EEDG/CE 6370
Design and Analysis of Reconfigurable Computing Systems
Spring 2022

Design Assignment

In this assignment we will design a simple 16-bit *customizable* microprocessor. The microprocessor can be considered the core for various user specific computing machines. It consists of a set of basic microprocessor features that can be used without any changes for some simple applications, or can be extended by the user in many application specific directions. Extensions can be achieved by adding new instructions or other features to the ***µUTD*** 's core, or by attaching functional blocks to the core without actually changing the core.

Requirements

The basic features of the microprocessor core are:

- 16-bit data bus and 12-bit address bus that enable direct access to up to 4096 16-bit memory locations
- two programmer visible 16-bit working registers, called A and B registers, which are used to store operands and results of data transformations
- memory-mapped input/output for communication with the input and output devices
- basically a load/store microprocessor architecture with a simple instruction cycle consisting of four machine cycles per each instruction;
- all data transformations are performed in working registers
- support of direct and the most basic stack addressing mode, as well as implicit addressing mode
- definable custom instructions and functional blocks which execute custom instructions can be added
- implemented using an FPGA

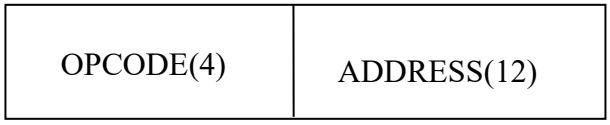
Instruction Formats and Instruction Set

The microprocessor instructions have very simple formats. All instructions are 16-bits long and require one memory word. In the case of direct addressing mode, 12 lower instruction bits represent an address of the memory location. All other instructions for basic data processing, program flow control, and control of processor flags use implied addressing mode. The core instruction set is:

Mnemonic	**Function**
LDA	**A ← M[address]**
LDB	**B ← M[address]**
STA	**M[address] ← A**
STB	**M[Address] ← B**
JMP	**PC ← Address**
JSR	**Stack ← PC, PC ← address, SP ← SP-1**
ADD	**A ← A+B**
AND	**A ← A AND B**
CLA	**A ← 0**

PUSHA	**Stack ← A, SP ← SP-1**
POPA	**SP ← SP+1, A ← stack**
CLB	**B ← 0**
CMB	**B ← B'**
INCB	**B ← B+1**
DECB	**B ← B-1**
CLflag	**Flag ← 0 (flag can be carry, zero)**
ION	**IEN ← 1, enable interrupts**
IOF	**IEN ← 0, disable interrupts**
SZ	**If Z=1, PC ← PC+1; skip if zero is set**
SC	**If C=1, PC ← PC+1; skip if carry set**
RET	**SP ← SP+1, PC ← stack**

All memory reference instructions use either direct or stack addressing mode and have the format as shown below:

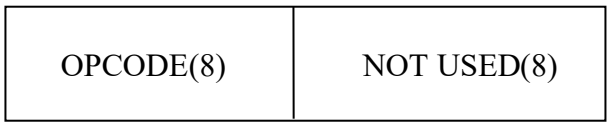


The four most significant bits are used as the operation code (**opcode**) field. As such operation code field can specify up to 16 different instructions. Twelve least significant bits are used as an address for instructions with direct addressing mode or they have no meaning for instructions using stack (implicit) addressing mode. Memory reference instructions with the direct and stack addressing modes are assigned the **opcodes** as shown below:

Opcode[15…12]	**Mnemonic**
0000	**LDA**
0001	**LDB**
0010	**STA**
0011	**STB**
0100	**JMP**
1000	**JSR**
1010	**PUSHA**
1100	**POPA**
1110	**RET**

Instructions in direct addressing mode have the most significant bit equal to 0. Those that use the stack have bit most significant bit equal to 1. The instructions which belong to the register reference instructions and are not using the stack have the most significant bit equal to 0 and four most significant bits equal to HEX 7, and instructions that operate on ***user specified (configurable)*** functional blocks have the most significant bit equal to 1 and four most significant bits equal to HEX F.

The remaining core instructions have the following instruction formats:



The **opcodes** are assigned as below:

01110001	**ADD**
01110010	**AND**
01110011	**CLA**
0111 0100	**CLB**
0111 0101	**CMB**
0111 0110	**INCB**
0111 0111	**DECB**
0111 1000	**CLC**
0111 1001	**CLZ**
0111 1010	**ION**
0111 1011	**IOF**
0111 1100	**SC**
0111 1101	**SZ**

Register reference instructions operate on the contents of working registers (A and B), as well as on individual flag registers used to indicate different status information within the processor or to enable and disable interrupts. Examples of those instructions are ADD and DEC instructions. Program flow control instructions are used to change program flow depending on the results of current computation are simple "skip if zero or carry" set (SZ and SC). These instructions in combination with unconditional JMP instruction can achieve conditional branching to any memory address.

Besides the shown instructions, the microprocessor provides instructions that invoke different application specific functional blocks. These instructions are designated with instruction bits [15…12] set to 1. *The individual instructions are coded by the least significant eight bits [7…0].*

Register Set

Microprocessor contains a number of registers that are used in performing micro-operations. These include 16 bit registers A and B. 12 bit program counter (PC) and 12 bit stack pointer (SP) are not accessible to users. Single bit carry (C), zero (Z) and, interrupt enable (IEN) are also available.

Instruction Execution

The microprocessor core instructions are executed as sequences of micro-operations presented by register transfers. The basic instruction cycle contains all operations from the start to the end of an instruction. It is divided into three major steps that take place in four machine clock cycles denoted by TO, TI, T2, and T3.

1. Instruction fetch is when a new instruction is fetched from an external memory location pointed to by the program counter. It is performed in two machine cycles. The first cycle, TO, is used to transfer the address of the next instruction from the program counter to the address register. The second cycle TI is used to actually read the instruction from the memory location into instruction register, IR. At the same time program counter is incremented by one to the value that usually represents the next instruction address.

2. Instruction decode is the recognition of the operation that has to be carried out and the preparation of effective memory address. This is done in the third machine cycle T2 of the instruction cycle.

3. Instruction execution is when the actual operation specified by the operation code is carried out. This is done in the fourth machine cycle T3 of instruction cycle.

Besides these three fundamental operations in each machine cycle, various auxiliary operations are also performed that enable each instruction to be executed in exactly four machine cycles. They also provide the consistency of contents of all processor registers at the beginning of each new instruction cycle.

Instructions are executed in the same sequence they are stored in memory, except for program flow change instructions. Besides this, the microprocessor provides a very basic single level interrupt facility that enables the change of the program flow based on the occurrence of external events represented by hardware interrupts. A hardware interrupt can occur at any moment since an external device controls it. However, the microprocessor checks for the hardware interrupt at the end of each instruction execution and, in the case that the interrupt has been required, it sets an internal flip-flop called interrupt flip-flop (IFF). At the beginning of each instruction execution, microprocessor checks if IFF is set. If not set, the normal instruction execution takes place.

If the IFF is set, microprocessor enters an interrupt cycle in which the current contents of the program counter is saved on the stack and the execution is continued with the instruction specified by the contents of memory location called the interrupt vector (INTVEC).

The interrupt vector represents the address of the memory location that contains the first instruction of the Interrupt Service Routine (ISR), which then executes as any other program sequence. At the end of the ISR, the interrupted sequence, represented by the memory address saved on the stack at the moment of the interrupt acknowledgment, is returned to using the "RET" instruction.