Faculty of Engineering

The Hashemite University, Zarqa-
Jordan

CPE443 Project 2020: Design Your Own ISA

Instructor: Dr. Dheya Mustafa

Names	Noor Alali, Rania Maali, Lana abulawi, Mayar Almousa (Group F)
University ID	1732975, 1732935, 1738349, 1732972
Due Date	20/12/2020

□ Objectives:

1. Our goal, in designing an ISA is to make a CPU which has enough functionality so that programmers can write programs for it.

2. Our ISA should be general enough to run all the provided benchmark programs, while being simple enough to allow you to implement a reasonably fast processor in the allotted time frame.

3. We need instructions to do basic math, data comparisons, deal with data of different sizes, and instructions to branch and jump (so we can implement decisions, loops and functions).

4. Our instruction width equals 10 bit, Data and address width (word size) equals 10 bit.

5. Our ISA should support two I/O instructions - in and out. These instructions allow a processor to communicate with the external world.

6. We want to have some registers, so that we don't have to keep going out to slow memory.

7. The data bus equals the natural word size of the ISA.

8. We have to implement enough instructions to provide rich set of operations that the programmer can perform.

9. As a CPU designer we think of **1-**operand instructions.

10. We think about leaving opcode space for expansion in case we leave out an instruction we need (to be defined in the future).

Phase one:

Our ISA format:

Op-code	Op-T	Operand
4-bits	2-bits	4-bits

Op-T	Operand type
00	Register
01	Immediate
10	Memory
11	I/O device

Registers	REG value
R0	0000
R1	0001
R2	0010
R3	0011
R4	0100
R5	0101
R6	0110
R7	0111
R8	1000
R9	1001
R10	1010
R11	1011
R12	1100
R13	1101
R14	1110
R15	1111

Our instructions are **1-**operand instructions, the first operand is always stored into R0 and the second operand is present either in Registers, Immediate value, Memory or form I/O device, the register R0 is the default address thus after data manipulation the results are stored into R0.

The two bits (Op-T) determine the second operand type whether it is a Register, Immediate value, Memory or form I/O device as shown above.

We have 16 registers, so that we don't have to keep going out to slow memory, we assigned 4-bits to the op-code so that we can implements 16 instructions in our ISA to do basic math, Data transfer, I/O instructions, Logical operations, and instructions to jump (so we can implement decisions, loops and functions).

Instruction	Op-code	Format	Operation	Description
Add	0000	Add X Add REG Add IMM	R0 <— R0 + [X] R0<— R0 + REG R0 <— R0 + IMM	Add the operand (Memory, register or immediate) to R0 and store the result in R0
Mul	0001	Mul X Mul REG Mul IMM	R0 <— R0 * [X] R0<— R0 * REG R0 <— R0 * IMM	Multiply the operand (Memory, register or immediate) with R0 and store the result in R0
Div	0010	Div X Div REG Div IMM	R0 <— R0 / [X] R0<— R0 / REG R0 <— R0/ IMM	Divide R0 by the operand (Memory, register or immediate) and store the result in R0
And	0011	And X And REG And IMM	R0<— M[X] . R0 R0 <— REG . R0 R0 <— IMM . R0	Performs the logical And operation between the register R0 and the operand (Memory, register or immediate) and store the result in R0.
Orr	0100	Orr X Orr REG Orr IMM	R0 <— R0 + [X] R0<— R0 + REG R0 <— R0 + IMM	Performs the logical OR operation between the register R0 and the operand (Memory, register or immediate) and store the result in R0
Shr	0101	Shr REG Shr IMM		This instruction performs a logical right shift on the destination register R0 with filling the LSB by zeros with the operands magnitude.
Shl	0110	Shl REG Shl IMM		This instruction shifts the word in R0 left by the operands value times with filling zeros in the MSB.
Eql	0111	Eql X Eql REG Eql IMM	if R0 == M[X] (Or R0 == REG Or R0 == IMM) , R0 = 1, else R0=0	Check if R0 is equal to the operand (Memory, register or immediate) and put 1 in R0 if R0 is equal to the operand.
Les	1000	Les X Les REG Les IMM	if R0 < M[X] (Or R0 < REG Or R0 < IMM) , R0 = 1, else R0=0	Compare the operand (Memory, register or immediate) with R0 and put 1 in R0 if R0 is less than the operand.
Jmp	1001	Jmp label	Jmp destination	The Jmp instruction causes an unconditional transfer of control (unconditional jump) to the label.

Ldd	1010	Ldd X
		Ldd REG
		Ldd IMM
			R0 ← M[X]
			R0 ← REG
			R0 ← IMM
				Load the content of the operand (Memory, register or immediate) into the R0 the destination register.

Stt	1011	Stt X
		Stt REG
			M[X] ← R0
			REG ← R0
				Store the content of the register R0 into the operand (Memory or register).

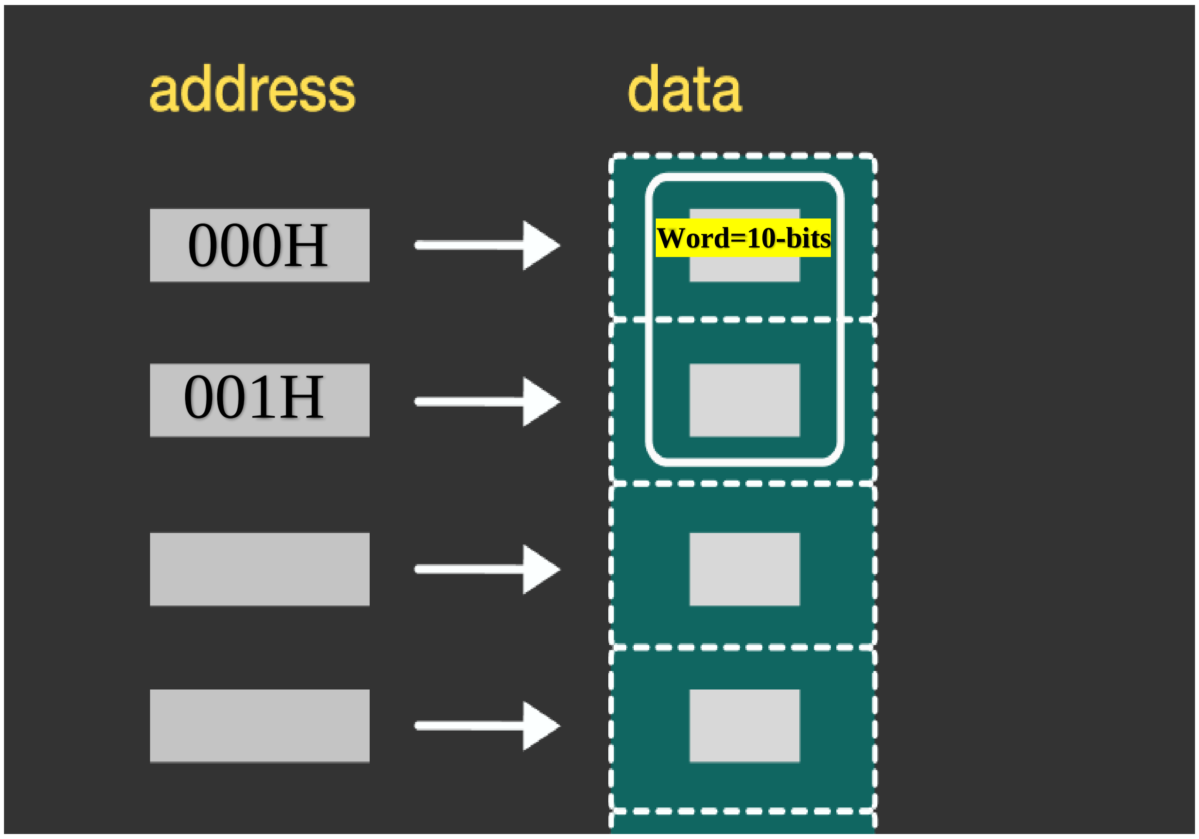
In	1100	In port 0H	R0 ← Port 0H	Input a word from I/O ports 0000H to FFFFH, the destination is R0.
Out	1101	Out port 0H	Port 0H ← R0	Output a word to I/O ports 0000H to FFFFH from R0.
Hlt	1110	Hlt	Halt the machine	It tells the machine to stop.
Tbd	1111	To be defined	To be defined	To be defined (for future use).

Memory Model:

The memory is in big endian format, a big-endian system stores the most significant byte of a word at the smallest memory address and the least significant byte at the largest, and the memory grows downwards.

Our instruction width equals 10 bit, Data and address width (word size) equals 10.

Memory: Our microprocessor can access up to 2^10 * 10 = 1280 byte =1.28 Kbyte of memory, addresses range (000 - 3FF).



□ Phase two: Instruction Set Simulator (ISS)

Our goal in this phase is to Implement an Instruction set simulator for our ISA Design in phase one. The input is a text file that contains an assembly program written in our ISA in phase one.

Our ISS is coded in C++ language, it mimics the behavior of the microprocessor by reading instructions and maintaining internal variables which represent the processor's registers.

We submitted the source code with descriptive comments, a clear readme file on how to run our code and we included three assembly programs to test the code with.

□ Project management’s activities:

We were unable to meet in person due to the Corona virus crisis, instead of that we met online via video and audio calls, we exchanged our ideas, shared useful links and resources and helped each other to reach the best result, each person was responsible for writing part of the code and then we compiled our work into one code.

□ List of bugs:

We have had several errors that we tried to get them fixed, we are going to mention some of them;

The first one that we faced was by the compiler reading the numbers as characters not as integers, we solved it by subtracting 48 from the immediate number’s ASCII value to get the integer value.

Another error got detected that the compiler didn't store what was read in the file, so we got it fixed by letting the complier store it in an array of strings.

Another error that was difficult to solve that is representing the instructions and get values from the input/output devices.