Computer Architecture 2022/23

TPC 2

Deadline: April 17th, 2023 at 23h59

This homework consists of a programming exercise to be carried out in groups of a maximum of two students. Students can clarify general doubts with colleagues, but the solution and the writing of the code must be strictly carried out by the members of the group. All resolutions will be compared automatically and cases of plagiarism will be punished according to the regulations in force.

The delivery method will be disclosed in due course, using a message via CLIP.

Exercise

In this exercise you have to complete the code of a simulator (written in the C language) of an architecture composed of a very simple CPU and a memory. This includes, already implemented, a console where it is possible for a user to give commands to read and write in the central memory, including reading into memory the values represented in a text file. You can also execute the instructions present in memory. The commands implemented are the following:

`poke` *EEE X* – writes the value *X* to address *EEE*.

`peek` *EEE* – reads the value in memory address *EEE*. The value is posted in hexadecimal.

`dump` *X* – shows all the contents of registers and *X* memory words starting from address 0.

`load` *FILE* – reads the file named *FILE*, interpreting each line of text as a value that is placed sequentially in memory.

`run` – execute instructions from memory address 0.

The values *X* and *EEE* can be given in base 10 or in base 16 if they start with 0x.

The code to be completed is in the dorun.c file and must implement the fetch, decode and execute cycle to simulate the execution of instructions in memory. Execution must always run from memory address 0, until the execution of the HALT instruction (see next section).

Processor machine instructions:

The processor uses 32-bit words and has 16 registers (AC). It also has a register as a Program Counter (PC) and another for the instruction to be executed (IR). Memory is also organized into 32-bit words per address, each address being 12 bits long. There are also two flags

- ZERO contains a non-zero value when the result of the last arithmetic operation is zero and 0 when the last arithmetic operation did not give zero
- POSITIVO contains a value different from zero when the last arithmetic operation gave a result >= 0 and zero when the last arithmetic operation gave a result < 0

The instructions have a fixed size of 32 bits, the 8 most significant bits for the opcode, 12 bits for specifying the registers that contain the two operands and the result of arithmetic instructions and the remaining 12 for the address, if any.

31 24	23 20	19 16	15 12	11 0
Instruction Code	Register 1	Register 2	Register 3	Address

Bits 31-24: Instruction code: specifies the instruction to execute and how bits 23 to 0 should be interpreted.

Bits 23-20: Register 1(source 1 (src1): specifies which register whose content will be operand 1 of the arithmetic and logic instruction specified in bits 31 to 24.

Bits 19-16: Register 2 (source 2 (src2): specifies which register whose content will be operand 2 of the specified arithmetic and logic instruction.

Bits 15-12: Register 3 (destination (dst): specifies which register whose content will contain the result of the specified arithmetic and logic instruction.

Bits 11-0: address: specifies the memory address to be used in the instructions that reference it (load, store, jumps)

There are exceptions to this organization and in some instructions there are bits that are not considered. The supported instructions and their machine code in hexadecimal representation are described below. In the Instruction column, the values presented are sequences of digits in base 16 (hexadecimal), each one corresponding to 4 bits. The following conventions are used:

Symbol	Meaning
X	4 bits to ignore
EEE	12-bit address; it is an unsigned integer
reg	4 bits that specify a register (0x0 to 0xF)
VVVVV	20 bits that specify a value encoded in 2's complement i.e. a signed integer that can represent values from 2^{19} -1 to -2^{19}

Instruction	Mnemonic	Description
00XXXXXX	HALT	Terminate the program
01regVVVVV	LOADI value	The register specified in bits 23 to 20 receives the value VVVVV; bits 31 to 20 of the register receive 0 if bit 19 is 0 and 1 if bit 19 is 1 (sign extension)
02regXXEEE	LOAD reg address	The register specified in bits 23 to 20 receives the contents of the EEE address that is specified in bits 11-0. Bits 19 to 12 are ignored.
03regXXEEE	STORE reg address	The contents of the register specified in bits 23 to 20 are stored in the memory location whose EEE address is specified in bits 11-0. Bits 19 to 12 are ignored.
04reg1reg2reg3XXX	ADD reg1 reg2 reg3	Register 3 (specified in bits 15 to 12) receives the sum of the contents of the register specified in bits 23 to 20 with the contents of the register specified in bits 19 to 16. Bits 11 to 0 are ignored.
05reg1reg2reg3XXX	SUB reg1 reg2 reg3	Register 3 (specified in bits 15 to 12) receives the subtraction of the contents of the register specified in bits 23 to 20 with the contents of the register specified in bits 19 to 16. Bits 11 to 0 are ignored.
06regXXXXXX	CLEAR reg	The content of the register specified in bits 23 to 20 becomes 0.
08XXXXEEE	JUMP EEE	The PC receives the EEE value which is in bits 11 to 0. Bits 24 to 12 are ignored.
09XXXXEEE	JZ EEE	The PC receives the EEE value that is in bits 11 to 0, if the condition code (flag) ZERO is True (like C). Bits 24 to 12 are ignored, as in the following instructions.
0AXXXXEEE	JNZ EEE	The PC receives the EEE value that is in bits 11 to 0, if the condition code (flag) ZERO is False (in C fashion).
0BXXXXEEE	JG EEE	The PC receives the EEE value that is in bits 11 to 0, if the result of the last arithmetic operation (addition or subtraction) gave a number greater than 0. This means that ZERO is False and POSITIVE is True.
0CXXXXEEE	JGE EEE	The PC receives the EEE value that is in bits 11 to 0, if the result of the last arithmetic operation (addition or subtraction) gave a number greater or equal to 0. This means that POSITIVE is True.
0DXXXXEEE	JB EEE	The PC receives the EEE value that is in bits 11 to 0, if the result of the last arithmetic operation (addition or subtraction) gave a number smaller than 0. This means that ZERO is False and POSITIVE is False.
0EXXXXEEE	JBE EEE	The PC receives the EEE value that is in bits 11 to 0, if the result of the last arithmetic operation (addition or subtraction) gave a number less than or equal to 0. This means that either ZERO is true or POSITIVE is False.

The following is an example of a program to multiply two unsigned integers (as an example, to calculate 2x3). Irrelevant bits are set to 0. Note that the file to be loaded into the simulator can only contain the middle column, without the header.

```
addr.    code         mnemonic
0x00:  0x02100010   LOAD r1 0x10 // r1 ← multiplicand
0x01:  0x02200011   LOAD r2 0x11 // r2 ← multiplier
0x02:  0x01300001   LOADI r3 1  // r3 ← 1(to decrement)
0x03:  0x06400000   CLEAR r4    // r4 ← 0 (for the comparisons)
0x04:  0x06500000   CLEAR r5    // r5 ← 0 (to accumulate the product)
0x05:  0x05242000   SUB r2 r4 r2 (r2 is 0 ?)
0x06:  0x0900000A   JZ 0x00A
0x07:  0x04155000   ADD r1 r5 r5
0x08:  0x05232000   SUB r2 r3 r2     //  r2 ← r2 - 1
0x09:  0x08000005   JMP 0x05
0x0A:  0x03500012   STORE r5 0x12
0x0B:  0x00000000   HALT

0x10:  2   (multiplicand  filled with poke)
0x11:  3   (multiplier  filled with poke)
0x12:  0   (result to be read with peek after the program ends)
```

The available `p.code` file contains the code and data for this program in hexadecimal notation, occupying one word of memory per line. Running the simulator, we can load the program by doing `load p.code` and execute it with `run`. We can query the result by looking at what is left in the address variable `0x12` with `peek`. We can change the variables corresponding to the multiplicand and multiplier using the `poke` command. The following is an example of a session to calculate 4x2:

```
cmd> load p.code

cmd> poke 0x10 4
cmd> poke 0x11 2
cmd> run
HALT instruction executed
cmd> peek 0x12
0x12: 0x8
```

Turning it in

Delivery is made via an email addressed to the teacher on your practical lab:

- P1, P3, P5, P7 Professora Maria Cecília Gomes (mcg@fct.unl.pt)
- P2, P4, P6, P8 Professor Kevin Gallagher (k.gallagher@fct.unl.pt)
- P9, P10 Monitor Henrique Campos Ferreira (hjp.ferreira@campus.fct.unl.pt)

Delivery must be made on time by submitting only your *dorun.c* file. Fulfillment of the specification by the delivered code is not the only criterion for defining the grade. Other criteria, such as code quality and solution completeness, are also taken into account.

The program will be compiled with the following command:

```
    cc -Wall -std=c11 -o sim sim.c dorun.c
```