CS61C Fall 2012 Final
Examination Booklet

Instructions:

This exam is closed book and closed notes. You are allowed a single two-sided crib sheet, and you do not need a calculator. It is worth **90** points and represents 20% of your course grade. It contains **10** sets of multiple-choice questions on **13** numbered pages, including this cover page.

You are also provided with a single two-sided **Answer Sheet.** Be sure to fill out your name, Student ID, and course login on the answer sheet. Please circle one correct option per question. You can work on problems on this booklet; however, **only the answers on the Answer Sheet will be collected and graded.**

The MIPS Green Card is appended to the examination booklet for reference.

If your course login cs61c-?? falls into the following ranges, you should be in the following rooms:

- ab – jl in 220 Hearst Gym
- jm – wh in 230 Hearst Gym
- wi – xm in 242 Hearst Gym

Question 1: *SIMD (14 points total)*

Remember Polynomial Differentiation:

$$\frac{d}{dx}(a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}) = a_1 + 2a_2x + \cdots + (n-1)a_{n-1}x^{n-2}$$

A degree $n-1$ polynomial is uniquely determined by n integer coefficients $a_0, \ldots, a_{n-1}$, which we can represent as an array of integer values. For example, $f(x) = 1 + 3x^2$ can be represented by:

```
int f[] = {1, 0, 3};
```

Taking the derivative of a polynomial reduces its degree by 1 (*Hint: What are the implications for the length of the array into which we store the derivative?*).

Consider the following function:

```
/* Differentiates the polynomial A and stores the result in A_PRIME.
 * a_0 is assumed to be stored at A[0], and A is a degree N-1
 * polynomial.
 */
void differentiate(int32_t *A_prime, int32_t *A, size_t n) {
    int i;
    for (i = 0; i < END; i += 1) {
        A_prime[i] = J * A[J];
    }
}
```

The function has two macros, `END` and `J` that are undefined. In the next two questions, select the one macro definition that results in correct behavior.

i. (1 point) Select the correct definition for the macro `END`.

(a) `#define END (n)`
(b) `#define END (n/4)`
(c) `#define END (n + 1)`
(d) `#define END (n - 1)`
(e) `#define END (n/4*4)`

ii. (1 point) Select the correct definition for the macro `J`.

(a) `#define J (i)`
(b) `#define J (4*i + 1)`
(c) `#define J (i - 1)`
(d) `#define J (4*i)`
(e) `#define J (i + 1)`

To improve the run-time, we will use SIMD to differentiate four terms at a time:

```
/* Differentiates the polynomial A and stores the result in A_PRIME.
 * a_0 is assumed to be stored at A[0], and A is a degree N-1 polynomial.
 */
void differentiate_SIMD(int32_t *A_prime, int32_t *A, size_t n) {
    int i, j = J_INIT;
    int32_t tmp[] = STEROIDS_INIT;
    __m128i j_on_steroids = _mm_loadu_si128(tmp);

    /* fringe */
    differentiate(A_prime, A, START + 1);

    /* translate j, j_on_steroids by START */
    j += START;
    j_on_steroids = _mm_add_epi32(j_on_steroids, _mm_set1_epi32(START));

    /* main loop */
    for (i = START; i < END; i += 4) {
        __m128i A_chunk = _mm_loadu_si128(&A[j]);
        _mm_storeu_si128(&A_prime[i], _mm_mul_epi32(j_on_steroids, A_chunk));
        j += 4;
        j_on_steroids = _mm_add_epi32(j_on_steroids, _mm_set1_epi32(4));
    }
}
```

The above has four undefined macros `J_INIT`, `STEROIDS_INIT`, `START`, and `END`. Select the one macro definition that leads to the best-performing correct behavior.

iii. (3 points) Select the correct definition for the macro `START`:

(a) `#define START 4`
(b) `#define START ((n-1)/4*4)`
(c) `#define START 0`
(d) `#define START (n % 4)`
(e) `#define START ((n – 1) % 4)`

iv. (3 points) Select the correct definition for the macro `J_INIT`:

(a) `#define J_INIT 0`
(b) `#define J_INIT 1`
(c) `#define J_INIT 2`
(d) `#define J_INIT 4`
(e) `#define J_INIT 5`

v. (3 points) Select the correct definition for the macro `STEROIDS_INIT`:

(a) `#define STEROIDS_INIT {0, 0, 0, 0}`
(b) `#define STEROIDS_INIT {1, 1, 1, 1}`
(c) `#define STEROIDS_INIT {0, 1, 2, 3}`
(d) `#define STEROIDS_INIT {1, 2, 3, 4}`
(e) `#define STEROIDS_INIT {4, 4, 4, 4}`

vi. (3 points) Select the correct definition for the macro `END`:

(a) `#define END (n – 1)`
(b) `#define END (n)`
(c) `#define END ((n – 1)/4*4)`
(d) `#define END (n/4*4)`
(e) `#define END (n/4)`

Question 2: *OpenMP (5 points total)*

Circle the one choice that results in the fastest parallel code with the same output as the initial serial code.

Note: parallel/critical block around "…" => `#pragma omp parallel/critical { ... }`

i. (3 points)

Serial:

```
int i;
for (i = 0; i < len_list; i += 1)
    total += list[i];
for (i = 0; i < len_result; i += 1)
    result[i] = total*i;
```

Parallel:

```
1   for (int i = omp_get_thread_num(); i < len_list; i += omp_get_num_threads())
2       total += list[i];
3   for (int i = omp_get_thread_num(); i < len_result; i += omp_get_num_threads())
4       result[i] = total*i;
```

(a) Add a parallel block around lines 1-2 and a parallel block around lines 3-4
(b) Add a parallel block around lines 1-2, a parallel block around lines 3-4, and a critical block around line 2
(c) Add a parallel block around lines 1-4 and a critical block around line 2
(d) Add a parallel block around lines 1-4
(e) None of the above causes the code to return the correct result

ii. (2 points)

Serial:

```
for(int i = 0; i < m; i += 1) {
    total += i;
    res[i] = total;
}
```

Parallel:

```
0   #pragma omp parallel
1   {
2     for(int i = omp_get_thread_num(); i < m; i += omp_get_num_threads())
3     {
4       total += i;
5       res[i] = total;
6     }
7   }
```

(a) Add a critical block around lines 2-6
(b) Add a critical block around lines 4-5
(c) Add a critical block around line 4
(d) The parallel code returns the correct result the fastest without adding anything
(e) None of the above causes the code to return the correct result

Question 3: *Potpourri (5 points total)*

i. (1 point) How many transistors in a warehouse-scale computer, to the nearest order of magnitude?
(a) 10^{14}
(b) 10^{17}
(c) 10^{20}
(d) 10^{23}
(e) 10^{26}

ii. (1 point) Which of these are a drawback of using polling instead of interrupts for I/O devices?
(a) Loss of Data
(b) Wasted CPU Time
(c) Increased Control Hazards
(d) Both (a) and (b)
(e) None of the above

iii. (1 point) Which of these is an asynchronous exception?
(a) Arithmetic Overflow
(b) Misaligned Memory Access
(c) Timer Expiration
(d) Divide By Zero
(e) Cache Miss

iv. (1 point) System *availability* can best be improved by:
(a) Increasing Mean Time To Failure while increasing Mean Time To Repair
(b) Increasing Mean Time To Failure while decreasing Mean Time To Repair
(c) Decreasing Mean Time To Failure while increasing Mean Time To Repair
(d) Decreasing Mean Time To Failure while decreasing Mean Time To Repair
(e) None of the above

v. (1 point) The following 12-bit word encodes 8 data bits and 4 parity bits using the standard Hamming Error Correcting Code Technique:

0 0 1 1 1 0 1 0 1 0 1 1

Which single one of the following is correct?
(a) The word is correct as is, and has no bits in error.
(b) The word has one bit in error, and its correction is 1 0 1 1 1 0 1 0 1 0 1 1
(c) The word has one bit in error, and its correction is 0 1 1 1 1 0 1 0 1 0 1 1
(d) The word has one bit in error, and its correction is 0 0 1 1 1 0 1 1 1 0 0 1
(e) The word has one bit in error, and its correction is 0 0 1 1 1 0 1 0 1 0 0 1

Question 4: *Boolean Logic Simplification and Equivalence (5 points total)*
Consider the four-input Boolean function

$$F(A, B, C, D) = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,D + A\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C\,D + A\,B\,C\,D + \overline{A}\,B\,C\,\overline{D} + A\,\overline{B}\,C\,\overline{D}$$

Note that $\oplus$ is the logical symbol for XOR (also known as the *difference function*; i.e., $A \oplus B$ is true if and only if its inputs have different Boolean values). The function is commutative, $A \oplus B = B \oplus A$, and associative, $A \oplus B \oplus C = (A \oplus B) \oplus C = A \oplus (B \oplus C)$.

i. (5 points) Select which one of the following equations is an equivalent form for F (only one is correct)

(a) $A \oplus B \oplus C \oplus D$

(b) $\overline{A \oplus B \oplus C \oplus D}$

(c) $\overline{A} \oplus \overline{B} \oplus \overline{C} \oplus \overline{D}$

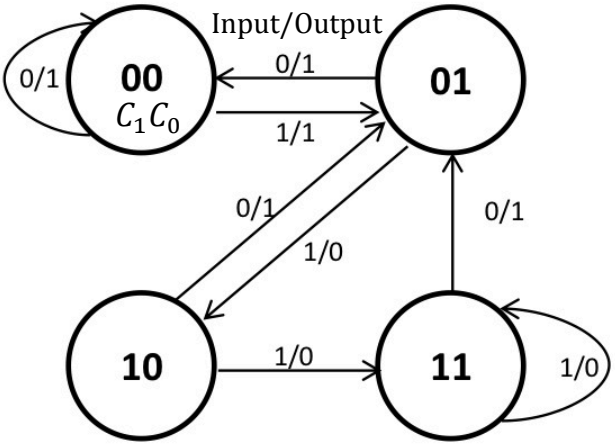
(d) $(A \oplus B)\,(C \oplus D) + (\overline{A} \oplus \overline{B})\,(\overline{C} \oplus \overline{D})$

(e) $\overline{A}\,\overline{B} + \overline{A}\,\overline{C} + \overline{C}\,\overline{D} + \overline{A}\,\overline{D}$

Question 5: *Finite State Machine (10 points total)*
Given the following transition diagram for a finite state machine, please answer the given questions.

Note: State labels are to be treated as 2-bit unsigned integers.
The current state is denoted as C with C_i being the i^{th} bit in C.
The next state is denoted as N with N_i being the i^{th} bit in N.

Input is denoted as I, and output is denoted as O.



Below is a scratch truth table for your use. It will not be graded as part of this question.

C_1	C_0	I	N_1	N_0	O
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

i. (5 points) Which of the following Boolean expressions correctly express the next state and the output?

(a) $N_1 = \overline{\overline{(C_1 + C_0)}\,\overline{I}}$
$N_0 = \overline{C_0}I + C_1$
$O = \overline{I\overline{(C_1 + C_0)}}$

(b) $N_1 = \overline{\overline{C_1\,C_0} + \overline{I}}$
$N_0 = C_1 + I$
$O = \overline{I}\,(\overline{C_1} + \overline{C_0})$

(c) $N_1 = (C_0 + C_1)I$
$N_0 = \overline{\overline{C_0 + \overline{I}} + C_1}$
$O = \overline{C_1}\,\overline{C_0} + \overline{I}$

(d) $N_1 = IC_0 + IC_1$
$N_0 = \overline{\overline{(C_0 + \overline{I})}\,\overline{C_1}}$
$O = \overline{I}\,\overline{(C_1 + C_0)}$

(e) None of the above

ii. (3 points) Using only NOT and two-input basic gates (AND, OR), and the current state and the input, what is the minimum number of gates needed to build the next state and output circuits?

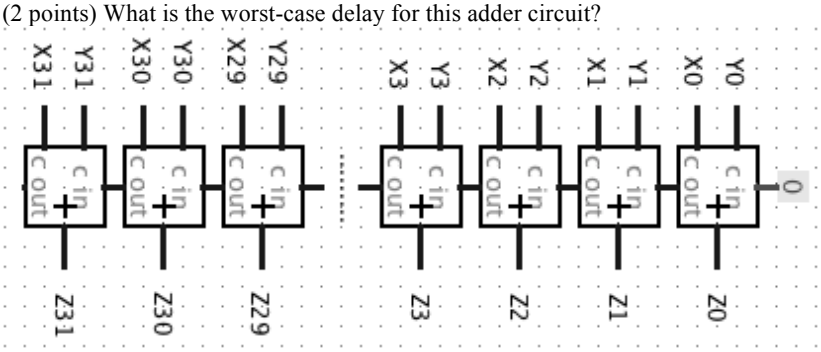
(a) $N_1 = 2$, $N_0 = 4$, $O = 4$
(b) $N_1 = 2$, $N_0 = 3$, $O = 4$
(c) $N_1 = 1$, $N_0 = 2$, $O = 3$
(d) $N_1 = 3$, $N_0 = 3$, $O = 4$
(e) None of the above

iii. (2 points) Name the event when the output is 1.
(a) The next state is a prime number.
(b) The next state is an odd number.
(c) The next state is lower than 2.
(d) All of the above.
(e) None of the above.

Question 6: *Adder (5 points total)*

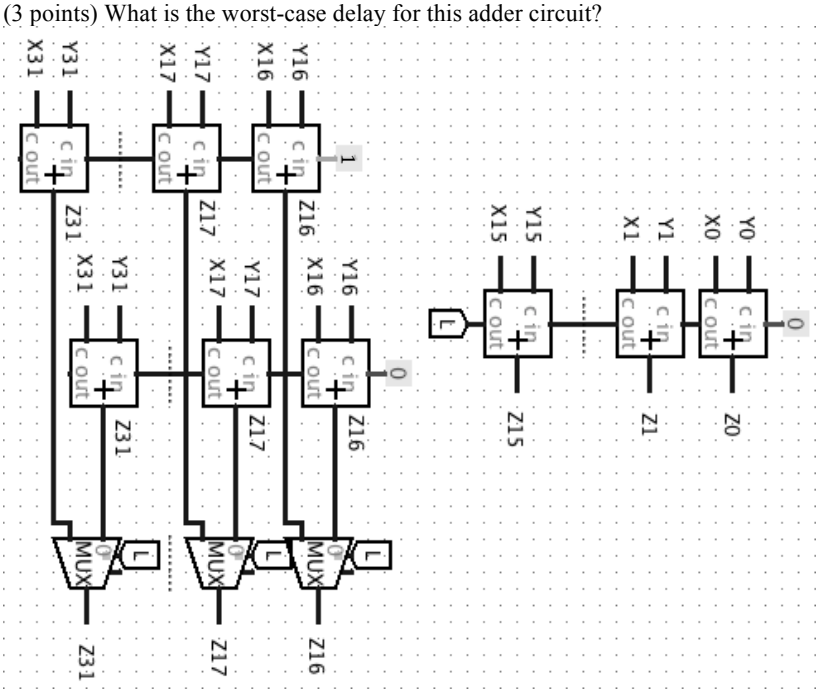
In the circuits below, each single bit adder has worst-case 5 ns delay, and each mux 20 ns worst-case delay.

i. (2 points) What is the worst-case delay for this adder circuit?



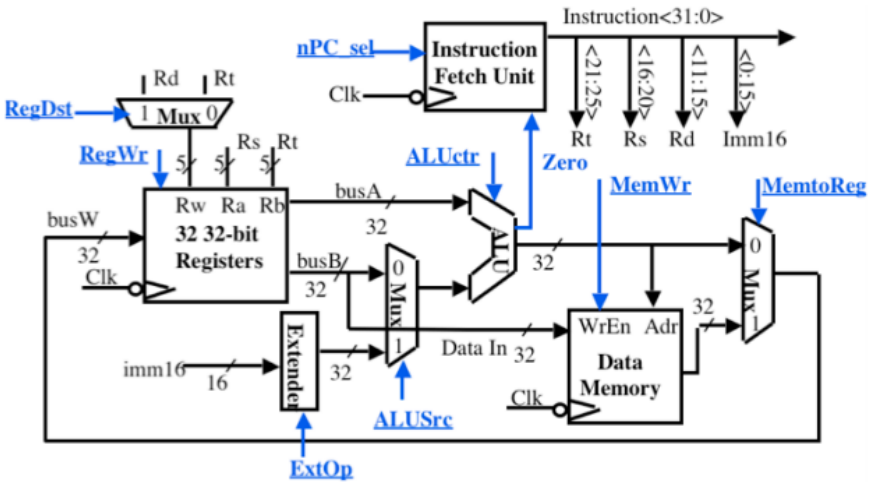
(a) 5ns
(b) 40ns
(c) 160ns
(d) 320ns
(e) None of the above

ii. (3 points) What is the worst-case delay for this adder circuit?

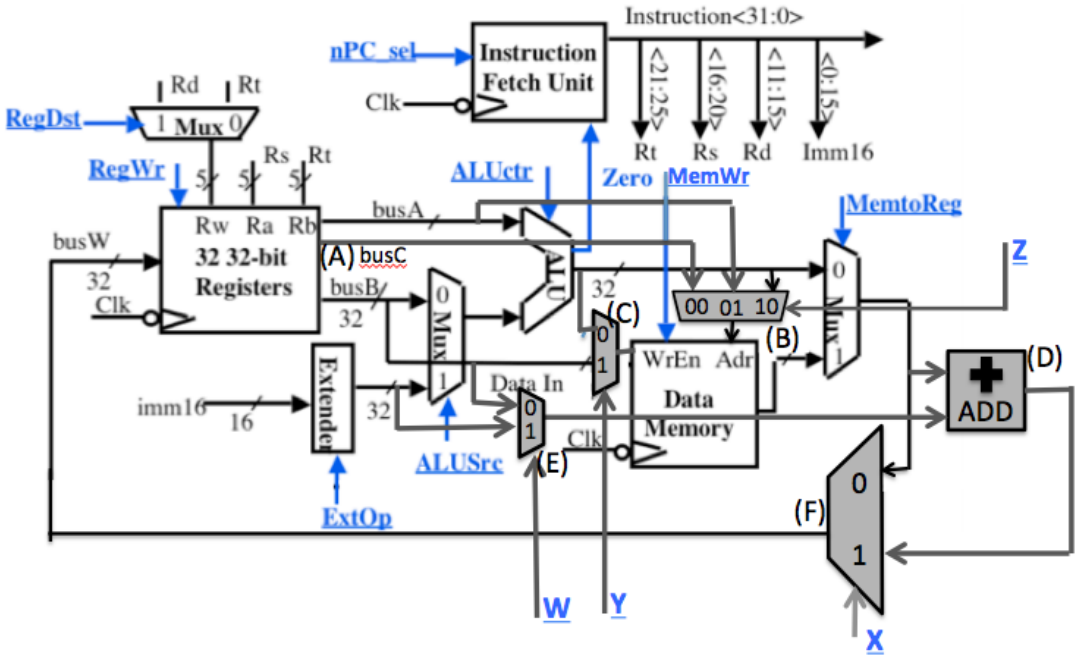


(a) 25ns
(b) 85ns
(c) 90ns
(d) 100ns
(e) None of the above

Question 7: *Single-Cycle CPU (16 points total)*
Given below is a **single-cycle non-pipelined** MIPS datapath:



The datapath is modified with **SIX** changes, labeled **(A)-(F)**, to support single-cycle memory access and addition instructions:



(A): The Register File is extended with an additional read port, with the address coming from rd, and the data output, R[rd], being driven out to a new busC.
(B): Mux busC, busA and ALU output into Memory Address slot, with selection signal Z.
(C): Mux ALU output and busB into the Memory's data input slot, with selection signal Y.
(D): Add an adder with the MemToReg output as one of the inputs into the adder.
(E): Mux busB and Extended Immediate into the second input of the adder, with selection signal W.
(F): Mux the MemtoReg output and the adder output, and feed it into the RegFile (busW), with selection signal X.

In the modified datapath, you have a total of 12 control signals. Below is a table with alternative control settings:

	RegDst	RegWr	nPC_sel	ExtOp	ALUSrc	ALUCtr	MemWr	MemtoReg	W	X	Y	Z
#1	0	1	+4	Sign	1	Add	0	1	X	0	X	10
#2	X	0	+4	X	0	Add	1	X	X	X	0	00
#3	1	1	+4	X	X	X	0	X	0	1	X	01
#4	X	0	+4	X	0	Add	1	0	X	0	0	10
#5	1	1	+4	X	X	X	0	1	0	1	X	01
#6	0	1	+4	Sign	X	X	0	1	1	1	X	01

Note: "X" signifies "don't care."

For each of the following instructions, pick the best option that maximizes the number of "don't cares" while performing the correct operation.

i. (4 points) `memaddr rd rs rt: R[rd] = R[rt] + M[R[rs]]`
Takes the value in register rt, adds it to the value in memory at location given by value of rs, and stores it in register rd.
(a) Option #1
(b) Option #3
(c) Option #5
(d) Option #6
(e) None of the above

ii. (4 points) `memaddi rt imm rs: R[rt] = SignExtImm + M[R[rs]]`
Takes the value in memory at location given by value of rs, adds it to the immediate, and stores it in register rt.
(a) Option #1
(b) Option #2
(c) Option #4
(d) Option #6
(e) None of the above

iii. (4 points) `storeadd rd rs rt: M[R[rd]] = R[rt] + R[rs]`
Takes the value in the register rs, adds it to the value in register rt, and stores it in memory at location given by value of register rd.
(a) Option #2
(b) Option #3
(c) Option #4
(d) Option #5
(e) None of the above

iv. (4 points) `lw rt imm rs: R[rt] = M[R[rs] + SignExtImm]`
Add the value in the register rs to immediate to obtain a memory location; store the value at that location into register rt.
(a) Option #1
(b) Option #3
(c) Option #5
(d) Option #6
(e) None of the above

Question 8: *Pipelined CPU (10 points total)*
Consider the 5-stage **single-issued pipelined** MIPS datapath consisting of Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory (MEM), and Write-Back (WB).

You are given the following MIPS instruction sequence:

```
# $s0 to $s3 = 56, 30, 30, 7
# $t0 to $t4 = 7, 7, 7, 7, 7
       add $t0, $s0, $0
       and $t1, $t0, $s1
       or $t2, $t0, $s2
       sub $t3, $t0, $s3
       srl $t4, $t0, 2
```

Start numbering the cycles with 1 when the add instruction enters the IF stage.

For part i. to iii. assume that the datapath is broken and there is no forwarding and no stalling.

i. (2 points) What are the values of `$t0` to `$t4` at the end of Cycle 7?
(a) 56, 24, 62, 7, 7
(b) 56, 24, 62, 49, 7
(c) 56, 6, 31, 7, 7
(d) 56, 6, 7, 7, 7
(e) None of the above

ii. (2 points) What are the values of `$t0` to `$t4` at the end of cycle 8?
(a) 56, 24, 62, 49, 7
(b) 56, 24, 62, 49, 14
(c) 56, 6, 31, 49, 7
(d) 56, 6, 31, 7, 7
(e) None of the above

iii. (2 points) What are the values of `$t0` to `$t4` at the end of cycle 9?
(a) 56, 24, 62, 49, 7
(b) 56, 24, 62, 49, 14
(c) 56, 6, 31, 49, 14
(d) 56, 6, 31, 0, 7
(e) None of the above

iv. (2 points) What instruction(s) is/are computing the wrong result(s) (choose the answer that includes ALL faulty instructions)?
(a) `add`
(b) `and, or`
(c) `and, or, sub`
(d) `and, or, sub, srl`
(e) `add, and, or, sub, srl`

v. (2 points) Say we want to completely fix the problem from part iv. using forwarding. Which forwarding path(s) do we need to provide in order to execute the code sequence correctly (it is implied that multiplexers are inserted to join the forwarded signals with the original signals)?
(a) Output of ALU in the EX stage back to the input of the ALU in the EX stage.
(b) Output of ALU in the MEM stage back to the output of Register File in the ID stage.
(c) Output of ALU in the MEM stage back to the input of ALU in the EX stage.
(d) Both (a) and (b).
(e) Both (a) and (c).

Question 9: *Caches (10 points total)*

Compare the performance of two cache designs for a byte-addressed memory system. The first cache design is a direct-mapped cache (DM) with four blocks, each block holding one four-byte word. The second cache has the same capacity and block size but is fully associative (FA) with a least-recently-used replacement policy.

For the following sequences of memory read accesses to the cache, compare the relative performance of the two caches. Assume that all blocks are invalid initially, and that each address sequence is repeated a large number of times. **Ignore compulsory misses when calculating miss rates.** All addresses are given in decimal.

i. (2 points) **Memory Accesses:** 0, 4, 0, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	0%	0%
(b)	0%	100%
(c)	100%	0%
(d)	100%	50%
(e)	100%	100%

ii. (3 points) **Memory Accesses:** 0, 4, 8, 12, 16, 0, 4, 8, 12, 16, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	20%	0%
(b)	40%	0%
(c)	20%	20%
(d)	40%	100%
(e)	100%	100%

iii. (5 points) **Memory Accesses:** 0, 4, 8, 12, 16, 12, 8, 4, 0, 4, 8, 12, 16, 12, 8, 4, (repeats)

	DM Miss Rate	FA Miss Rate
(a)	25%	0%
(b)	25%	25%
(c)	50%	0%
(d)	50%	100%
(e)	100%	100%

Question 10: *Virtual Memory (10 points total)*
You have the following memory configuration on a MIPS system:

- Virtual Address Space of 4GB.
- 16 MB of byte-addressed physical memory.
- 4KB page size.

i. (2 points) Which of the following is the correct number of VPN bits, Page Offset Bits (VPN, Offset):
(a) 22, 10 (b) 18, 14 (c) 20,12 (d) 20, 10 (e) 18, 16

ii. (2 points) How many bits does a physical address have and what is the maximum number of pages that can be actively held in memory (AddrBits, MaxPages):
(a) 24, 2^{12} (b) 20, 2^{10} (c) 22, 2^{16} (d) 28, 2^{12} (e) 26, 2^{16}

Note: Note: You may assume that only the following program is running on the entire machine. Additionally, you may assume that memory is only used for the stack and heap.

```
#define NUM_NODES 16777216 /* = 2^24 */
typdef struct node {         // a node struct for a singly-linked list.
       struct node *next;    // a pointer to the next node.
       int pin;              // a 4-byte integer to store a pin.
} snode;

static snode nodes[NUM_NODES]; //Create an array of pointer nodes. Note: This does not
                               //link them. Assume array starts at Virtual Address 0x0.
void main()
{
       int i, pin;

       /* This function will make each node in the nodes array point to another,
        * so that all nodes are covered. Eg. If we have the array {s0,s1,s2,s3}, then
        * possible linked lists are:
        *       s0 -> s1 -> s3 -> s2
        *       s0 -> s3 -> s2 -> s1
        *              ...
        * NOTE: The singly-linked list will always start from the 0th element of the array.
        */
       CreateSinglyLinkedLinkedList(nodes);
       pin = getSearchPin(); //gets Pin from user

       snode *ptr = &nodes[0];  //We point ptr to the head of the list
                                //(the 0th element of the array).
       for (i = 1; ptr != NULL; i++) {
              if (ptr->pin == pin) break;  //Search for matching pin
              ptr = ptr->next;             //move ptr forward to the next node
       }
}
```

Given that the `CreateSinglyLinkedList` function can create any possible permutation of snodes on the list, and **considering only the for loop** in the given code (and assuming that the loop variable is stored in a register and not in memory), answer the following questions by selecting one option.

iii. (3 points) In the best-case scenario, what percentage of data memory accesses causes a page fault?
(a) 0% (b) 1% (c) 2% (d) 3% (e) 4%

iv. (3 points) In the worst-case scenario, what percentage of data memory accesses causes a page fault?
(a) 20% (b) 33% (c) 50% (d) 66% (e) 100%