Name: ______________________

First Name: ______________________

Student ID: ______________________

2nd session examination

Design of Digital Circuits SS2012

(252-0014-00S)

Srdjan Čapkun, Frank K. Gürkaynak

Examination Rules:

1. Written 90 minutes.
2. No books, no calculators, no computers or communication devices. Five pages of handwritten notes are allowed.
3. Write your answers on this document, space is reserved for your answers after the questions. Blank pages are avaliable 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 assitant.
6. Answers will only be evaluated if they are readable
7. Write with black or blue in (no pencil, no green or red color).
8. Show your work. For some questions, you may get partial credit even if the end result is wrong due to a calculation mistake.

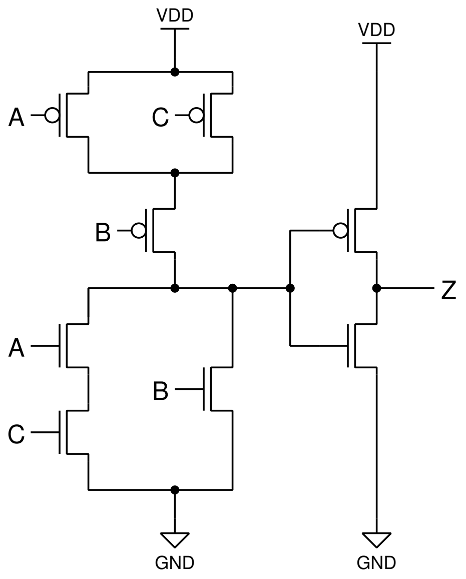
Question:	1	2	3	4	5	6	Total
Points:	5	10	20	15	15	10	75
Score:							

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

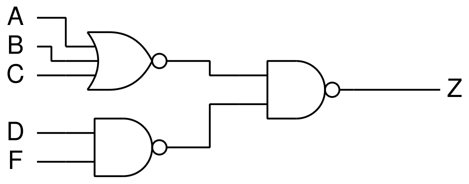
$(-5)_{10}$ using six-bit sign magnitude:	$(10\,0101)_2$
$(38)_{10}$ using six-bit unsigned:	$(10\,0110)_2$
$(-28)_{10}$ using six-bit two's complement:	$(10\,0100)_2$
Hexadecimal $(2C)_{16}$ using six-bit unsigned:	$(10\,1100)_2$

(b) (2 points) Consider the transistor level schematic below. What is the output going to be when:

A=1, B=0, C=1	**Z = 1**
A=0, B=1, C=1	**Z = 1**



(c) (1 point) Find a simplified Boolean function realized by the following circuit.
(*Hint: use bubble pushing to simplify the circuit*)



Solution: Z = (A + B + C) + (DF)

2. A circuit has four inputs and two outputs. The inputs $A_{3:0}$ represent a number from 0 to 15. Output P should be TRUE if the number is prime (0 and 1 are not prime, but 2,3,5 and so on are prime). Output D should be TRUE if the number is divisible by 3. (*hint: 0 is not divisible by 3*).

   (a) (2 points) Complete the following truth table

A_3	A_2	A_1	A_0	P	D
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	1	0	1

   (b) (4 points) Write Sums of Products (SOP) representation for P and D. No simplification required here.

Solution:
$P = A_3\,A_2\,A_1\,A_0 + A_3\,A_2\,A_1\,A_0 + A_3\,A_2\,A_1\,A_0 +$
$A_3\,A_2\,A_1\,A_0 + A_3\,A_2\,A_1\,A_0 + A_3\,A_2\,A_1\,A_0$
$D = \overline{A_3}\,\overline{A_2}\,A_1\,A_0 + \overline{A_3}\,A_2\,A_1\,\overline{A_0} + A_3\,\overline{A_2}\,\overline{A_1}\,A_0 + A_3\,A_2\,\overline{A_1}\,\overline{A_0} + A_3\,A_2\,A_1\,A_0$

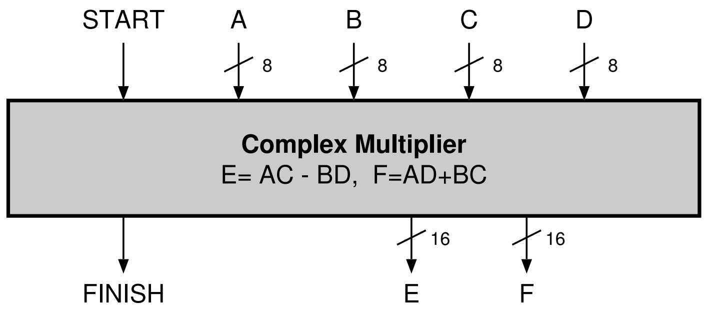
   (c) (4 points) Write simplified equations for both P and D.
   (*Hint: you can use Karnaugh maps to simplify equations*)

Solution:
$P = \overline{A_3}\,A_2\,A_0 + \overline{A_3}\,A_1\,A_0 + \overline{A_3}\,\overline{A_2}\,A_1 + \overline{A_2}\,A_1\,A_0$ **or**
$P = \overline{A_3}\,A_1\,A_0 + \overline{A_3}\,\overline{A_2}\,A_1 + \overline{A_2}\,A_1\,A_0 + A_2\,\overline{A_1}\,A_0$
D can not be simplified further!
$D = \overline{A_3}\,\overline{A_2}\,A_1\,A_0 + \overline{A_3}\,A_2\,A_1\,\overline{A_0} + A_3\,\overline{A_2}\,\overline{A_1}\,A_0 + A_3\,A_2\,\overline{A_1}\,\overline{A_0} + A_3\,A_2\,A_1\,A_0$

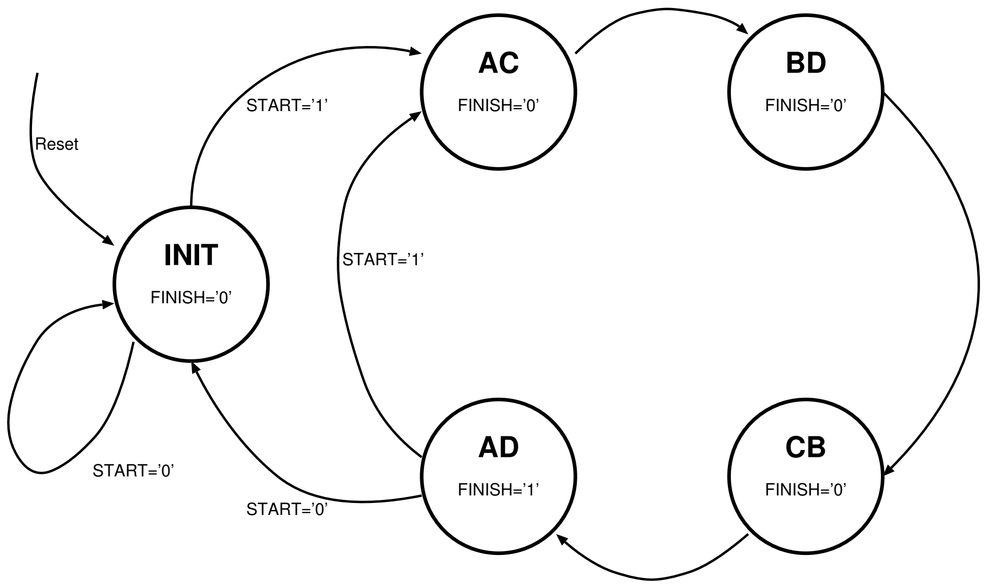
3. In this question we will investigate a circuit that can calculate the multiplication of two complex numbers expressed in the form $(a + bi)$ where a and b are integers expressed in 8-bit two's complement form and i is the complex number which satisfies the equation $i^2 = -1$. As you may remember the multiplication of two complex numbers is

$$\begin{aligned}(e + fi) &= (a + bi) \times (c + di) \\ &= (ac - bd) + (ad + bc)i\end{aligned}$$

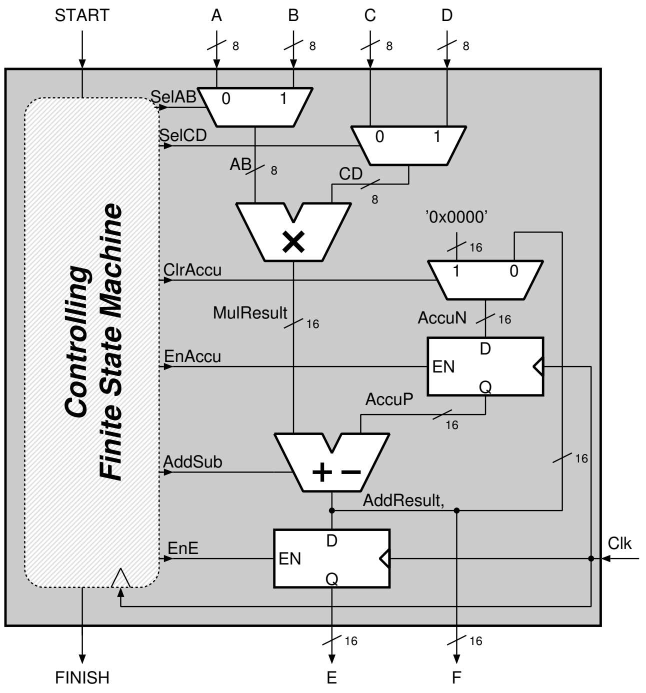
The circuit that we will build has four separate 8-bit inputs (A, B, C, D) for the two complex numbers. A START signal is used to tell the circuit that a new pair of complex numbers are ready to be processed. The circuit has two 16-bit outputs (E, F) for the complex result, and a FINISH signal that tells that the circuit has result has been calculated and the result is ready at outputs E and F.



A colleague of yours has designed an architecture that performs the complex multiplication operation serially using only one multiplier and an adder subtractor. The circuit is controlled by a finite state machine. The following is the state diagram of the state machine. The state names correspond to the operators used.



The circuit block diagram is also given. It can be seen that the FSM controls a series of internal signals. We are interested in determining the correct value of these signals for each state.



(a) (3 points) In your own words describe how the circuit works. For each of the calculation states (AC, BD, CB, AD), explain which operations take place:

When the `Reset` *signal is active the circuit moves to the* `INIT` *state. The* `ClrAccu` *signal is '0' clearing the accumulator while in this state. The circuit stays in this state as long as* `START` *signal remains '0'...*

Solution:

As soons as `START` is active, we first move to state `AC`. In this state, input `A` and `C` are selected and multiplied. The result goes to the accumulator. In the next state `BD`, this time inputs `B` and `D` are selected. This is subtracted from the accumulator, and the result is written to output register`E`. At the same time the accumulator is cleared. In the cycle `CB`, this time `C` and `B` are selected, multiplied and added to the accumulator. In the last cycle `AD`, inputs `A` and `D` are selected, multiplied and added to the accumulator. The output `FINISH` is set to 1. If the signal `START` is 1, the accumulator is cleared and the system moves to `AC`. Otherwise, the system moves to `INIT` state.

(b) (7 points) Complete the following table, so that we can determine how to design the FSM.

Signal	**Value**	**Description**
`SelAB`	1	Select input B
`SelCD`	1	Select input D
`ClrAccu`	1	Assign the value '0' to the accumulator register input
`EnAccu`	1	Enable the accumulator register
`AddSub`	1	Perform Addition
`EnE`	1	Enable the register for output E

State	`START`	**NextState**	`SelAB`	`SelCD`	`ClrAccu`	`EnAccu`	`AddSub`	`EnE`	`FINISH`
`INIT`	0	`INIT`	X	X	1	1	X	0	0
`INIT`	1	`AC`	X	X	1	1	X	0	0
`AC`	X	`BD`	0	0	0	1	1	0	0
`BD`	X	`CB`	1	1	1	1	0	1	0
`CB`	X	`AD`	1	0	0	1	1	0	0
`AD`	0	`INIT`	0	1	1	1	1	0	1
`AD`	1	`AC`	0	1	1	1	1	0	1

page intentionally blank

(c) (10 points) The following is the Verilog code that should realize the complex multiplier. In this circuit the FSM is instantiated. The code is divided into 4 parts. Each part contains at most 2 errors. For each part indicate how many errors there are, and re-write the statements with errors so that they are correct.
(*Hint: there are not more than 4 errors in total*)

```
module complex (A,B,C,D,E,F, START, FINISH, Clk);
 input       [7:0] A,B,C,D;
 output     [15:0] F;
 input             START, Clk;
 output            FINISH;

 wire          SelAB, SelCD, EnE;
 wire          ClrAccu, EnAccu, AddSub;
 wire [7:0]    AB, CD;
 wire [15:0]   AccuN;
 reg  [15:0]   MulResult, AddResult, AccuP;
```

Solution: There is one problem in this section.
The signal E is not defined. It should be defined as `output reg [15:0] E;` since its value is assigned in a process.

```
 // instantiate FSM, no errors in this statement
 FSM myFSM (.Clk(Clk), .START(START), .FINISH(FINISH),
            .SelAB(SelAB), .SelCD(SelCD), .EnE(EnE),
            .EnAccu(EnAccu), .ClrAccu(ClrAccu),
            .AddSub(AddSub));

 assign AB    = SelAB ? B : A; // 1:B
 assign CD    = SelCD ? D : C; // 1:D
 assign AccuN = ClrAccu ? 16'b0 :AddResult; // 1:Clr
```

Solution: There are no mistakes in this section.

```
21  always @ (AB,CD)
22     MulResult <= AB * CD;
23
24  always @ (MulResult, AccuP)
25     if (AddSub)  AddResult <= MulResult+AccuP; //1:Add
26     else         AddResult <= MulResult-AccuP; //0:Sub
```

Solution: There are two problems in this section:
In line 24, the sensitivity list for the second process does not have the signal `AddSub`
In line 26, the subtraction should be the other way around:
`AccuP - MulResult`.

```
27  always @ (posedge Clk)
28    if (EnE) E <= AddResult;
29
30  always @ (posedge Clk)
31    if (EnAccu) AccuN <= AccuP;
32
33  assign F = AddResult;
34
35 endmodule
```

Solution: There is one problem in this section:
In line 31, the Register definition should be `AccuP<=AccuN`

4. In this question we will continue with the circuit from question 3. You will first be asked to estimate the area and speed of the circuit. Then in the second part you will be asked to design a parallel implementation and report the size of the circuit.

   Use the following table to estimate the speed and area of the circuit. Consider that the FSM **is very small** and **is not part of the critical path** for this exercise.
   Note: 1 ns = 10^{-9} s and 1 μm^2 = 10^{-12} m^2

Description	**Bit-width**	**Area** [µm²]	**Critical Path** [ns]
Adder	8-bit	300	1.0
Subtracter	8-bit	300	1.0
Adder/Subtracter	8-bit	350	1.1
Multiplier	8-bit	2,000	2.5
Multiplexer	8-bit	100	0.2
Register(with enable)	8-bit	200	0.0 (ideal)
Adder	16-bit	600	2.0
Subtracter	16-bit	600	2.0
Adder/Subtracter	16-bit	700	2.1
Multiplier	16-bit	8,000	5.0
Multiplexer	16-bit	200	0.2
Register(with enable)	16-bit	400	0.0 (ideal)

   (a) (2 points) Calculate the size of the circuit in Question 3.

   Solution:

$$\begin{aligned} A_{old} = & \quad 2 \times A_{mux\,8b} + A_{mul\,8b} + A_{mux\,16b} + A_{addsub\,16b} + 2 \times A_{reg\,16b} \\ = & \quad 2 \times 100 + 2,100 + 200 + 700 + 2 \times 400 \\ = & \quad 4'000 \end{aligned}$$

   (b) (2 points) Calculate the critical path of the circuit in Question 3.

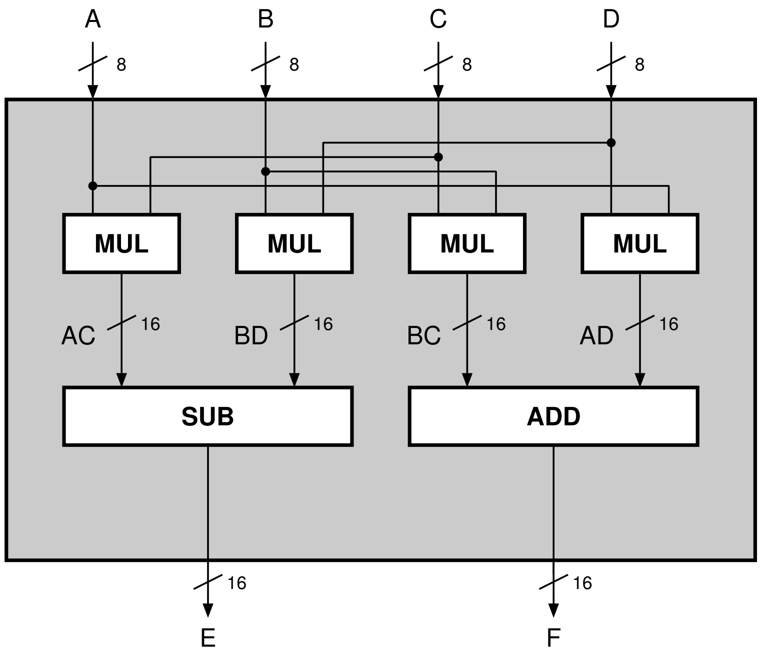
   Solution:

$$\begin{aligned} t_{crit\,old} = & \quad t_{mux\,8b} + t_{mul\,8b} + t_{addsub\,16b} + t_{mux\,16b} + t_{reg\,16b} \\ = & \quad 0.2\,ns + 2.5\,ns + 2.1\,ns + 0.2\,ns + 0\,ns(setup) \\ = & \quad 5.0\,ns \end{aligned}$$

   (c) (2 points) How long (in ns) will it take this circuit to complete one complex multiplication? *Note: it will take more than one clock cycle*

Solution: There are four cycles needed to complete the operation. Each cycle takes $t_{crit\,old} = 5\,ns$ this means that a complex multiplication can be completed every $t_{old} = 20\,ns$.

(d) (5 points) It turns out that the circuit in Question 3, is too slow for your application. Draw the block diagram for an architecture that calculates the complex multiplication combinationally (i.e. within one clock cycle).



(e) (2 points) How much larger is this new circuit when compared to the previous circuit from Question 3?

Solution:

$$\begin{aligned} A_{new} &= 4 \times A_{mult\,8bit} + A_{adder\,16bit} + A_{subtracter\,16bit} \\ &= 4 \times 2,100 + 600 + 600 \\ &= 9'600 \end{aligned}$$

The circuit is $A_{new}/A_{old} = 9,600/4,000 = 2.4$ times larger.

(f) (2 points) How much faster does this new circuit compute a complex multiplication when compared to the previous circuit from Question 3?

Solution:

$$\begin{aligned} t_{new} &= t_{mult\,8bit} + t_{adder\,16bit} \\ &= 2.5\,ns + 2.0\,ns \\ &= 4.5\,ns \end{aligned}$$

The circuit is $t_{old}/t_{new} = 20/4.5 = 4.4$ times faster.

5. In this section, you will be given a task and two code snippets in MIPS assembly language. You will have to decide which of the code snippets can be used for the task. **For all the questions assume the following initial values**:

Registers:

Register	Value
`$s0`	`0x0000 00FF`
`$s1`	`0x0000 0004`
`$s2`	`0x0000 0008`
`$s3`	`0x0000 000C`

Memory:

Address	Value
`0x0000 00000`	`0x0000 FF00`
`0x0000 00004`	`0x0000 00FF`
`0x0000 00008`	`0xFFFF FFF7`
`0x0000 0000C`	`0x1234 5678`

(a) (3 points) Set the content of the register `$t1` to `0x0000 1234`

(A)

```
   lw  $t1, 0xC($0)
   srl $t1, $t1, 16
```

(B)

```
   xor $t1, $t1, $t1
   ori $t1, 0x1234
```

☐ none ☐ A ☐ B [✓ **Both A and B**]

(b) (3 points) Starting from the address `0x0000 4000` write all zeroes to 1024 consecutive memory locations (until `0x0000 5000`)

(A)

```
      addi $s0, $s0, 0x1000
LOOP: sw   $0,  0x4000($s0)
      addi $s0, $s0, -1
      bne  $s0, $0,  LOOP
```

(B)

```
      addi $s0, $s0, 0x4000
      addi $s1, $s0, 0x1000
      addi $s2, $0,  1
LOOP: sw   $0,  $s0
      sub  $s1, $s1, $s2
      bne  $s0, $s1, LOOP
```

☐ none [✓ **A**] ☐ B ☐ Both A and B *B is incorrect since the assignment is on $s0 which constant at 0x4000. If that line were to read:*

```
LOOP: sw $0, $s1
```

it would be correct.

(c) (3 points) Add all the numbers from 0 to 255

(A)

```
      lw    $s1, $s0
      xor   $s0, $s0, $s0
LOOP: add   $s0, $s0, $s1
      addi  $s1, $s1, -1
      bne   $s1, $0, LOOP
```

(B)

```
      addi $s1, $0, 255
      lw   $s0, $0
LOOP: addi $s1, $s1, -1
      beq  $s1, $0, DONE
      jmp  LOOP
DONE:
```

☐ none [✓ **A**] ☐ B ☐ Both A and B

B is incorrect since there is no addition of numbers anywhere. The loop is correct though

---

(d) (3 points) Jump to subroutine STOP if only the 4th bit from the right (representing 2^3) of the data written at address `0x0000 0020` is 1. Otherwise continue with the program at CONT.

(A)

```
      lw   $s0, 0x20($0)
      srl  $s0, $s0, 3
      addi $s1, $0, 1
      beq  $s0, $s1, CONT
      jmp  STOP
CONT: ...
STOP: ...
```

(B)

```
      addi $s0, $0, 0x20
      lw   $s1, $s0
      lw   $s2, 0x8($0)
      and  $s3, $s1, $s2
      bne  $s3, $0, CONT
      jal  STOP
CONT: ...
STOP: ...
```

☐ none ☐ A [✓ **B**] ☐ Both A and B

A looks ok, but it would also work if other bits (higher than 4) are one as well, the jump is not to a subroutine, and the condition is inverse

---

(e) (3 points) Save the two registers `$s0` and `$s1` to the stack

(A)

```
      jal   SAVE
      ...
SAVE: sw    $sp, $s0
      sw    $sp, $s1
      jr    $ra
```

(B)

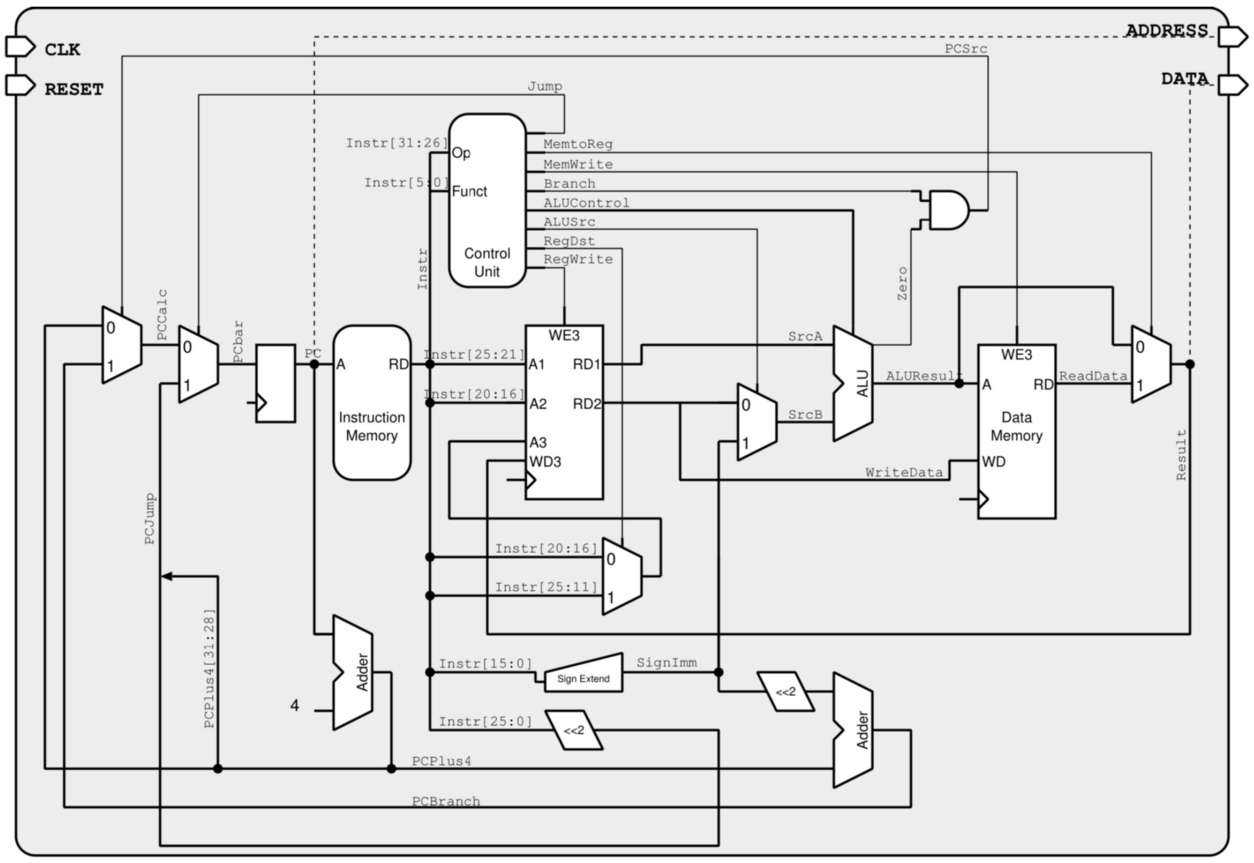
```
   addi $sp, $sp, -8
   sw   $s0, 8($sp)
   sw   $s1, 4($sp)
```

☐ none ☐ A [✓ **B**] ☐ Both A and B

A is incorrect since both values overwrite the last value in the stack.

---

6. The figure below is the block diagram of the MIPS process that you have designed during the laboratory exercises.



It turns out that this processor is **not fast enough** for a project that you want to do. You are in a meeting to discuss what can be done to improve the performance of this processor. Your colleagues have made the following suggestions. For each suggestion, first state whether or not the idea will work, and then *briefly* explain why.

(a) (2½ points) Alice: "Let us add a cache to the system. This will reduce the average memory access time, and make the processor faster, as it will spend less time to access memory":

Solution: This idea **does not work**. While the statement is true in general, the processor that is used in the exercise already has 1 clock cyle access time. This can not be reduced further. No matter what caching system is used, the memory access will still be the same.

(b) (2½ points) Bob: ”We can use a pipelined architecture. The pipeline stages will reduce the critical path and allow a higher clock rate to be used”

Solution: This idea **could work**. One of the limiting factors in the processor speed is the critical path. By inserting pipeline registers the critical path can be shortened, and the processor can be operated at a higher frequency. Provided that hazards associated with a pipelined architecture can be handled, this idea would work.

(c) (2½ points) Charlie: ”The processor in the exercise uses a single cycle. This means that it uses 1 CPI (cycles per instruction). I propose using a five-cycle architecture. In this case the processor will use 5 CPI, and will be faster.”

Solution: This idea **does not work**. The speed of the processor depends on the clock cycle time. If the cycle time is the same than a processor that uses 5 CPI is 5 times slower than a processor that uses 1 CPI. A multi-cycle processor can only be faster if the reduction in the clock cycle time is more than the increase in the average CPI. In general this is not possible. Multi-cycle processors are mostly used to share expensive resources (memories, adders).

(d) (2½ points) Diane: ”Since we are not limited by area or power, how about using a multi-processor system using 2 processors in parallel rather than a single processor?”

Solution: This idea **could work**. In theory 2 processors would be twoce as fast as a single processor. However, there are not many problems that have a fully parallelized solution where two proecssors can work independently. Data dependency may reduce the efficiency of parallelization, but in general the idea would increase the performance.