Computer Hardware and Digital Design
Assessed assignment
Part 1

Introduction
The assignment will last for up to eight lab sessions. During the assignment you will design part of the core of a RISC microprocessor, and demonstrate it working. The assessment will be based on a report that you submit in January.

In part 1 of the assignment you will design the two main building blocks that will be used for this assignment: an ALU and a memory to hold data. In units 1.5 and 2.5 you can find basic templates and testbenches for an ALU and a memory that you will need to adapt to your requirements. This part of the assignment should take about 2 weeks to complete.

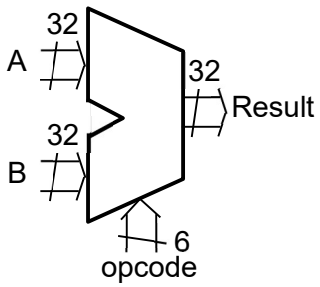
In part 2 of the assignment you will extend your data memory to hold the values of variables that your program is processing. Then you will connect the ALU and memory file together, and run a machine instruction on them. This part of the assignment should take one to two weeks.

In part 3 of the assignment you will resolve problems with the synchronisation of the instructions to ensure that a stream of machine instructions can run correctly. This part of the assignment should take the remainder of the lab sessions.

In order to ensure that your design is individual to you, various aspects of the design will be dictated by your student ID number. These include the machine code used by your design, the contents of the register file, and the purpose of the program that you will use to demonstrate the correctness of your design. This means that every student should have a unique design.

You may work as an individual or as a pair in labs, but each student will have to demonstrate their own individual design.

The ALU
Write a VHDL description of an ALU with two 32-bit inputs, *A* and *B*, and a 32-bit output *Result*.



The result is derived from one or both of the inputs according to the value of a 6-bit opcode. The operations that the ALU can perform are listed below:

- a + b
- a - b
- |a| (i.e. the absolute value of a: this is achieved using the VHDL function *abs(a)*)
- -a
- |b| (i.e. the absolute value of b)
- -b
- a or b
- not a
- not b
- a and b
- a xor b

The opcode that will be used to represent each of these operations is determined by the last digit of your student ID number. The table below shows which opcode you should use in your design for each instruction.

Digit of ID no.	0	1	2	3	4	5	6	7	8	9
a + b	9	13	4	2	14	1	5	4	12	6
a – b	6	15	8	7	15	8	11	8	6	2
|a|	15	11	15	14	8	12	12	11	9	4
-a	5	3	3	9	2	11	14	10	1	12
|b|	12	5	6	4	10	6	3	14	2	5
-b	2	1	11	8	12	9	15	6	15	7
a or b	10	10	5	1	6	15	4	7	4	8
not a	8	8	10	15	3	5	13	9	3	14
not b	11	6	14	3	11	7	8	15	13	11
a and b	13	14	9	5	5	3	7	2	5	10
a xor b	1	2	1	12	4	10	10	3	10	1

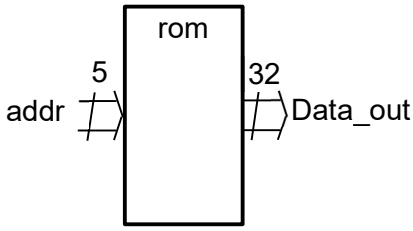
So, for example, if the last digit of your ID number is 2, then a+b is to be represented by opcode 4, a-b is to be represented by opcode 8, |a| is to be represented by opcode 15 and so on. (These are shown as denary values; your design will of course have to use binary or hexadecimal values.)

If the ALU receives an unrecognised opcode then it should output zero.

Create a testbench to show the correct operation of your ALU with a variety of test input conditions.

The data memory

Inside a modern processor there is a very small amount of memory that is used to hold the operands that it is presently working on. This is called the *register file*. In order to introduce the operation of the register file, we will start off by building a small memory whose purpose is to hold the data that a program works on. It will have the following appearance:



The ROM contains 32 data locations that each hold a 32-bit data word. These are addressed by a 5-bit input called **addr**. The data value stored at the location addressed by the **addr** input appears at the output called **data_out**.

The initial data values stored in the memory are determined by the second-from-last digit of your student ID, and are shown in the table below:

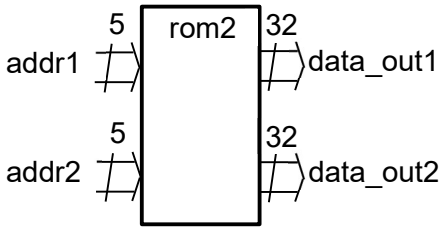
	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	44309	54083	59411	59403	70536	95800	82273	66376	1112	10445
2	48661	74882	72887	67753	42658	100	66352	14802	11921	8252
3	71123	90074	57042	78894	67141	14707	64699	31392	79022	61109
4	46963	94400	38919	86495	25998	18742	58523	94368	35790	41697
5	94184	83579	15468	29463	86650	96349	94706	79082	41880	79800
6	54348	78191	48735	54652	64211	51659	87371	82790	7070	73179
7	98773	77290	5806	93736	56067	36605	35167	74916	76467	1752
8	90477	65213	91724	11349	40159	810	38420	86971	49839	46597
9	51970	29771	41597	59998	69723	33574	97392	87561	95239	69246
10	31197	56377	75583	84776	28861	75594	12623	76196	25986	44321
11	3228	73787	38453	2623	59537	11911	82381	9854	93093	13869
12	78333	5206	57328	35390	33726	60399	1351	85251	56581	17877
13	14898	93375	22896	44236	23913	4551	26750	2924	80949	2459
14	53013	97813	99493	66551	35711	36851	52205	46027	24668	84870
15	12736	12575	41064	93847	85087	33944	97081	59224	88182	80011
16	10364	99349	25498	56134	22853	68565	55668	76271	66426	75336
17	99994	99096	61834	8672	72191	5949	8333	28610	9963	83796
18	27057	54922	79365	47135	87837	46175	36571	85974	5357	13409
19	58050	89981	22591	91858	5042	68522	75396	88460	83220	16187
20	81551	10671	36484	22818	84884	42903	17914	49938	14104	76179
21	58814	80973	93947	23338	22842	76433	20398	50797	30255	54263
22	97301	20890	13349	12959	77156	9652	32505	80165	19949	52093
23	58994	71269	47734	47357	17363	60342	11732	49100	35010	50132
24	92110	64528	61535	93564	87296	42989	66447	15077	35253	9961
25	14807	32599	10729	83474	45117	71221	49600	25760	70177	23214
26	60520	76796	53298	16695	91034	57200	32011	63506	61917	29891
27	63108	44172	73910	65434	73021	64981	95851	28179	34887	57031
28	57916	27627	96075	15297	56444	61865	42929	61505	40413	48274
29	40675	60874	15828	21729	53900	30023	3541	96568	62391	47551
30	357	21828	4677	60374	78916	20631	75397	32572	95961	1375
31	0	0	0	0	0	0	0	0	0	0

So, for example, if your student ID is 714583 then the second-from-last digit of your ID number is 8; so items 0 should be 0, item 1 should be 1112, item 1 should be 11921, and so on. N.B. these values are in *denary (i.e. base 10)*. You will need to convert them to binary or hexadecimal.

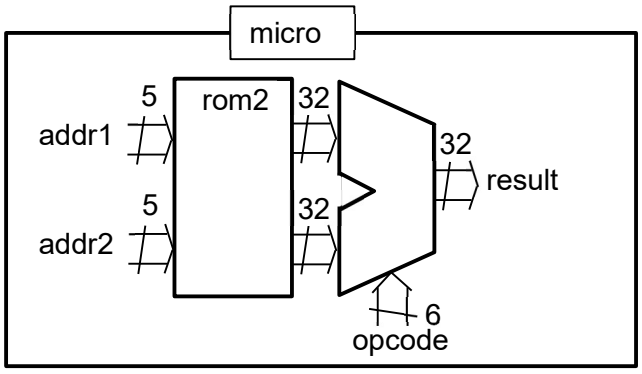
If you need the above data in spreadsheet form, you can download it from the assignment section of the Canvas pages in the spreadsheet *CHDDdata.xls.* You may find the excel function *dec2hex* helpful in doing the conversions.

Create a testbench to show the correct operation of the ROM.

A two-port data memory
Now modify your ROM to create a dual ported ROM. This has two address inputs and two data outputs associated with the one set of data items stored in the ROM. The data value addressed by **addr1** appears at the output **data_out1** and the value addressed by **addr2** appears at the output **data_out2**.



Connecting to the ALU
Now create a new VHDL file called **micro**. Use this to integrate the ALU and the memory together as shown below.



This arrangement can be used to perform arithmetic operations on the data values stored in the memory. For example, to add item 5 and item 6 you would supply the inputs **addr1**=5, **addr2**=6 and use the **opcode** value corresponding to add. The outcome of the computation will appear at the **result** output.

Use simulations to check out that your design works as expected.