Department of Computer Engineering
Digital Hardware Systems
CpE 3202 - Computer Organization and Architecture

Laboratory Exercise #2
"The Control Unit (CU)"

Instructions: Based on the lecture about the Control Unit, emulate the function of a control unit using C/C++ program. Refer to Fig. 1 for the structure and organization of the control unit.

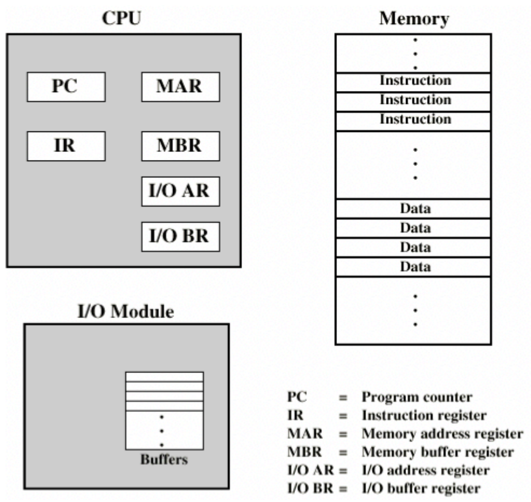

CPU
PC
MAR
IR
MBR
I/O AR
I/O BR
Memory
Instruction
Instruction
Instruction
Data
Data
Data
Data
I/O Module
Buffers
PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = I/O address register
I/O BR = I/O buffer register

Fig. 1. Control Unit Structure and Organization

Since the system is an 8-bit computer, the MBR and I/O BR are 8-bits while registers handling memory address such as the PC, MAR and I/O AR are all 11-bits. Since the instruction width in this exercise is 16-bits, IR is 16-bit as well. The control unit's main function in this exercise is to *1) fetch instruction* & *2) execute instruction* (instruction cycle).

1. Initial function prototype *`int CU(void)`*. Return 1 if operation is successful.
2. PC, IR, MAR, MBR, I/O AR, I/O BR should be declared as local variables. Only the function *`CU(void)`* can access these variables.
3. Create a memory system (see Figure 1) that is 8-bits wide and 2^n number of cells implemented on an array. Since the address is 11-bits therefore n = 11. Starting address of the Program Memory (instruction) is 000_{16} while the Data Memory starts at 400_{16}.
4. Create an I/O buffer which is 8-bits wide and 2^n number of cells where n = 5. Starting address is 000_{16}.
5. For this exercise, construct the following instructions:

Table 1. Instructions

Function	Instruction	Instruction Format	Operand
write to memory	WM	**0000 1**xxx xxxx xxxx$_2$	where 'x' is 11-bit memory address
read from memory	RM	**0001 0**xxx xxxx xxxx$_2$	where 'x' is 11-bit memory address
branch	BR	**0001 1**xxx xxxx xxxx$_2$	where 'x' is 11-bit memory address
read from IO buffer	RIO	**0010 0**xxx xxxx xxxx$_2$	where 'x' is 11-bit memory address
write to IO buffer	WIO	**0010 1**xxx xxxx xxxx$_2$	where 'x' is 11-bit memory address
write data to MBR	WB	**0011 0**000 xxxx xxxx$_2$	where 'x' is 8-bit data
write data to IOBR	WIB	**0011 1**000 xxxx xxxx$_2$	where 'x' is 8-bit data
end of program	EOP	**1111 1**uuu uuuu uuuu$_2$	u - unused bits

Take note of the instruction format. The upper 5 (in boldface) bits are the instruction code or “opcode” which is a unique identifier for the instruction. The next 11 bits are used as the instruction operand. For example, the instruction WB has an instruction code of 00110_2. The operand of this instruction is an 8-bit data to be written to the MBR register. For example:

WB 0x15 *(assembly code)* -> **0011 0**000 0001 0101_2 *(machine code)*

Another example is the RM instruction where the operand is an 11-bit memory address:

RM 0x402 *(assembly code)* -> **0001 0**100 0000 0010_2 *(machine code)*

6. Since the memory is only 8-bits wide, therefore the instruction will be stored in two adjacent memory address in Big Endian order. The fetch cycle requires two “clock cycles” to fetch the instruction: fetching the upper byte first then the lower byte. The fetched instruction will be stored in IR as 16-bit instruction. Below is the fetch cycle written in C.

```
…
…
/* fetching upper byte */
IR = dataMemory[PC];          // get upper byte from memory pointed to by PC
IR = IR << 8;                 // mover the byte to the correct position
PC++;                         // point to the address of the lower byte

/* fetching lower byte */
IR = IR | dataMemory[PC]      // get lower byte from memory pointed to by PC
                              // 16-bit instruction is now fetched
PC++;                         // points to the next instruction
…
…
```

7. After fetching, the Control Unit will decode the instruction by extracting the 5-bit instruction code and its 11-bit operand from the 16-bit instruction in IR.

```
…
…
/* decoding instruction */
inst_code = IR >> 11;         // get the 5-bit instruction code
operand = IR & 0x07FF;        // get the 11-bit operand
…
…
```

8. Once the instruction code and its operand are fetched, the Control Unit will then execute the instruction. It will check first the instruction code. The example below is for the instruction WM (write to memory) and WIO (write to I/O).

```
…
…
if(inst_code==0x01)           // WM instruction (write to memory)
{
        MAR = operand;        // load the operand of WM (memory address) to MAR
        dataMemory[MAR] = MBR; // data in MBR is written to memory address pointed
                              // to by MAR
}
…
…
else if (inst_code==0x05)     // WIO instruction (write to I/O buffer)
{
        IOAR = operand;       // load the operand of WIO (memory address) to IOAR
        ioBuffer[IOAR] = IOBR; // data in IOBR is written to memory address pointed
                              // to by IOAR
}
…
…
```

Follow the data flow as discussed in the lecture on Control Unit. In this exercise the data bus (BUS) and address bus (ADDR) are assumed. It will be covered in the next exercise.

9. The Control Unit shall continue to execute instructions. Conditions like the EOP instruction or unknown instruction code shall force the Control Unit to stop. It shall return a value of 1 if it encounters EOP otherwise 0 if unknown instruction code.

10. Your `main()` should look like this:

```
void main(void)
{
       initMemory();  // this function populates the memory

       if(CU()==1)    // check the return value
              printf("Program run successfully!");
       else
              print("Error encountered, program terminated!");
}
```

11. Write a function `initMemory()` where it will populate the data memory and I/O buffer with instruction or data. Take note of the start address of the program and data memory. As given procedures #3 and #4, the data memory (program and data) and I/O buffer are separate.

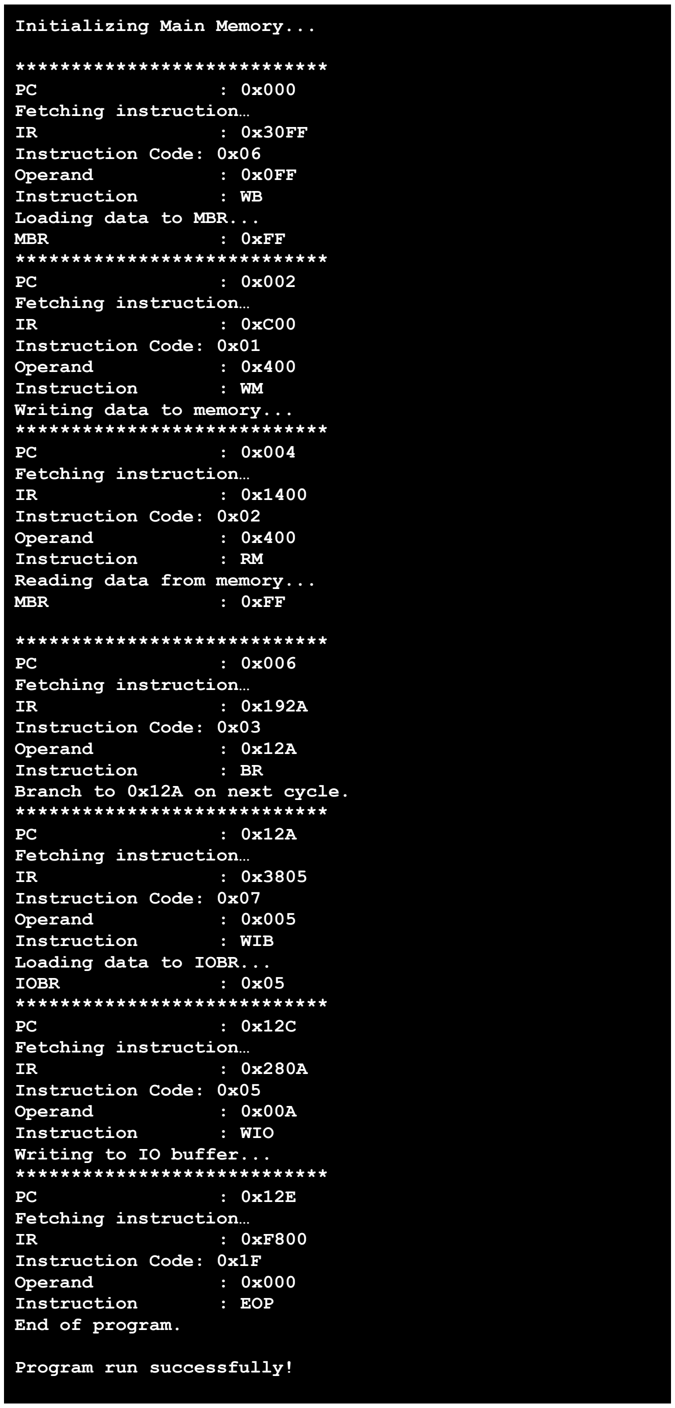
```
void initMemory(void)
{
       dataMemory[0x000] = 0x30;     // WB - write data 0xFF to MBR
       dataMemory[0x001] = 0xFF;
       …
       …
}
```

Populate the data memory with the instructions in the Table 2.

Table 2. Main Memory Data

Memory Address	Data (bin)	Data (hex)	Instruction	Function
0x000	0011 0000	0x30	WB	assigns the data "1111 1111" (0xFF) to MBR
0x001	1111 1111	0xFF		
0x002	0000 1100	0x0C	WM	writes data from MBR to memory address "100 0000 0000B" (0x400)
0x003	0000 0000	0x00		
0x004	0001 0100	0x14	RM	reads data at address "100 0000 0000" (0x400) and store MBR
0x005	0000 0000	0x00		
0x006	0001 1001	0x19	BR	branch to memory address "001 0010 1010" (0x12A)
0x007	0010 1010	0x2A		
…	…	…		…
0x12A	0011 1000	0x38	WIB	assigns the data "0000 0101" (0x05) to IOBR
0x12B	0000 0101	0x05		
0x12C	0010 1000	0x28	WIO	writes data from IOBR to IO buffer address "0 1010" (0x0A)
0x12D	0000 1010	0x0A		
0x12E	1111 1000	0xF8	EOP	end of program
0x12F	0000 0000	0x00		
…	…	…		…

12. Once the `int CU(void)` and `initMemory(void)` function are done, run the Control Unit (CU) from the `main()` (see procedure #10). Echo the process in the console as seen in Figure 2.



```
Initializing Main Memory...

****************************
PC                : 0x000
Fetching instruction…
IR                : 0x30FF
Instruction Code: 0x06
Operand           : 0x0FF
Instruction       : WB
Loading data to MBR...
MBR               : 0xFF
****************************
PC                : 0x002
Fetching instruction…
IR                : 0xC00
Instruction Code: 0x01
Operand           : 0x400
Instruction       : WM
Writing data to memory...
****************************
PC                : 0x004
Fetching instruction…
IR                : 0x1400
Instruction Code: 0x02
Operand           : 0x400
Instruction       : RM
Reading data from memory...
MBR               : 0xFF

****************************
PC                : 0x006
Fetching instruction…
IR                : 0x192A
Instruction Code: 0x03
Operand           : 0x12A
Instruction       : BR
Branch to 0x12A on next cycle.
****************************
PC                : 0x12A
Fetching instruction…
IR                : 0x3805
Instruction Code: 0x07
Operand           : 0x005
Instruction       : WIB
Loading data to IOBR...
IOBR              : 0x05
****************************
PC                : 0x12C
Fetching instruction…
IR                : 0x280A
Instruction Code: 0x05
Operand           : 0x00A
Instruction       : WIO
Writing to IO buffer...
****************************
PC                : 0x12E
Fetching instruction…
IR                : 0xF800
Instruction Code: 0x1F
Operand           : 0x000
Instruction       : EOP
End of program.

Program run successfully!
```

Fig. 2. Echo output on the console.

13. Verify If the output of the instructions are correct. You can experiment by expanding the contents in the program memory by adding more instructions.
14. Save your source code as "<LAST NAME>_CU.c" and submit in Canvas. Make sure that the source code submitted has no compile and run-time errors.

Tips:

- All variables that contain addresses shall have an initial value of NULL.

- Add a `getch()` function before the next instruction cycle so that you can evaluate the instruction one at a time.
- You can pre-write data in the data memory (for testing) as long as it is within the address range.
- Do not write any data to any address the program memory.

Assessment

Criteria	Excellent (10 pts)	Satisfactory (8.5 pts)	Marginal (7.5 pts)	Not Acceptable (5 pts)	Not delivered (0 pt)
Logic	Control Unit logic demonstrated clearly and following exactly the instruction cycle.	Control Unit logic demonstrated with modifications with respect to the instruction cycle.	Control Unit logic demonstrated with some instructions not executed properly.	Control unit logic was not demonstrated, instructions are not executed	
Emulation	Emulation of the Control Unit is very close to the actual.	Emulation of the Control Unit is slightly close to the actual.	Emulation of the Control Unit is ver far from the actual.	Emulation of the Control Unit is not demonstrated.	
Coding	Coding is neat, systematic, logical and followed accepted coding standards.	Coding is logical and somewhat followed some coding standards.	Coding is logical followed little coding standards.	Coding is a mess and did not follow any coding standards.	

Copyright Information

Author: Van B. Patiluna (vbpatiluna@usc.edu.ph)
Contributors: none
Date of Release: February 21, 2021
Version: 1.0

Change log:

Date	Version	Author	Changes
September 19, 2019	3.0	Van B. Patiluna	Original laboratory guide for CpE 415N.
February 21, 2021	1.0	Van B. Patiluna	- Version for CpE 3202. - Revised some procedures. - Added procedures and sample codes. - Removed references to CpE 415N.