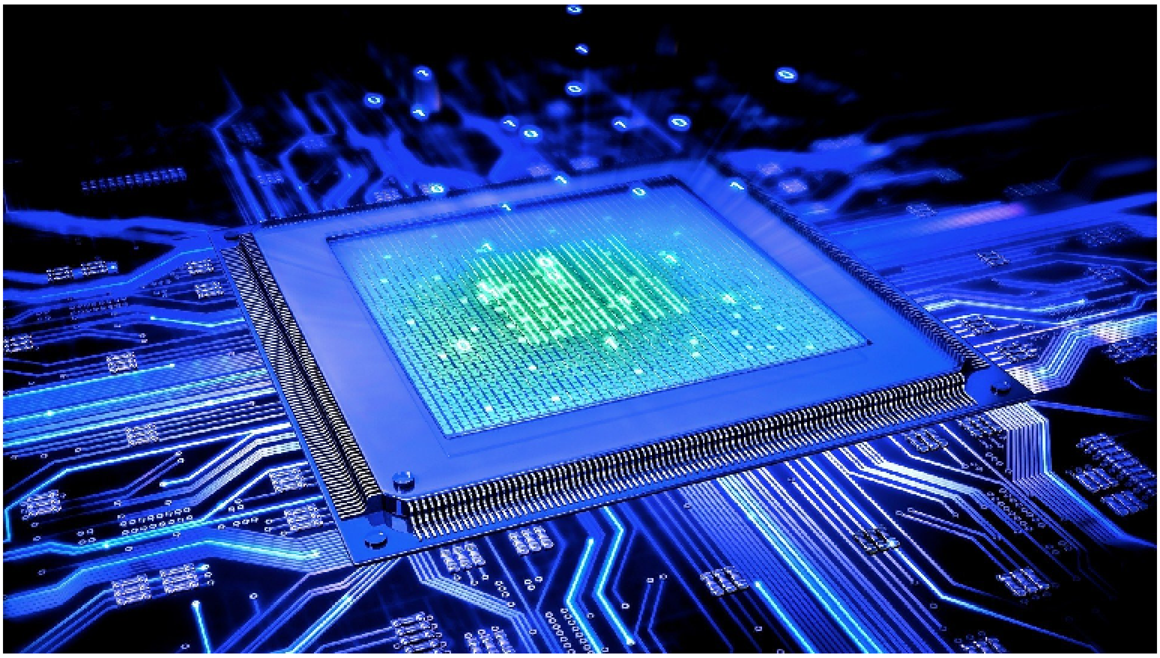
MIDDLE EAST TECHNICAL UNIVERSITY

METU EE445 Computer Architecture I

HW1 – Designing a Basic Computer with Verilog
Deadline: 10.12.23 @23.59



You will submit your homework via **ODTUCLASS** as a **.zip** file containing the relevant source files and your PDF report. Name your file as "HW1_studentid.zip".

Using code directly taken from any resource except those provided in ODTUCLASS, is prohibited. You can verbally discuss the homework with your friends, but you should not exchange codes with each other or write your codes together. **Cheating and plagiarism will result in zero grades, whereas disciplinary actions may also be taken. Late submissions are not allowed.**

In this homework, you will design Basic Computer I (as described in lecture notes) using Verilog. Then, use cocotb to simulate your designs. Finally, you will write a very short report detailing your simulation and results, and you can also briefly discuss the key points of your design. The report should not contain any code.

1 Operations to Be Implemented

Operations to be implemented are shown in Figure 1. That is all the data-processing and memory operations with indirect/direct interrupts. No input/output operations are required to be implemented. The details about these instructions are given in the basic computer lecture notes. **You should use the machine code translations used in the lecture notes which are shown in the figure below. That is 0XXX for AND, 7400 for CLE, etc. Do not design your own.**

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
~~INP~~	~~F800~~		~~Input character to AC~~
~~OUT~~	~~F400~~		~~Output character from AC~~
~~SKI~~	~~F200~~		~~Skip on input flag~~
~~SKO~~	~~F100~~		~~Skip on output flag~~
ION	F080		Interrupt on
IOF	F040		Interrupt off

Figure 1: List of Operations to Be Implemented
Implementations of INP, OUT, SKI, and SKO operations are not required.

2 Module Design with Verilog HDL (20% Credits)

To construct the datapath, several modules are needed. In this section, each of them will be described.

2.1 Multiplexer

Implement a W-bit 8 to 1 multiplexer to be used as a bus in your datapath, where W is a parameter specifying the data width of the input.

2.2 Arithmetic Logic Unit (ALU)

Implement a W-bit ALU, where W is a parameter specifying the data width of its inputs. The ALU will have 2 W-bit data inputs named AC and DR, a 1-bit input named E, a 3-bit operation select input, and a W-bit result output. The ALU should also have four other status output bits: Carry-out (CO), overflow (OVF), negative (N), and zero (Z). E input will be connected to a register that stores the CO. Negative and zero bits are affected by all the ALU operations, whereas carry-out and overflow can only be affected by arithmetic operations. The ALU operations to be implemented are given in Table 1, and the status descriptions are given in Table 2.

Table 1: ALU Operation Control

ALU Operation	**Output**
Addition	$AC + DR$
AND	$AC \wedge DR$
Transfer	DR
Complement	AC'
Shift Right	$\{E, shr[AC]\}$
Shift Left	$\{shl[AC], E\}$

Table 2: ALU Status Descriptions

Status	**Description**
CO	1 if there is a Carry-out from add or subtract operations; 0 for logic operations
OVF	1 if the add or subtract operation results in overflow; 0 for logic operations
Z	1 if the result is zero
N	1 if the result is negative

2.3 Register with Load, Reset, and Increment

Implement a positive edge-triggered register with parallel load, write enable, synchronous reset, and increment. The specifications of the register are provided in Table 3. All the operations should take place in the positive clock edge, and DATA is always the parallel load input.

Table 3: Register with Synchronous Reset, Write Enable, and Increment

Reset	Write Enable	Increment	Operation
0	0	0	$A \leftarrow A$ (Retain)
0	0	1	$A \leftarrow A + 1$
0	1	X	$A \leftarrow DATA$
1	X	X	$A \leftarrow 0$

2.4 Memory Unit

As a memory unit, you will implement a **word-addressable** memory. The module has the inputs of clock, write enable, 16-bit write data, 12-bit input address, and the 16-bit output of read data. Memory should be 4096x16.

Memory addressing should be combinational. Read data output should change the moment the input address changes. Memory writes should be sequential. That is, data should be written in the positive clock-edge.

3 Datapath Design (10% Credit)

Datapath should be the same as the one shown in Figure 2 with the addition of the IEN register for the interrupt enable/disable functionality.

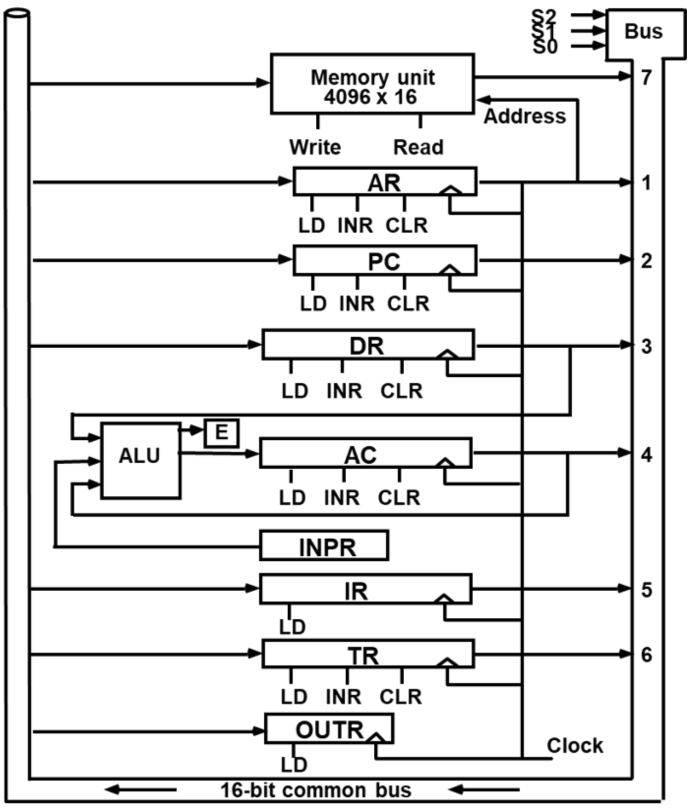


Figure 2: Datapath Illustration

4 Controller Design (30 % Credit)

The controller will be designed to give the datapath correct signals for executing the instructions in Figure 1. **The controller will be written without any restrictions**. This means it can be a single standalone .v file or instantiate other modules like the datapath and may use whatever description model, etc.

For the interrupts, you can assume the FGI input of the Basic Computer is asserted correctly. You do not need to store it as I/O is not implemented.

5 Putting The Computer Together (10 % Credit)

The computer is put together by connecting the datapath with the controller. This will be done in another .v file by instantiating the datapath and the controller and then connecting them with wire variables. This file should be your top-level design file.

Your computer should have PC, AR, IR, AC, and DR registers as its outputs and clk, FGI as its input. A Verilog file (BC_I.v) is given with the homework; you should use that file as your top-level module. This is important as your design will be put through a test bench as part of the grading.

6 Simulation (30% Credit)

Write a sample test program in assembly language and translate it into machine code. Use **at least ten different instructions**.

Prepare a testbench using cocotb to check for the correct operation of your implementation using the fully connected computer. Load your sample test code to the memory of the basic computer you have implemented, and with the help of your testbench, compare register/memory contents with the expected results.

One thing to note here is that the **test bench should be fully automated**. Just printing the results and checking them by hand is unacceptable. You can use prints for your debug purposes, but the final test bench should tell if there is an error in the design or not and if there is an error where it has happened.

7 Deliverables

You can submit your project folder as long as it contains the elements below.

1. All the Verilog codes necessary (reg, mem, mux, ALU, datapath, controller, BC_I.v,...).
2. cocotb test bench (.py file, makefile).

3. A short report in PDF format with the assembly program you used for the simulation and the simulation results. You can also briefly discuss the key points of your design.