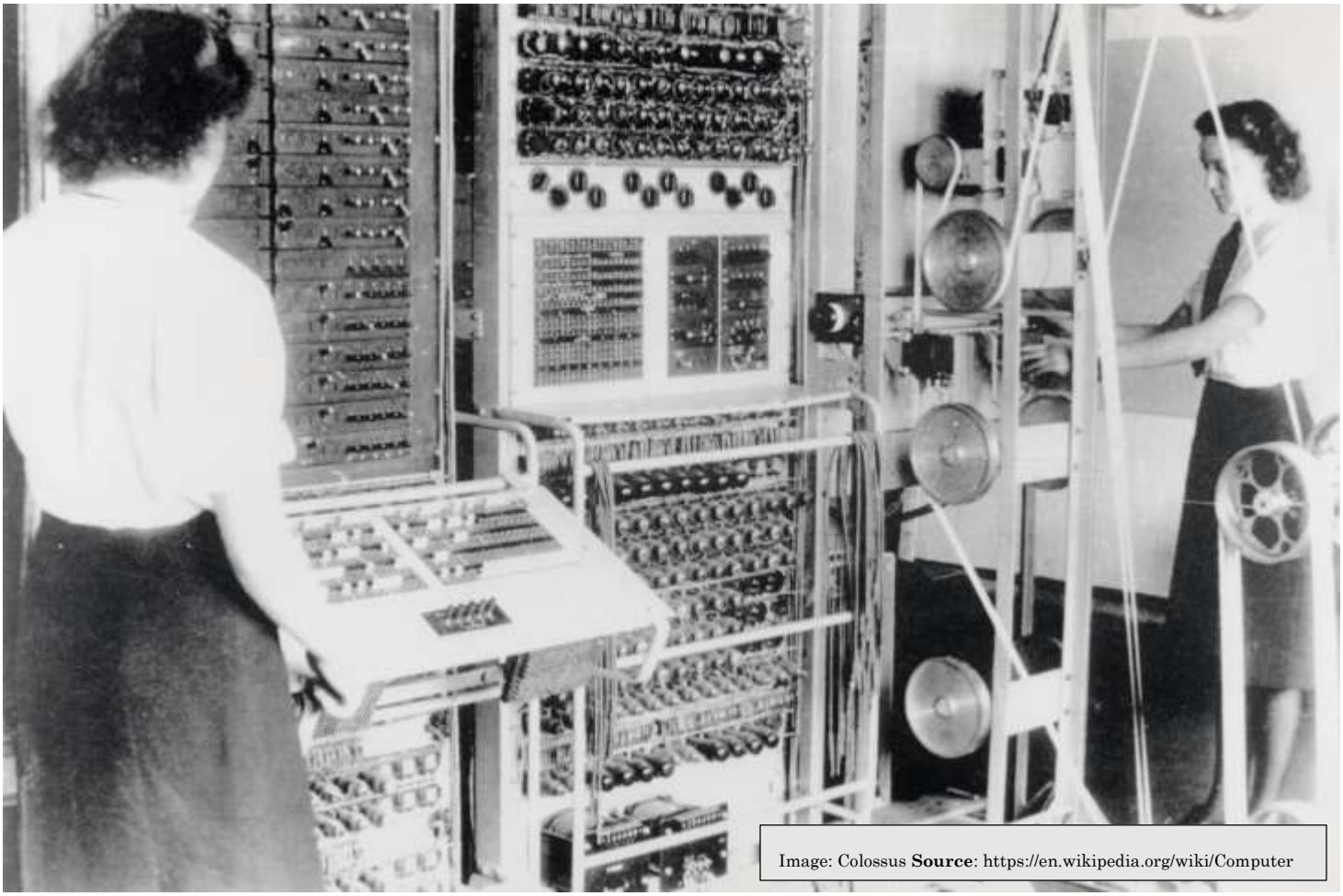


Image: Colossus **Source**: https://en.wikipedia.org/wiki/Computer

DEU Electronic Universal Automatic Reduced Computer (DEUARC) Design

CME 2206 Computer Architecture 2022-2023 Term Project

CME 2206 – LAB PROJECT

DESCRIPTION

You will design a basic computer that called DEUARC (**D**EU **E**lectronic **Uni**versal **A**utomatic **R**educed **C**omputer). DEUARC has nine registers, three memories, arithmetic and logic unit, control unit and bus system.

Quartus II software will be used to design and verify DEUARC. The project is given as a term project and will be implemented in weekly lab sessions. It is advised that you read problem definitions of all of them before actually starting to implement your design, i.e., Common Bus and Registers.

Please submit zipped All Files (don't forget to submit waveform of the results) of the simulations for each lab session.

GENERAL STRUCTURE OF DEUARC

REGISTERS

DEUARC has 9 registers which are *Address Register, Program Counter, Stack Pointer, Input Register, Output Register, Instruction Register* and *3* general purpose registers.

MEMORIES

In DEUARC, there is three memories, which are ***instruction (32x11), data (16x4) and stack (16x5).*** Each has "read enable" signals and "data inputs". *Data and stack memory* also have "write enable input".

BUS SYSTEM

Common bus system will be responsible for data flow and provide data transfer between register and/or memories.

ARITHMETIC AND LOGIC UNIT

In ALU, arithmetic and logical operations will be held.

CONTROL UNIT

Control unit processes instructions to direct the micro-operations for computer's memory, registers and arithmetic/logic unit. Control unit consists of decoders and a number of control logic gates. It should produce operation signals and time periods for fetching, decoding and executing the instructions.

ASSIGNMENT 2 – COMMON BUS SYSTEM

You are expected to implement the common bus architecture designed in Quartus II and save it as a block diagram ('symbol file') with the name, "Common_Bus_System_StudentID_GroupID". Then test and simulate your implementation by loading (transferring) data from Instruction memory and Data memory to registers and applying INR and CLR operations on them.

REGISTERS

DEUARC has 9 registers which are *Address Register, Program Counter, Stack Pointer, Input Register, Output Register, Instruction Register* and *3* general purpose registers.

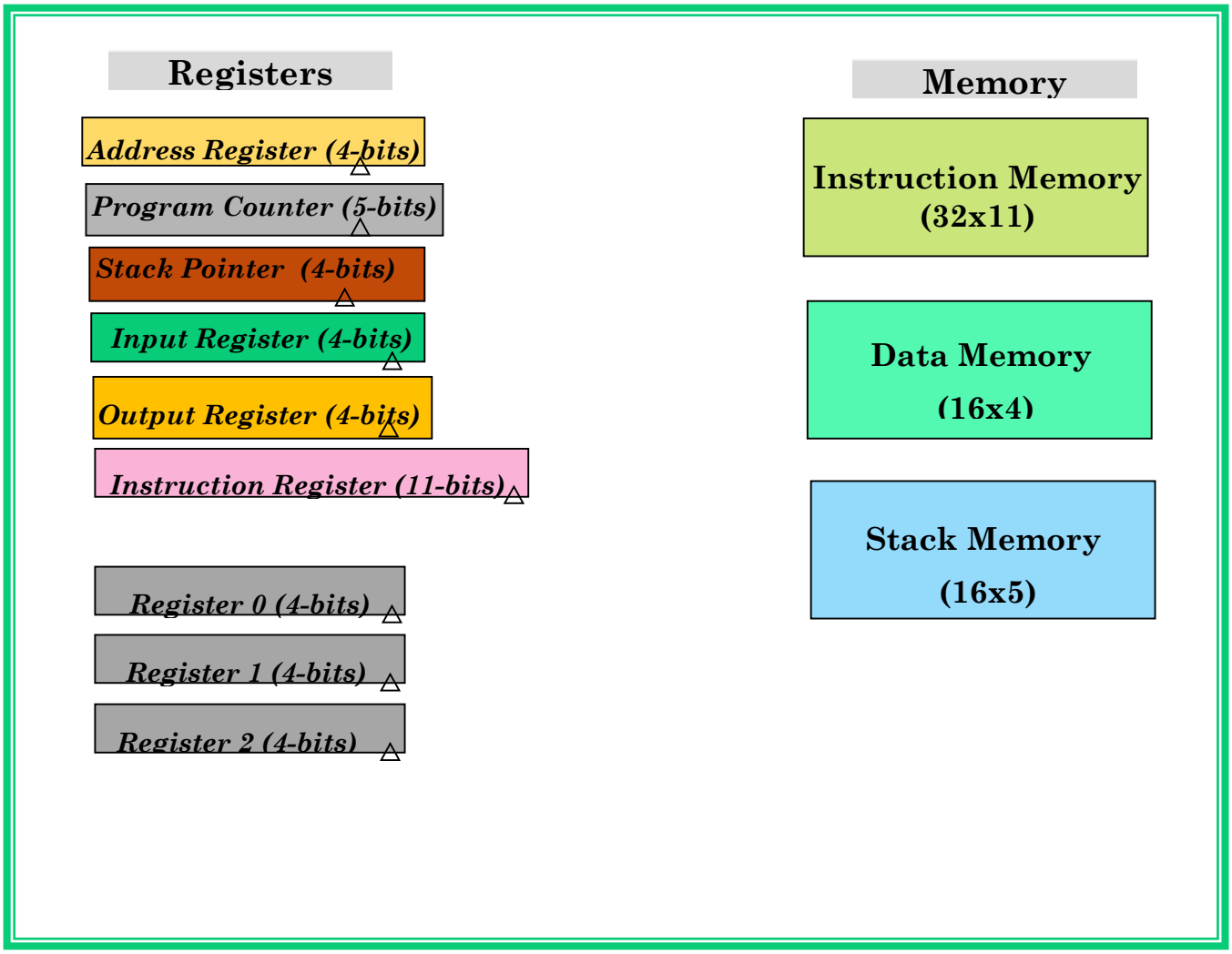
Register Symbol	***Register name***	***Number of bits***	***Function***
AR	Address Register	4	Holds address for data memory. It can be cleared with related control signal.
PC	Program Counter	5	Holds address of instruction. It can be cleared and increase by one with related control signals.
IR	Instruction Register	11	Holds instruction code
SP	Stack Pointer	4	Holds address od stack memory. It can be cleared, increased and decreased by using related control signals.
InpR	Input Register	4	Holds input data
OutR	Output Register	4	Holds output data
R0, R1, R2	General Purposed Registers	4	Holds operands and other data

Table 1 - List of Registers for DEUARC

MEMORY

In DEUARC there is three memories which are ***instruction, data and stack*** memories. Each has read enable and data inputs. Data and stack memory have also write enable input.

1. Instruction Memory (32x11)
2. Data Memory (16x4)
3. Stack Memory (16x5)



COMMON BUS SYSTEM VERIFICATION TEST

Write data 1001 to 0011 address of Data Memory (write data in the *hex* or *mif* file-manually)
Write data 00010011000 to 0010 address of Code Memory (write data in the *hex* or *mif* file-manually)

1. Write data 0100 to Input Register manually, i.e. by using Quartus input.
2. Read the data from Input Register and load to R0
3. Read data from 0011 address of Data Memory and write to R2
4. Read data 00010011000 from 0010 address of Code Memory (write data in the *hex* or *mif* file)
5. Read data from R0 and R2 and execute ALU operation
   (opcode: 0010, Rd: 01->R1, S1: 10->R2, S2:00->R0)
6. Read data ALU result and load to R1 (Rd)
7. Read data from R1 and load to Output Register
8. Read data from R1 and load to R2
9. Write data from R2 to 1000 address of Data Memory (write data in the *hex* or *mif* file)
10. Read data from 1000 address of Data Memory