Laboratory Exercise #3
"The Buses"

Instructions: Based on the lecture about the control unit and buses, modify the previous Control Unit (CU) from Laboratory Exercise #2 and refer to the following:

1. Modify the Control Unit (CU) on the previous exercise to include the data bus (instruction and data), address bus, control bus (lines). Refer to Fig. 1 for the Control Unit architecture. The computer system architecture is on the appendix (Fig. 1).

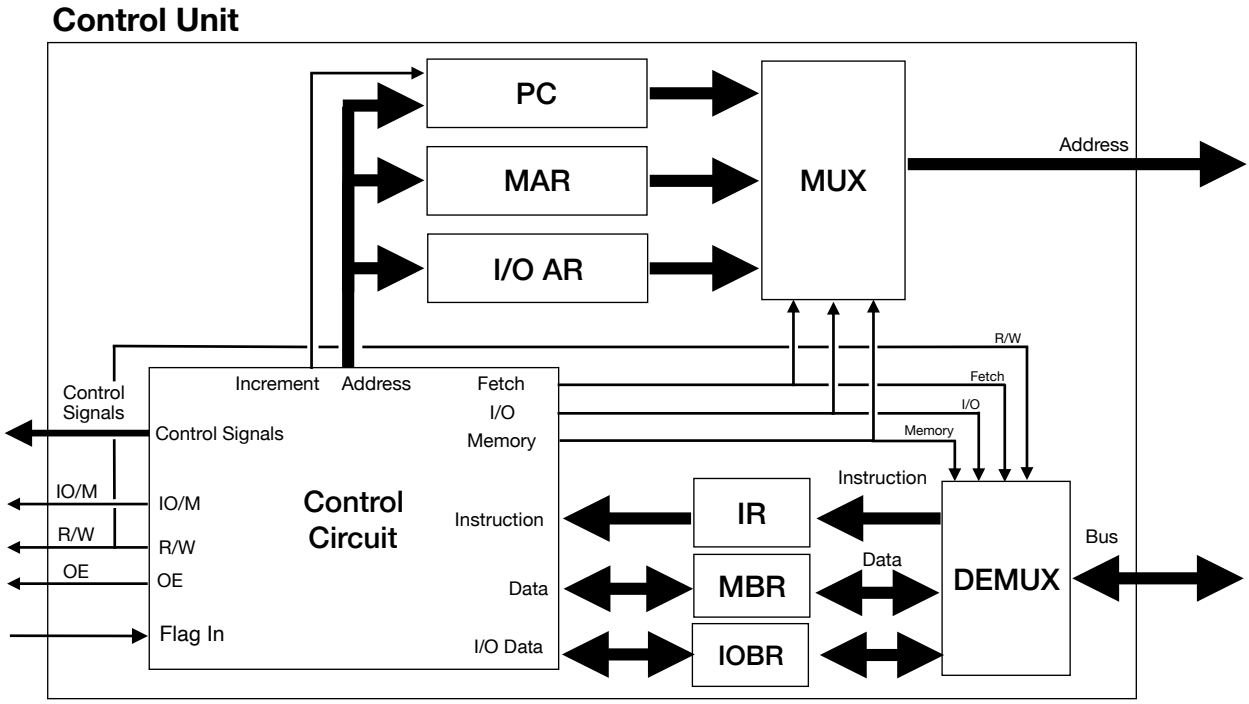


Figure 1. The Control Unit in Detail

2. Implement the Address Bus (Address), Data Bus (Bus) & Control Signals, IO/M and R/W as global variables while the Increment, Fetch, I/O, Memory control lines are local. The bus widths are; Address (11), Bus (8) and Control Signals (5). See Fig. 2 for the bit assignments for the Control Signals.

4	3	2	1	0
Instruction Code				

Figure 2. Control Signal

3. Data going outside and inside must pass through the buses through the multiplexers. For example, sending out the address at the MAR, check the internal control signals (Fetch, I/O, Memory), if the "Fetch" signal is 1, then assign the address at MAR to the address bus (Address); ADDR = MAR, where ADDR is the address bus.

4. Modify *CU( )* such that the fetch cycle be part of the workflow. Since the instruction or data has to pass to the data bus and is demultiplexed so that it can be stored to either MAR or IR. The following lines of code shows the logic of the latter:

```
…
MainMemory();
if(Fetch==1)
  IR = BUS;
…
```

Note that the "Fetch" signal is a local variable (see Fig. 1). You can include all operation during fetch cycle inside this condition.

5. Create the function *and void MainMemory(void).* Maintain the memory arrays created in Laboratory Exercise 2. Accessing the main memory and I/O memory requires calling the functions which returns the instruction/data pointed to by the *address bus (ADDR)*. For example, reading data from main memory; *dataMemory[ADDR]*. The latter access data at address pointed to by ADDR. In this case, IOM = 1 and RW = 0 since the operation is main memory access and data read is performed. The control signal OE will signal the start of read operation. This is important so that no data collision will happen especially when the reading from or writing to the accumulator (ACC). OE should be '1' only if the operation requires access to the main or IO memory access or any bus operations. Data read from memory is assigned to the bus. See Fig. 3 for the block diagram of the Main Memory and I/O Memory. Refer to the code below:

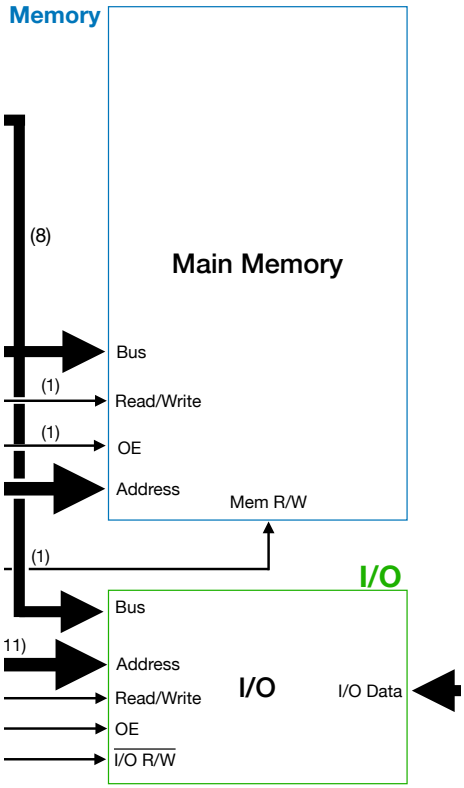


Fig. 3. Main Memory and I/O Memory

```
void MainMemory(void)
{
   if(IOM==1)
   {
     if(RW==0 && OE==1)  // memory read
         BUS = dataMemory[ADDR];
     else if(RW==1 && OE==1)  // memory write
         MainMemory[ADDR] = BUS;
   }
{
```

Create a similar function for the IO buffer as *void IOMemory(void).*

6. Due to the memory width is only 8 bits, the instruction fetch cycle will be done in two (2) cycles by fetching the upper 8 bits first (Big Endian) followed by the lower 8-bits. To do this, fetch the upper 8-bits and assign to IR. Shift IR to the left by 8 bits. Then fetch the lower 8-bits and assign the value to the lower 8 bits of IR. See code below.

```
…
/* setting external control signals */
CONTROL = inst_code;    // setting the control signals
IOM = 1;          // Main Memory access
```

```
RW = 0;           // read operation (fetch)
OE = 1;           // allow data movement to/from memory

/* Fetching Instruction (2 cycle) */
Fetch = 1;        // set local control signal Fetch to 1 to signify fetch operation
I/O = 0;
Memory = 0;

/* fetching the upper byte */
ADDR = PC;
MainMemory();     // fetch upper byte

if(fetch==1)
{
      IR = (int) BUS;   // load instruction to IR
      IR = IR << 8;                 // shift IR 8 bits to the left
      PC++;             // points to the lower byte
      ADDR = PC;        // update address bus
}

/* fetching the lower byte */
MainMemory();           // fetch lower byte
if(fetch==1)
{
      IR = IR | BUS;    // load the instruction on bus to lower
                        // 8 bits of IR
      PC++;             // points to the next instruction
}
/* Instruction Decode */
…
```

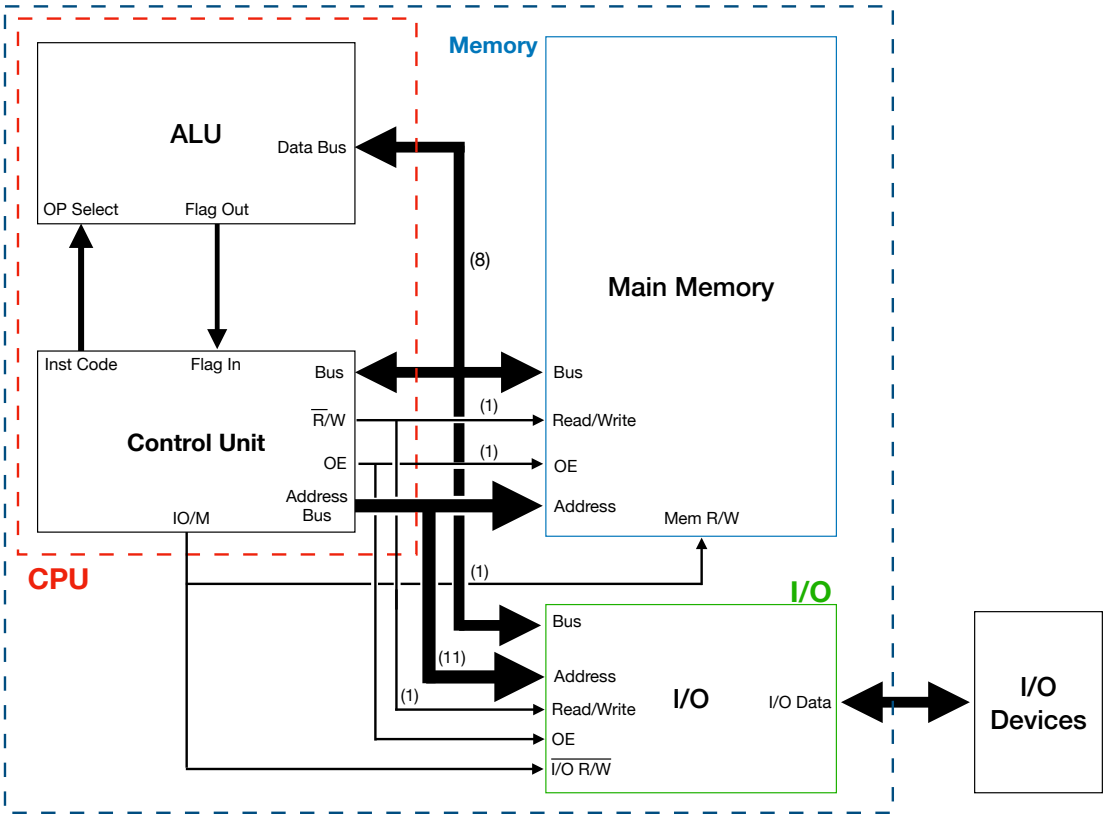
The code above also shows that the PC has ownership of the Address Bus (in fetching both) since the control signal “Fetch” is set to ‘1’.

7. The operations shall also follow the logic of the buses including the local control signals. For example, the instruction WM or “write to memory” is shown below:

```
…
/* Write to Memory */
if (inst_code == 0x0001)      // check instruction code if WM
{
      MAR = operand;    // load the operand to MAR (address)

      /* setting local control signals */
      Fetch = 0;
      Memory = 1;       // accessing memory
      IO = 0;

      /* setting external control signals */
      CONTROL = inst_code;    // setting the control signals
      IOM = 1;          // Main Memory access
      RW = 1;           // write operation
      OE = 1;           // allow data movement to/from memory

      ADDR = MAR;       // load MAR to Address Bus
      if(Memory)
         BUS = MBR;     // MBR owns the bus since control signal Memory is 1
      MainMemory();     // write data in data bus to memory

      /* echo */
}
…
```

8. Modify the process echoing system in Laboratory Exercise #2. Highlight/display the data in the following: BUS, ADDR, PC, MAR, IOAR, IOR, IOBR, CONTROL. Display the latter in hexadecimal or binary.
9. Refer to Laboratory Exercise #2 for the population of the data memory and I/O buffer.
10. Run your program and verify the values of the buses and control signals. The instructions should function the same as Laboratory Exercise #2.
11. Save the program as "<LAST NAME>_CUver2.c" and submit it in Canvas.

Appendix



TRACS Architecture

Assessment

Criteria	Outstanding (10 pts)	Competent (8.5 pts)	Marginal (7.5 pts)	Not Acceptable (5 pts)	Not delivered (5.0)
Logic (45%)	Bus logic demonstrated clearly and following exactly the functions on each type of bus.	Bus logic demonstrated with modifications with to the bus functions.	Bus logic demonstrated with some buses not implemented.	Bus logic was not demonstrated, buses were not used.	
Emulation (45%)	Emulation of the Buses is very close to the actual.	Emulation of the Buses is slightly close to the actual.	Emulation of the Buses is very far from the actual.	Emulation of the Buses is not demonstrated.	
Coding (10%)	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: March 3, 2021
Version: 1.0.2

Change log:

Date	Version	Author	Changes
September 31, 2019	3.2.1	Van B. Patiluna	Original Laboratory Guide from CpE 415N.
February 25, 2021	1.0	Van B. Patiluna	- Removed references to CpE 415N. - Fixed some grammatical & typographical errors.
February 28, 2021	1.0.1	Van B. Patiluna	- Fixed some inconsistencies in the text and program. - Updated the some images.
March 3, 2021	1.0.2	Van B. Patiluna	- Updated the sample code in procedure #6. - Updated Figure 1 to include the Flag In to the Control Unit.