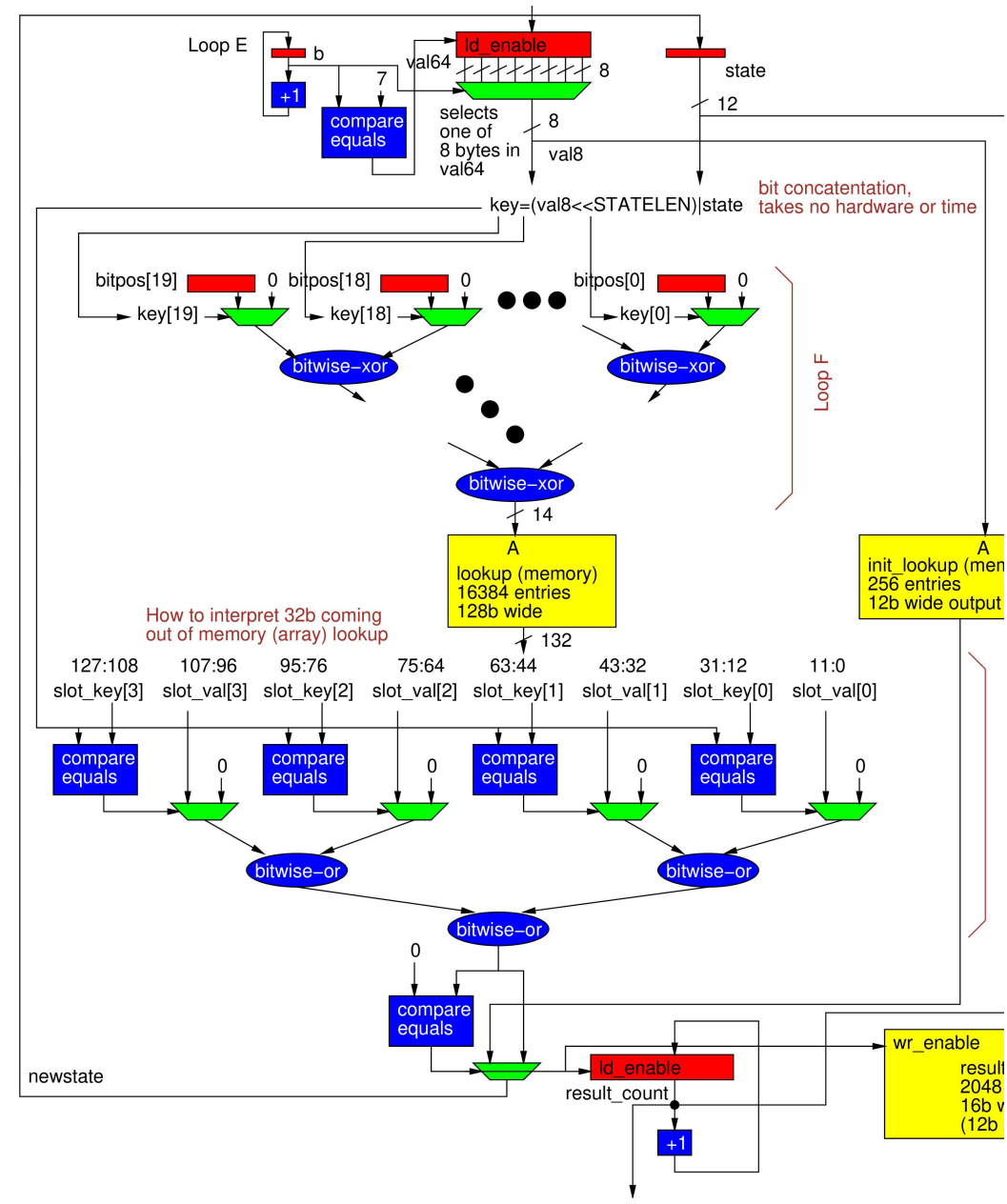
```
#define NFRAMES 256
#define MAX_RESULTS 8*NFRAMES
#define frame_type ap_int<240>
#define lookup_result_type ap_int<128>
#define STATELEN 12
#define KEYLEN (STATELEN+8)
#define KEY_MASK 0x0FFFFF
#define VAL_MASK 0x0FFF
#define NUM_SLOTS 16384
#define BUCKET_MASK 0x0FFFFFFFF
#define slot_type uint32_t
#define BUCKET_CAPACITY 4
#include<stdint.h>
extern lookup_result_type lookup[NUM_SLOTS];
extern uint16_t init_lookup[256];
void extract_compress(frame_type frames[NFRAMES],
                      uint8_t bitlocs[64],
                      uint16_t bitpos[KEYLEN],
                      uint16_t results[MAX_RESULTS],
                      int *num_results)
{
  uint64_t tmp[NFRAMES];

  for (int i=0;i<NFRAMES;i++) { // Loop A
    uint64_t result=0;
    int finalpos=1;
    frame_type val=frames[i];
    for (int j=0;j<64;j++) { // Loop B
      uint8_t bitloc=bitlocs[j];
      for (int k=128;k>0;k=k/2) { // Loop C
          if ((bitloc&0x01)==1)
              val=val/k;
          bitloc=bitloc/2;
        }
```

```
      if ((val&0x01)==1)
          result|=finalpos;
      finalpos=finalpos*2;
    }
   tmp[i]=result;
  }
  int result_count=0;
  int state=0;
  for (int i=0;i<NFRAMES;i++) { // Loop D
    uint64_t val64=tmp[i]; // val64 is input to pipeline pix
    for (int b=0;b<8;b++)  // Loop E (also shown in pipeline pix)
     {
       <see pipeline in next box; complete code for question 4>
     }
  }
  *num_results=result_count;
  return;
}
```

Here is a pipeline that implements loop E and the code (and loops) inside it.

Memories **bitpos**, **lookup**, **init_lookup**, and **results** are arrays defined in the code above. Registers **val64**, **b**, **state**, and **result_count** are variables defined in the code above.



Incorrect

Question 2

0 / 5 pts

Assuming:

- up to 6-input xors or ors can be packed into a single 1ns operation
- a comparison followed by a mux can be performed in a single 1ns operation

    - mux alone can also be performed in a 0.2ns
- each memory operation can be performed in a single 2ns operation
- an add or increment can occur in a single 1ns operation
- bitpos registers are preloaded

What is the cycle-bound II for loop E in ns?

16.4

Question 3

Not yet graded / 5 pts

Explain your II answer to previous question.

Your Answer:
Inputs to the loop are `val64` and `b`. At every iteration of the loop, we se e 16.4 ns of work, decomposed into:

- 0.2ns for mux using b which outputs val8
- 1ns for the ld_enable memory operation (since is register memory, as op posed to RAM)
- 0ms for assigning key
- 0ms for assigning to bitpos register, since pre-loaded
- 0.2ns for all the mux operations using key bits
- For bitwise-xors:
    - log_2(10) bitwise xors = 3.3 ~= 4 => 10 + 5 + 3 + 2 + 1 = 21
    - 10 => 2 * 1ns = 2ns, all others take only 1ns
    - Total latency = 6 ns
- 2ns for both init_lookup and lookup
- 1ns for compare equals followed by mux
- 2ns for both layers of bitwise-or operations
- 1ns for compare equals followed by mux
- 2ns for results memory access + ld_enable register store
- 1ns for increment

Total latency = 16.4ns

Question 4

Not yet graded / 10 pts

Provide code for the body of Loop E based on the pipeline show.

Recreate loops as flagged in the diagram.

Your Answer:

```
#define NFRAMES 256
#define MAX_RESULTS 8*NFRAMES
#define frame_type ap_int<240>
#define lookup_result_type ap_int<128>
#define STATELEN 12
#define KEYLEN (STATELEN+8)
#define KEY_MASK 0x0FFFFF
#define VAL_MASK 0x0FFF
#define NUM_SLOTS 16384
#define BUCKET_MASK 0x0FFFFFFFF
#define slot_type uint32_t
#define BUCKET_CAPACITY 4
#include<stdint.h>
extern lookup_result_type lookup[NUM_SLOTS];
extern uint16_t init_lookup[256];

/**
 * Memories bitpos, lookup, init_lookup, and results are arrays defined in t
he code above.
 *
 * Registers val64, b, state, and result_count are variables defined in the c
ode above.
 */

void extract_compress(frame_type frames[NFRAMES],
                      uint8_t bitlocs[64],
                      uint16_t bitpos[KEYLEN],
                      uint16_t results[MAX_RESULTS],
                      int *num_results)
```

```
{
  uint64_t tmp[NFRAMES];

  for (int i=0;i<NFRAMES;i++) { // Loop A
    uint64_t result=0;
    int finalpos=1;
    frame_type val=frames[i];
    for (int j=0;j<64;j++) { // Loop B
      uint8_t bitloc=bitlocs[j];
      for (int k=128;k>0;k=k/2) { // Loop C
          if ((bitloc&0x01)==1)
              val=val/k;
          bitloc=bitloc/2;
       }
      if ((val&0x01)==1)
          result|=finalpos;
      finalpos=finalpos*2;
     }
    tmp[i]=result;
  }
  int result_count=0;
  int state=0;
  for (int i=0;i<NFRAMES;i++) { // Loop D
    uint64_t val64=tmp[i]; // val64 is input to pipeline pix
    for (int b=0;b<8;b++)  // Loop E (also shown in pipeline pix)
     {
       // <see pipeline in next box; complete code for question 4>

       // assign ID_enable register
       int Id_enable = (b == 7);

       /**
        * Select the correct val8 based on b reg val
        *
        * Using a switch-case statement here would be
        * most similar to the mux select operation
        * described in the diagram.
        */
       uint8_t val8 = val64 >> 8*b && 0xFF;
```

```
        uint32_t key = (val8 << STATELEN) | state;
        uint16_t init_lookup_result = init_lookup[val8];

        // Bitwise-XOR operation
        // Loop F

        // stores temp result of bitwise-xor operations
        uint8_t temp1[20];
        uint8_t temp2[20];
        uint8_t temp3[20];

        for (int j = 0; j < 20; ++j)
          temp1[j] = key[j] ? bitpos[j] : 0;

        int d = 20;
        while(d > 1){
          for(int k = 0; k < d-1; k+=2)
            temp2[k] = temp1[k] ^ temp1[k + 1];
            temp3 = temp1;
            temp1 = temp2;
            temp2 = temp3;
            d /= 2;
        }

        // Loop G - unrolled for ease of writing

        // last output of the bitwise-xor gets sent to lookup
        lookup_result_type result = lookup[temp1[0]];
        lookup_result_type temp4[4];
        // inputs to muxes
        temp4[3] = (key[3] == result[127:108]) ? val8[3] : 0;
        temp4[2] = (key[2] == result[95:76]) ? val8[2] : 0;
        temp4[1] = (key[1] == result[63:44]) ? val8[1] : 0;
        temp4[0] = (key[0] == result[31:12]) ? val8[0] : 0;

        // use each of these compare equals outputs to select either slot_val
or 0

        // then perform 2 levels of bitwise ors
```

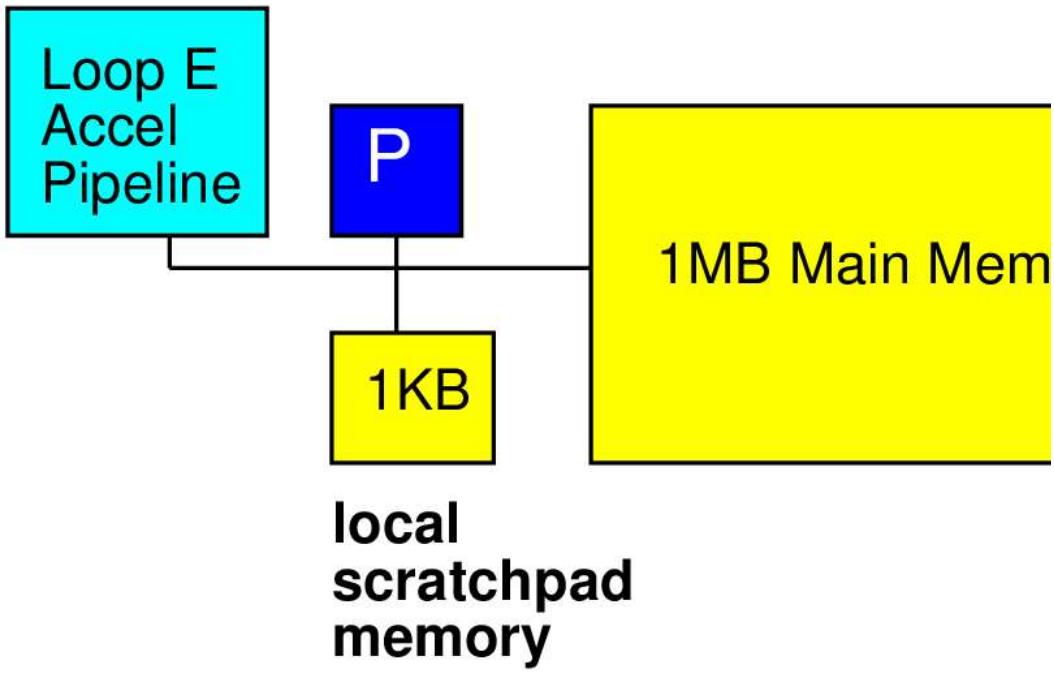
```
        // compare equals step to select between bitwise or or init_lookup out
put

        // Write compare equals output to results memory

        // increment ld_enable
      }
  }
  *num_results=result_count;
  return;
}
```

Consider the following baseline system:

We start with a baseline, single processor system as shown.



- For simplicity throughout, we will treat non-memory indexing adds (subtracts count as adds), compares, logical operations (&&, ||,|,^&), min, max, divides, and multplies as the only compute operations. We'll assume the other operations take negligible time or can be run in parallel (ILP) with the listed compute and memory operations. (Some consequences: You may ignore loop and conditional overheads in

processor runtime estimates; you may ignore computations in array indicies.)

- Baseline processor can execute one operation (as defined previous bullet) per cycle and runs at 1 GHz.
- Reads from and writes to the 1 MB main memory issue in one cycle, but require 5 cycles of latency (including issue) to get the first 64b result; memory can supply one 64b read or write each cycle. Reads larger than 64b return 64b per cycle following the first result.
- Up to 64b reads from and writes to the 1 KB scratchpad memory take 1 cycle.
- By default, all arrays live in the main memory and all array references are to main memory.
- Assume non-array variables live in registers.
- Assume all additions are associative. Max and min are associative.
- A lookup in a small memory (1KB or small) can complete in 1ns.
- A write to the pipeline accelerator above can be performed in one cycle.

Incorrect

Question 5

0 / 5 pts

Estimate the throughput in cycles per frame for loop A running on the baseline processor.

25,101

Question 6

Not yet graded / 5 pts

Explain your throughput answer above.

Your Answer:
Latency of loop A is X cycles, and is comprised of:

- 5 + 1 + 1 + 1 = 8 cycles for fetching `val` from `frames` array
  - `frame_type` is 240b, so take 4 cycles to get the value the first time
  - `val` stored in scratchpad therafter
- Loop B runs 64 times, and inside it
  - 5 cycles to fetch `bitloc`
  - Loop C runs 128 times, and inside it
    - 1 cycle for logical and
    - 1 cycle for division of `val`
    - 1 cycle for division of `bitloc`
  - 3 more cycles for other operations after Loop C
- 5 cycles for assignment to `tmp` array

Latency = 8 + 64 * (5 + 128 * 3 + 3) + 5
= 25,101 cycles

Question 7

4 / 4 pts

Where is the bottleneck in throughput processing frames?

- (●) Loop A compute
- ( ) Loop A memory
- ( ) Loop E compute
- ( ) Loop E memory

Question 8

Not yet graded / 4 pts

What is the Amdahl's Law speedup if you were to accelerate the bottleneck identified in the previous question? Support your answer with calculations.

Your Answer:
For Loop A:
- 25,101 cycles
- CPU at 1e9 Hz
- Latency
  - 25,101 cycles / 1e9 cycles per second
  - 2.5101e-5 seconds
  - 25101 ns

Out of the above time estimate, compute operations (describe below) take up a majority of the time

- Loop B runs 64 times, and inside it
  - Loop C runs 128 times, and inside it
    - 1 cycle for logical and
    - 1 cycle for division of `val`
    - 1 cycle for division of `bitloc`
  - 3 more cycles for other operations after Loop C

Latency = 64 * (128 * 3 + 3)
= 24,768 cycles

For Loop E:
- Latency is 16.4 ns

Maximum possible Speedup according to Amdahl's Law is 1/(1-Y). Y represents the fraction of the code you accelerate

In our case, Y = compute time of non-memory operations in Loop A

Y = 24,768 / 25,101 = 0.9867

Thus, best case speedup = 1/(0.013266) = 75.378x speedup

Incorrect

Question 9

0 / 4 pts

What is the smallest granularity that you can profitably stream data between Loop A and Loop E?

- ○ Entire tmp[] (all NFRAMES words in tmp[], each of which is a 64b word)
- ◉ single 64b word
- ○ no streaming possible

Question 10

Not yet graded / 10 pts

Use the scratchpad memory to accelerate memory operations in Loop A.

Indicate which data you place in the scratchpad.

Provide code or other clear description of how you modify the provided code for Loop A to exploit the scratchpad memory.

Use part of this box to provide justification to the numerical answer in the next question.

Your Answer:
We know that up to 64b reads from and writes to the 1 KB scratchpad me mory take 1 cycle.

```cpp
uint64_t tmp[NFRAMES];              // scratchpad
frame_type frames_prefetched[2];    // scratchpad
uint8_t bitlocs[64];                // scratchpad
```

Copy `tmp` and `bitlocks` into 64MB on-
chip memory at beginning of function and operate on it from there.

Prefetch the relevant frame `val` and the next one from the frames array a
t every iteration.

All references to `frames` are changed to `frames_prefetched`.

Incorrect

Question 11

0 / 4 pts

For your revised code in the previous question, what is the throughput in cycles per frame for the revised implementation of Loop A?

24,835

Partial

Question 12

8 / 14 pts

Classify each loop as sequential, reduce, or data parallel:

- Loop A data parallel
- Loop B sequential
- Loop C sequential
- Loop D data parallel
- Loop E sequential
- Loop F sequential
- Loop G data parallel

Answer 1:

data parallel

Answer 2:

sequential

Answer 3:

sequential

Answer 4:

data parallel

Answer 5:

sequential

Answer 6:

sequential

Answer 7:

data parallel

Incorrect

Question 13

0 / 5 pts

What is the cycle-bound II (unlimited hardware) for loop A (assuming no bottleneck on input frames or output tmp)?

10

Unanswered

Question 14

0 / 5 pts

What is the latency bound (unlimited hardware) for loop A executing all NFRAME frames (assuming no bottleneck on input frames or output tmp)?

Unanswered

Question 15

Not yet graded / 5 pts

Support your numeric answers to the previous two questions.

Your Answer:

Unanswered

Question 16

Not yet graded / 9 pts

Describe a VLIW architecture (types of operators and numbers of each, custom memories (memory ports) as needed) for executing Loop A that has a Resource Bound throughput (cycles per frame) that is the same throughput as the pipeline for Loop E such that Loop A is no longer the bottleneck. For simplicity, you may assume a monolithic, multi-ported register file. Try to identify an architecture with minimum hardware achieving the target resource bound. Provide description and calculations to support your answer.

Your Answer:

Unanswered

Question 17

Not yet graded / 5 pts

Given the resources identified in the previous question, how close to the target resource bound will a scheduled computation achieve? Explain your reasoning.

Your Answer:

Quiz Score: **13** out of 100