Name: ______________________

First Name: ______________________

Student ID: ______________________

2nd session examination

Design of Digital Circuits SS2013

(252-0014-00S)

Markus Püschel, Frank K. Gürkaynak

Examination Rules:

1. Written exam, 90 minutes total.
2. No books, no calculators, no computers or communication devices. Five pages of handwritten notes are allowed.
3. Write all your answers on this document, space is reserved for your answers after each question. Blank pages are available at the end of the exam.
4. Put your Student ID card visible on the desk during the exam.
5. If you feel disturbed, immediately call an assistant.
6. Answers will only be evaluated if they are readable
7. Write with a black or blue pen (no pencil, no green or red color).
8. Show all your work. For some questions, you may get partial credit even if the end result is wrong due to a calculation mistake.

Question:	1	2	3	4	5	6	7	Total
Points:	5	16	15	18	5	10	6	75
Score:								

This page intentionally left blank

1. (a) (2 points) For the following four numbers given in decimal or hexadecimal notation, write the corresponding binary number using the indicated format.

$(-6)_{10}$ using 6-bit sign magnitude:	$(10\,0110)_2$
$(37)_{10}$ using 6-bit unsigned:	$(10\,0101)_2$
$(-28)_{10}$ using 6-bit two's complement:	$(10\,0100)_2$
$(2B)_{16}$ using 6-bit unsigned:	$(10\,1011)_2$

(b) (3 points) State whether the following statements about the binary representation of numbers are *true* or *false*. Give **brief** explanations for the statements that are *false*.

- Both two's complement and sign/magnitude representation can be used to represent negative numbers in binary.

Solution: True, however it is more difficult to design arithmetic circuits that work with sign/magnitude format. Still they are used.

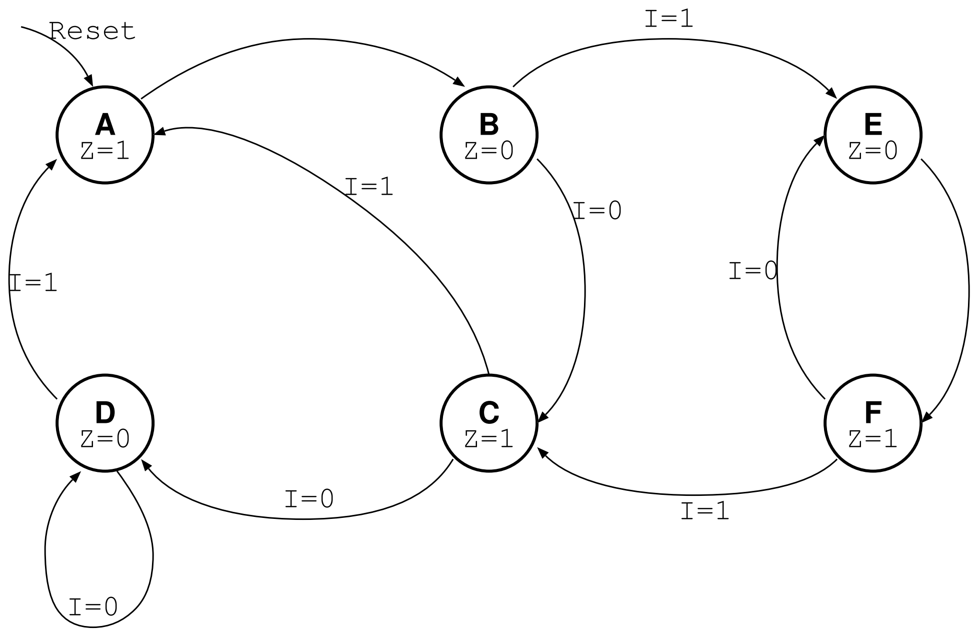
- Using N bits it is possible to represent 2^N different numbers when an unsigned number system is used.

Solution: True.

- While there are methods to represent both positive and negative integers, it is not possible to represent fractions or real numbers using binary numbers.

Solution: False, fixed and floating point number systems can be used to represent such numbers.

2. Consider the following state diagram of an FSM with 1-bit input (I) and 1-bit output (Z).



The state has been coded using a 3-bit vector $S = S_2\, S_1\, S_0$ according to the following table:

State			
name	S_2	S_1	S_0
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0
F	1	0	1

(a) (1 point) Is this a Moore or Mealy type FSM? Briefly explain.

Solution: Moore, outputs depend only on the present state and nothing else.

(b) (6 points) Fill in the following state transition table that determines the next state vector $N = N_2\, N_1\, N_0$ based on the current state S and the input I.

State				**Input**	**Next State**			
name	S_2	S_1	S_0	I	name	N_2	N_1	N_0
A	0	0	0	X	B	0	0	1
B	0	0	1	0	C	0	1	0
B	0	0	1	1	E	1	0	0
C	0	1	0	0	D	0	1	1
C	0	1	0	1	A	0	0	0
D	0	1	1	0	D	0	1	1
D	0	1	1	1	A	0	0	0
E	1	0	0	X	F	1	0	1
F	1	0	1	0	E	1	0	0
F	1	0	1	1	C	0	1	0

Note that there are different ways of writing this table to represent the same result.

(c) (3 points) Write the Next State Equations from the table you have filled above using either *Product of Sums (POS)* or *Sum of Products (SOP)* form. **Do not spend time minimizing the equations, this will be next question**.

Solution:

$$N_0 = \overline{S_2}\,\overline{S_1}\,\overline{S_0} + \overline{S_2}\,S_1\,\overline{S_0}\,\overline{I} + \overline{S_2}\,S_1\,S_0\,\overline{I} + S_2\,\overline{S_1}\,\overline{S_0}$$

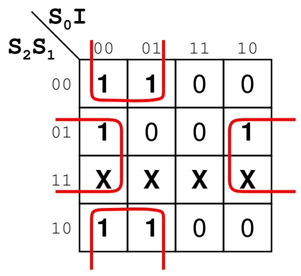
$$N_1 = \overline{S_2}\,\overline{S_1}\,S_0\,\overline{I} + \overline{S_2}\,S_1\,\overline{S_0}\,\overline{I} + \overline{S_2}\,S_1\,S_0\,\overline{I} + S_2\,\overline{S_1}\,S_0\,I$$

$$N_2 = \overline{S_2}\,\overline{S_1}\,S_0\,I + S_2\,\overline{S_1}\,\overline{S_0} + S_2\,\overline{S_1}\,S_0\,\overline{I}$$

(d) (6 points) Minimize the Next State Equations from the previous part. Note that the FSM requires only six states. This means that there are several State (S) / Input(I) combinations for which the outputs can be treated as *Don't Care*, which should help minimizing the boolean equations.
Hint: Consider using Karnaugh diagrams to solve this problem.

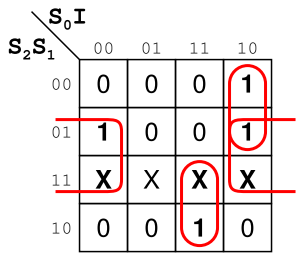
Solution:

It is important to note that for $S_2\,S_1 = 11$ the next state N can be taken as $N_2\,N_1\,N_0 = XXX$. This can simplify the Boolean equations significantly. It is best to use a Karnaugh map to find the simplifications.



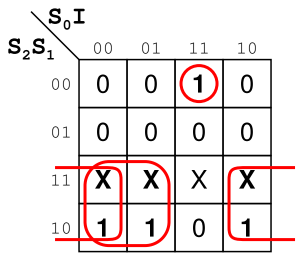
S_2S_1 \ S_0I	00	01	11	10
00	1	1	0	0
01	1	0	0	1
11	X	X	X	X
10	1	1	0	0

$\mathbf{N_0}$



S_2S_1 \ S_0I	00	01	11	10
00	0	0	0	1
01	1	0	0	1
11	X	X	X	X
10	0	0	1	0

$\mathbf{N_1}$



S_2S_1 \ S_0I	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	X	X	X	X
10	1	1	0	1

$\mathbf{N_2}$

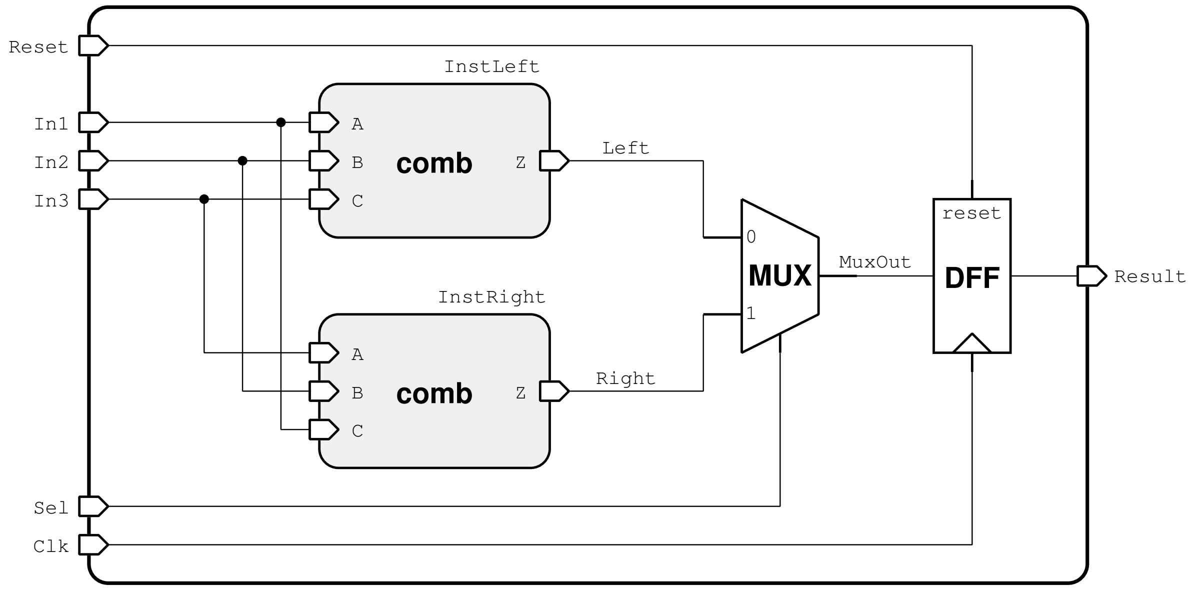
$$N_0 = \overline{S_1}\,\overline{S_0} + S_1\,\overline{I}$$

$$N_1 = S_1\,\overline{I} + S_2\,S_0\,I + \overline{S_2}\,S_0\,\overline{I}$$

$$N_2 = S_2\,\overline{S_0} + S_2\,\overline{I} + \overline{S_2}\,\overline{S_1}\,S_0\,I$$

This page intentionally left blank

3. (15 points) In this question you are required to write the Verilog code that implements the following block diagram.



The block called *Controller* has six 1-bit inputs and a single 1-bit output (*Result*). It contains two instances of a combinational block called (*comb*). The following is the declaration part of this module:

```
module comb ( input A , input B , input C, output Z);
  // definition of the combinational circuit comb

endmodule
```

Notes:

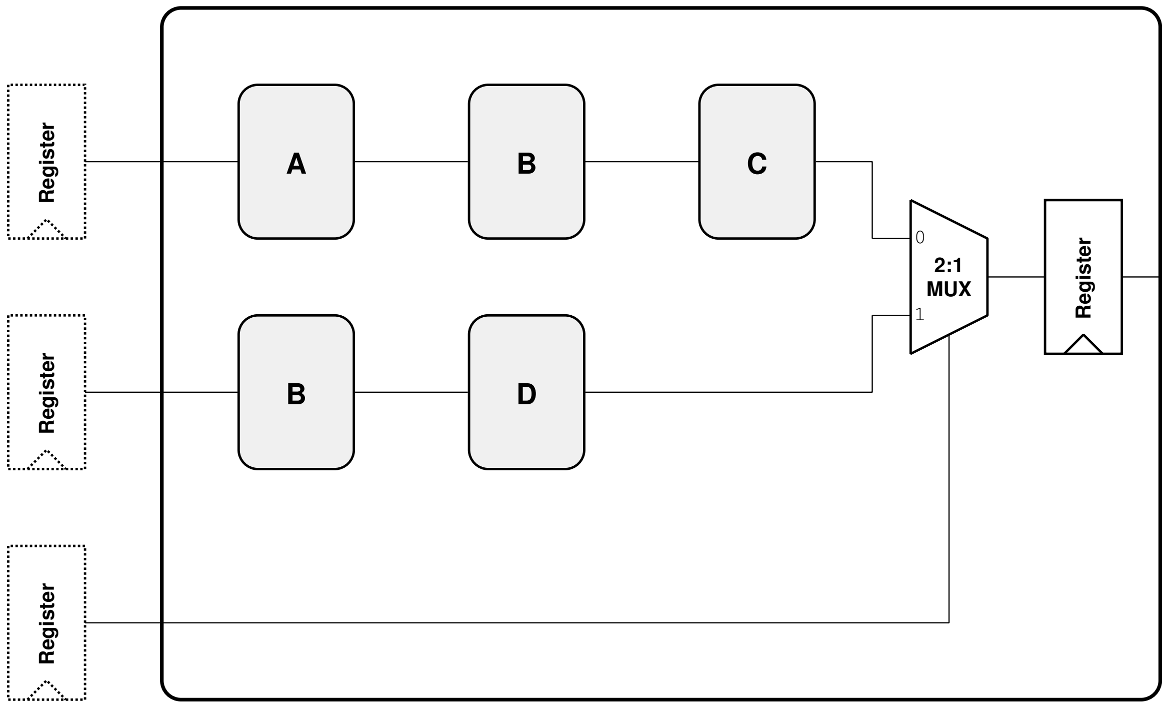
- The flip-flop and the multiplexer will not be instantiated, you will have to write the corresponding Verilog code.
- The flip-flop uses an asynchronous reset, the output is zero when reset is one.
- Note that Verilog is case sensitive.
- Write legibly.

This page intentionally left blank

Solution:

```
module Controller (
  input Clk,
  input Reset,
  input In1,In2,In3,
  input Sel,
  output reg Result);

// Define internal signals
  wire Left, Right, MuxOut;

// instantiate the module comb two times
  comb InstLeft  (.A(In1), .B(In2), .C(In3), .Z(Left) );
  comb InstRight (.A(In3), .B(In2), .C(In1), .Z(Right) );

// The multiplexer
  assign MuxOut = (Sel) ? Right: Left;

// The FF
  always @ (posedge Clk, posedge Reset)
     if (Reset) Result <= 1'b0;
     else       Result <= MuxOut;

endmodule
```

4. In this question you are required to calculate different timing paths for the circuit given below.



Note that the block *B* is used *twice*. All the inputs are supplied from external registers. The timing properties of all the blocks used in the circuit are given in the table below:

Block	Propagation Delay	Contamination Delay	Setup Time	Hold Time
Register	0.0 ns	0.0 ns	0.1 ns	0.0 ns
2:1 Multiplexer	0.3 ns	0.3 ns	n.a.	n.a.
A	0.6 ns	0.4 ns	n.a.	n.a.
B	1.8 ns	1.0 ns	n.a.	n.a.
C	1.2 ns	0.8 ns	n.a.	n.a.
D	1.0 ns	0.8 ns	n.a.	n.a.

Hint: $\frac{1}{1\,ns} = 1\,GHz$, *a clock with 1 GHz has a period of 1 ns. 1 GHz = 1000 MHz*

(a) (2 points) What is the *critical path* of this circuit? Calculate its path delay.

Solution:
Critical path is the longest timing path in the circuit:

$$\begin{aligned} CriticalPath &= t_{pd,A} + t_{pd,B} + t_{pd,C} + t_{pd,mux} + t_{setup,FF} \\ &= 0.6 + 1.8 + 1.2 + 0.3 + 0.1\, ns \\ &= 4.0\, ns \end{aligned}$$

(b) (1 point) What is the maximum operating frequency of this circuit?

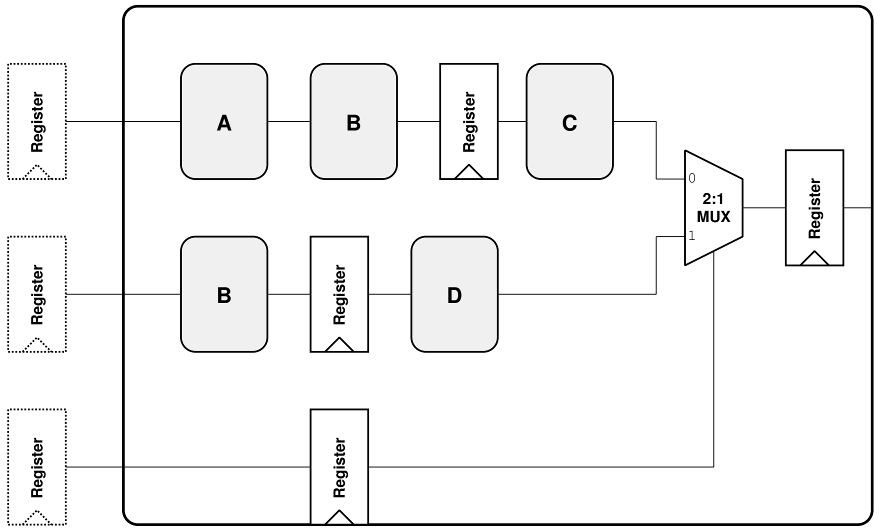
Solution: Maximum operating frequency is 1/Critical Path, $1/4\, ns = 250\, MHz$.

(c) (2 points) What is the *shortest path* of this circuit ?

Solution: The short path is the fastest route a signal can propagate through the circuit. In this case it is the signal that goes through the select input of the multiplexer and reaches the register. This signal only passes through the multiplexer, so the short path is $0.3\, ns$.

(d) (8 points) In order to make the circuit faster, it is suggested to introduce pipelining. Determine the **best** location between the blocks to place the pipeline registers assuming a **one stage** pipeline implementation and redraw the schematic. (*Hint: in a one stage pipeline, the latency increases only by 1 clock cycle*).

Solution:



Note that you will have to insert a pipeline register in all paths, including the multiplexer control register.

(e) (2 points) What is the *critical path* of this pipelined circuit and calculate its path delay.

Solution:

$$\begin{aligned} CriticalPath &= t_{pd,A} + t_{pd,B} + t_{setup,FF} \\ &= 0.6 + 1.8 + 0.1\,ns \\ &= 2.5\,ns \end{aligned}$$

(f) (1 point) What is the speed-up we have achieved by pipelining?

Solution: Old critical path / New Critical path = 4.0 / 2.5 = 1.6. The new circuit has a 60% shorter critical path, therefore it can be clocked 60% faster.

(g) (2 points) Why did we not achieve the theoretical maximum speed-up from the pipelining. State at least **two** reasons.

Solution:

1. It was not possible to place the pipeline registers at exactly half the previous critical path. Unless we move the pipeline registers into the blocks themselves we can not improve any more.
2. The pipeline register has a small overhead ($t_{setup,FF} > 0$). Even if we managed to divide the circuit in equal halves, this overhead would not allow us to reach 100% speedup. The problem gets worse the shorter the critical path is.

5. The questions 5, 6 and 7 are based on the MIPS assembly code given below.

```
        addi $t0, $0, 8
        xor  $s0, $s0, $s0
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        add  $t3, $t1, $s0
        add  $s0, $t2, $t3
        addi $t0, $t0, -1
        j    loop
done:
```

(a) (2 points) Briefly explain what the above MIPS assembly code does.

Solution:
The program will execute a loop 8 times. In each iteration of the loop, the content of the address `0x0000 0004` and the content of the address `0x0000 0024` will be added together and added to the register `$s0` which was initialized to the value 0 (A xor A is 0).

(b) (1 point) Assuming the data at memory location `0x0000 0004` is decimal 16 and at memory location `0x0000 0024` is decimal 32, what will be the content of the register `$s0` in **hexadecimal** when the program execution jumps to `done:`?

Solution: The program calculates $8 \times (mem(0x00000004) + mem(0x00000024))$. This equals to $8 \times (32 + 16) == 384$. In hexadecimal this will be $0x0180$. It is actually very easy to do the calculation if you do it in binary.

(c) (2 points) Briefly explain the three different MIPS instruction types (R, I, J) and show **one** instruction of each type from the example code above.

Solution:

R-type Uses up to three registers, two for source and one for destination.
Example: `xor $s0, $s0, $s0`.

I-type Uses a 16-bit constant as part of the instruction.
Example: `addi $t0, $0, 8`.

J-type Uses a 26-bit constant that can be used to calculate the address of the next instruction, allowing the program execution to jump to a new location.
Example: `j loop`.

6. In this question you are required to make calculations regarding the processor's speed. For this question, use the MIPS assembly program from the previous section.

   (a) (2 points) How many instructions are executed until the program finishes?

   Solution: There are two initial instructions, and the loop contains 7 instructions which is executed 8 times, in addition there is one last `beq` command that will be executed after the last loop. $N_{instructions} = 2 + (7 \times 8) + 1 = 59$ (56, or 58 could also be accepted)

   (b) (1 point) Assume that you are using a single cycle microarchitecture running at 1 GHz. If the CPI (Cycles Per Instruction) is 1, how long will it take for the program to finish execution?

   Solution: 59 instructions x 1 CPI = 59 clock cycles. At 1 GHz, 1 clock cycles is 1 ns, so the entire program will execute in 59 ns.

   (c) (2 points) In a second microarchitecture, due to a very slow memory, every *load word* (`lw`) instruction requires 10 clock cycles. Calculate the number of cycles needed to execute the entire program.

   Solution: The loop contains 2 `lw` instructions each take 10 clock cycles. The remaining 5 instructions take 1 clock cycle. Every loop takes 25 clock cycles. There are 8 iterations of the loop plus two additional initial instructions and one last `beq`: $N_{cycles} = 2 + (8 \times (5 \times 1 + 2 \times 10) + 1 = 203$ (200, or 202 could also be accepted)

   (d) (1 point) For this second microarchitecture, what is the average CPI? (*approximate numbers ±10% are ok*)

   Solution: There are 59 instructions that are executed in 203 clock cycles. $CPI = 203/59 = 3.44$. Acceptable are also approximate calculations $CPI = 200/60 = 3.33$ or $CPI = 210/60 = 3.5$

(e) (4 points) For the second microarchitecture with the slow memory, what can be done in both the code and the processor to improve the speed of this particular program other than using a faster memory. Write two short suggestions that would have the largest impact and briefly explain why that would improve the performance.
Note: The result of the program should depend on the two values stored in the memory, these can not assumed to be constants.

Solution:

- The most obvious problem is the slow memory access. Introducing a proper cache could alleviate this problem. The first iterations would still be slow, but the remaining accesses would be fast.
- The program could be written more efficiently. The loop essentially multiplies the sum of the two values in memory by eight. Instead of a loop, a shift left operation could be used.

```
lw   $t1, 0x4($0)
lw   $t2, 0x24($0)
add  $s0, $t1, $s2
sll  $s0, $s0, 3
```

- Standard pipelining will not work in this architecture, there are two consecutive memory accesses which are the problem, all other instructions can already be calculated in a single cycle.
- Using a better technology would increase the speed. However, this is not one of the easiest solutions. The first two solutions are clearly better suited.

7. (6 points) A friend suggests adding a cache system to speed up the execution of MIPS assembly code on the micro-architecture from question 6 (slow load word (`lw`) instruction that requires 10 clock cycles for memory access).

   Which of the two cache design options would make the program in question 5 run faster? Briefly explain why.

   Solution A A direct-mapped cache with a capacity of 8 words and an access time of $t_{cache} = 1$ *clock cycle*

   Solution B A two-way set associative cache with the same capacity of 8 words, but with twice the access time $t_{cache} = 2$ *clock cycles*.

   Note: The question is specific to the program in question 5 and not about speeding up any arbitrary program.

Solution:

In a direct mapped cache, there is only one location where data can be placed in a cache. In a direct mapped cache with 8 locations, the address `0x0000 0004` and `0x0000 0024` would map to the same cache location. If this cache is used, every memory access would result in a cache miss. Using solution A would make the program run even slower, as every memory access would have first a cache miss (1 clock cycle) followed by the actual memory access (10 cycles).
$N_{cycles} = 2 + (8 \times (5 + 2 \times 11)) = 218$

Solution A can not be used to speed up the program.

In a 2-way set associative memory, there are two locations where data can be placed in a cache. Although the addresses `0x0000 0004` and `0x0000 0024` still map to the same set, they can both be placed in the cache. The first two memory accesses would be compulsory misses requiring 12 clock cyles (2 for the cache miss plus 10 for the actual memory access). But for the following 7 iterations, every memory access would be a cache hit, requiring only 2 clock cycles.
$N_{cycles} = 2 + (7 \times (5 + 2 \times 2)) + (5 + 2 \times 12) = 94$

Solution B is the only viable alternative, that would speed up the program by more than 2x.

Note that the program used is virtually identical to the one used in class notes to explain the problems with Direct-mapped cache.