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

Laboratory Exercise #4
"ALU Version 2"

Instructions: Revise the Arithmetic and Logical Unit (ALU) to conform to the computer architecture presented.

Develop a new ALU function. Refer to the following details:

1. Function prototype: `int ALU(void)` - since all the external signals are global, no need to pass it through. The accumulator (ACC) is no longer a global variable but a local variable within `ALU()`. It should be declared as `static int ACC` so that the value will not flushed upon exit of function.
2. For this exercise, all the arithmetic operations are limited to unsigned operations (signed numbers not supported).
3. Figure 1 shows the block diagram of the ALU. The ACC returns to the ALU circuitry as Operand 1 while Operand 2 is directly connected to BUS.
4. Control signals are the *operation select* for the ALU. It only executes instruction within the unit.
5. The 8-bit accumulator can be written and read (see instructions WACC and RACC).
6. ACC can load the data on the BUS to its high byte or low byte depending on the instruction. Inversely, the high byte and low byte can be read and assigned to the BUS using the instructions described in #4.
7. There are a total of nine (9) ALU operations (see Table 1 for details). Before any ALU operation, the control unit sets up the BUS first by assigning the data in MBR to BUS. The code below is from the control unit, which sets up the BUS by loading it with the data on MBR.

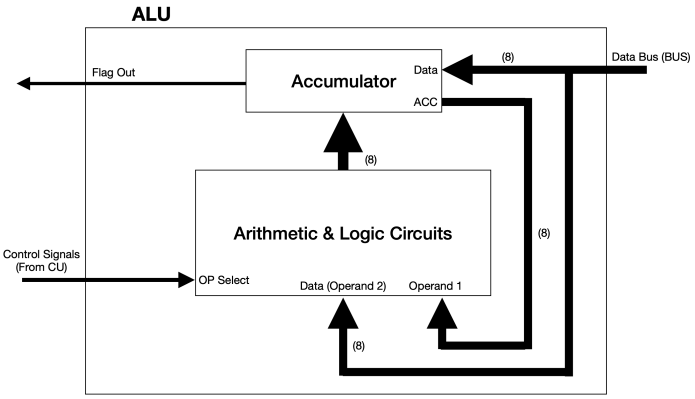


Figure 1. The ALU in Detail

```
…
else if(inst_code==0x1E)                // ADD
{
    /* Setting local control signals */
    Fetch = 0; Memory = 1; IO = 0;      // operation is bus access through MBR

    /* Setting global control signals */
    CONTROL = inst_code;                // setup the Control Signals
    IOM = 0;  RW = 0; OE = 0;           // operation neither "write" or "read"

    if(Memory)
          BUS = MBR;                    // load data on BUS to MBR (ACC high byte

    ALU();                              // executes ALU operation
}
…
```

On the ALU side, the arithmetic and logical operations are performed. The following code shows how the ADD operation on the ALU is done.


```
…
if(CONTROL==0x1E || CONTROL==0x1D)                    // ADD or SUB
{
   /* Sign and Operation Check Logic */
   if(control_signal==0x1D)                           // check if operation is SUB
          temp_OP2 = twosComp(BUS);                   // 2's complement operand2
   else
          temp_OP2 = BUS;

   /* 8-bit Adder */
   temp_ACC = (int) ACC + temp_OP2;
   ACC = (unsigned char) temp_ACC;
}
else if(CONTROL==0x1B)                               // MUL
{
     …
     …
     …
}
…

setFlags(temp_ACC);
```

Setting the bus is true to all arithmetic and logical operations except the NOT or invert operation which does not require a second operand.

8. Create a global variable (unsigned char) called FLAGS and remove the individual flag variables. Modify `setFlags()` so that it will update the flag bits in the FLAGS register. See Figure 2 for more information on the the bit positions of the zero flag (ZF), sign flag (SF), overflow flag (OF) and carry flag (CF).

7	6	5	4	3	2	1	0
OF	**-**	**-**	**-**	**-**	**SF**	**CF**	**ZF**
FLAG Register							

Figure 2. FLAGS register Detail

To update the zero flag (ZF) to '1' for example:

```
FLAGS = FLAGS | 0x01;      // update ZF to '1' at bit position 0
```

9. The assembly code presented below shows the logic of the "memory to memory" load and store operation. It only requires the MBR as the transit of the data to and from memory. It also shows how to write and read the ACC. A literal value can also be passed to the second operand of the ALU through the use of WB or WIB instructions. Refer to Table 1 for details of the instructions.

```
000     WB 0x05              ; write 0x05 to MBR
002     WM 0x400             ; write data on MBR to memory at address 0x400
004     WB 0x03              ; write 0x03 to MBR
006     WM 0x401             ; write data on MBR to memory at address 0x401
008     RM 0x400             ; read address 0x400, store data to MBR
00A     WACC                 ; write MBR to ACC (via BUS)
        ; at this point, ACC = 0x05
00C     RM 0x401             ; read data at memory address 0x401, store to MBR
00E     ADD                  ; add ACC to MBR (through BUS)
        ; at this point, ACC = 0x08
010     RACC                 ; read ACC, store to MBR
012     WM 0x402             ; write data on MBR to memory address 0x402
        ; at this point, memory address 0x402 = 0x08
014     WB 0x00              ; write 0x00 to MBR
016     RM 0x402             ; read address 0x402, store data to MBR
018     WACC                 ; write MBR to ACC (via BUS)
```

```
        ; at this point, ACC = 0x08
01A     WB 0x05             ; write 0x05 to MBR
01C     MUL                 ; ACC to MBR (through BUS)
        ; at this point, ACC = 0x28
01E     RACC                ; read ACC, store to MBR
020     WM 0x403            ; write data on MBR to memory address 0x403
        ; at this point, memory address 0x403 = 0x28
022     SHL                 ; shift left ACC
        ; at this point, ACC = 0x50
024     RACC                ; read ACC, store to MBR
026     WM 0x405            ; write data on MBR to memory at address 0x405
        ; at this point, memory address 0x405 = 0x50
028     EOP
```

10. All the arithmetic and logic operation are implicit where it does not have any operand declared on the instruction; ACC <- ACC <operation> BUS (see Table 1).
11. Create a new source file with the filename “ALU-CU.c”. Combine the `CU()` and `ALU()` functions.
12. For the Control Unit, write the rest of the instructions in Table 1 including the arithmetic/logic operations control signals.
13. Modify the `ALU()` and add the compare-branch instructions (BRE, BRNE, BRGT & BRLT). Remember that the compare is at the ALU and the branch operation is at the CU.
14. Use the same `MainMemory()` and `IOMemory()` functions in Laboratory Exercise #3.
15. Test the integrated CU and ALU using the assembly program in Appendix A. Convert the assembly program into machine code. Verify if the results of the instruction execution are correct.
16. Submit the program in Canvas. Double check if there are no compile errors before submission.

Table 1. Instruction Set

Instruction	Operation Description	Syntax	Operation	Flags	Inst Code	OPCODE	Remarks
	Arithmetic & Logical						
ADD	Adds the data on the BUS to ACC register, sum stored to ACC	ADD	ACC <- ACC + BUS	ZF, SF OF, CF	11110_2	$1111\ 0uuu\ uuuu\ uuuu_2$	u - unused bits
SUB	Subtract the data on the BUS from the ACC register, difference stored to ACC	SUB	ACC <- ACC - BUS	ZF, SF OF, CF	11101_2	$1110\ 1uuu\ uuuu\ uuuu_2$	u - unused bits
MUL	Multiply the value of ACC to BUS, product stored to ACC	MUL	ACC <- ACC x BUS	ZF, SF OF, CF	11011_2	$1101\ 1uuu\ uuuu\ uuuu_2$	u - unused bits
AND	AND the value of ACC and BUS, result stored to ACC	AND	ACC <- ACC & BUS	ZF	11010_2	$1101\ 0uuu\ uuuu\ uuuu_2$	u - unused bits
OR	OR the value of ACC and BUS, result stored to ACC	OR	ACC <- ACC | BUS	ZF	11001_2	$1100\ 1uuu\ uuuu\ uuuu_2$	u - unused bits
NOT	Complement the value of ACC, result stored to ACC	NOT	ACC <- !ACC	ZF	11000_2	$1100\ 0uuu\ uuuu\ uuuu_2$	u - unused bits
XOR	XOR the value of ACC and BUS, result stored to ACC	XOR	ACC <- ACC ^ BUS	ZF	10111_2	$1011\ 1uuu\ uuuu\ uuuu_2$	u - unused bits
SHL	Shift the value of ACC 1 bit to the left, CF will receive MSB of ACC	SHL	ACC <- ACC << 1, CF <- ACC <7>	CF	10110_2	$1011\ 0uuu\ uuuu\ uuuu_2$	u - unused bits
SHR	Shift the value of ACC 1 bit to the right, CF will receive LSB of ACC	SHR	ACC <- ACC >> 1, CF <- ACC <0>	CF	10101_2	$1010\ 1uuu\ uuuu\ uuuu_2$	u - unused bits
	Data Movement						

WM	Write data in MBR to memory at address pointed to by MAR	WM addr	BUS <- MBR	NA	00001_2	0000 1xxx xxxx $xxxx_2$	x - address
RM	Read data from memory with the specified address, stores data to MBR	RM addr	MBR <- BUS	NA	00010_2	0001 0xxx xxxx $xxxx_2$	x - address
RIO	Read data from IO memory with the specified address, stores data to IOBR	WIO addr	IOBR <- BUS	NA	00100_2	0010 0xxx xxxx $xxxx_2$	x - address
WIO	Write data in IOBR to memory at address pointed to by IOAR	WIB addr	BUS <- IOBR	NA	00101_2	0010 1xxx xxxx $xxxx_2$	x - address
WB	Write literal value to MBR	MBR I	MBR <- literal	NA	00110_2	0011 0000 xxxx $xxxx_2$	x - literal
WIB	Write literal value to IOBR	WIB I	IOBR <- literal	NA	00111_2	0011 1000 xxxx $xxxx_2$	x - literal
WACC	Write data on BUS to ACC	WACC	ACC <- BUS	NA	01001_2	0100 1uuu uuuu $uuuu_2$	u - unused bits
RACC	Read ACC to bus	RACC	BUS <- ACC	NA	01011_2	0101 1uuu uuuu $uuuu_2$	u - unused bits
SWAP	Swap data of MBR and IOBR	SWAP	MBR <- IOBR, IOBR <- MBR	NA	01110_2	0111 0uuu uuuu $uuuu_2$	u - unused bits
	Program Control						
BR	Branch to specified address	BR addr	PC <- addr	NA	00011_2	0001 1xxx xxxx $xxxx_2$	x - address
BRE	Compare ACC and BUS, if equal branch to address specified	BRE addr	ACC <- ACC - BUS if ZF = 1 then PC <- addr	ZF, SF OF, CF	10100_2	1010 0xxx xxxx $xxxx_2$	x - address
BRNE	Compare ACC and BUS, if not equal branch to address specified	BRNE addr	ACC <- ACC - BUS if ZF = 0 then PC <- addr	ZF, SF OF, CF	10011_2	1001 1xxx xxxx $xxxx_2$	x - address
BRGT	Compare ACC and BUS, if ACC > BUS, branch to address specified	BRGT addr	ACC <- ACC - BUS if SF = 0 then PC <- addr	ZF, SF OF, CF	10010_2	1001 0xxx xxxx $xxxx_2$	x - address
BRLT	Compare ACC and BUS, if ACC < BUS, branch to address specified	BRLT addr	ACC <- ACC - BUS if SF = 1 then PC <- addr	ZF, SF OF, CF	10001_2	1000 1xxx xxxx $xxxx_2$	x - address
EOP	End of program, no more instructions, CU will halt	EOP	NA	NA	11111_2	1111 1uuu uuuu $uuuu_2$	u - unused bits

Assessment

Criteria	Excellent (10 pts)	Satisfactory (8.5 pts)	Marginal (7.5 pts)	Not Acceptable (5 pts)	Not delivered (0 pt)
Logic	ALU logic demonstrated clearly and following exactly the instruction cycle.	ALU logic demonstrated with modifications with respect to the instruction cycle.	ALU logic demonstrated with some instructions not executed properly.	ALU logic was not demonstrated, instructions are not executed	
Emulation	Emulation of the ALU is very close to the actual.	Emulation of the ALU is slightly close to the actual.	Emulation of the ALU is very far from the actual.	Emulation of the ALU 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: March 8, 2021
Version: 1.0.2

Change log:

Date	Version	Author	Changes
October 5, 2019	2.6	Van B. Patiluna	Original Laboratory Guide for CpE 415N
March 7, 2021	1.0	Van B. Patiluna	- Modified the Flags as a global variable. - Modified Figure 1 to add FLAGS inside ALU
March 8, 2021	1.0.1	Van B. Patiluna	- Integrated the supplementary material.
March 8, 2021	1.0.2	Van B. Patiluna	- Corrected some factual errors. - Revised the test assembly program.

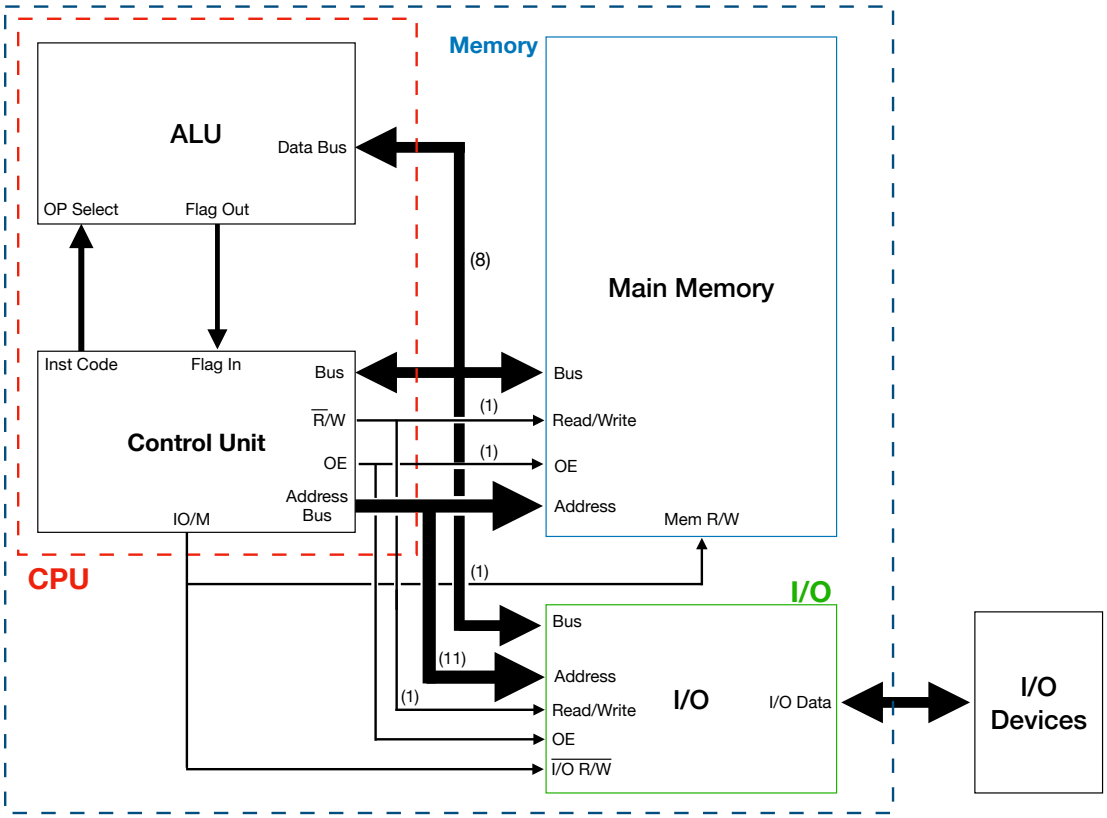
Appendix A

(Test Assembly Program and Machine Code)

```
; program starts here
000     WB      0x15        ; MBR = 0x15
002     WM      0x400       ; dataMemory[0x400] : 0x15
004     WB      0x05        ; MBR = 0x05
006     WACC                ; ACC = 0x05
008     WB      0x08        ; MBR = 0x08
00A     ADD                 ; ACC = (0x05) + (0x08) = 0x0D          ZF=0, CF=0, OF=0, SF=0
00C     RM      0x400       ; MBR = 0x15
00E     MUL                 ; ACC = (0x0D) x (0x15) = 0x11          ZF=0, CF=1, OF=1, SF=0
010     RACC                ; MBR = 0x11
012     WM      0x401       ; dataMemory[0x401] : 0x11
014     WIB     0x0B        ; IOBR = 0x0B
016     WIO     0x000       ; ioBuffer[0x000] : 0x0B
018     WB      0x10        ; MBR = 0x10
01A     SUB                 ; ACC = (0x11) - (0x10) = 0x01          ZF=0, CF=0, OF=0, SF=0
01C     RACC                ; MBR = 0x01
01E     WIO     0x001       ; ioBuffer[0x001] : 0x0B
020     SHL                 ; ACC = (0x01) << 1 = 0x02              ZF=0, CF=0, OF=0, SF=0
022     SHL                 ; ACC = (0x02) << 1 = 0x04              ZF=0, CF=0, OF=0, SF=0
026     SHR                 ; ACC = (0x04) >> 1 = 0x02              ZF=0, CF=0, OF=0, SF=0
024     RM      0x401       ; MBR = 0x11
028     OR                  ; ACC = (0x02) OR (0x11) = 0x13         ZF=0, CF=0, OF=0, SF=0
02A     NOT                 ; ACC = NOT (0x13) = 0xEC               ZF=0, CF=0, OF=0, SF=0
02C     RIO     0x001       ; IOBR = 0x0B
02E     SWAP                ; MBR = 0x0B, IOBR = 0x11
030     XOR                 ; ACC = (0xEC) XOR (0x0B) = 0xE7        ZF=0, CF=0, OF=0, SF=0
032     WB      0xFF        ; MBR = 0xFF
034     AND                 ; ACC = (0xE7) AND (0xFF) = 0xE7        ZF=0, CF=0, OF=0, SF=0
036     RM      0x401       ; MBR = 0x11
038     BRE     0x03C       ; ACC = (0xE7) - (0x11) = 0xD6          ZF=0, CF=0, OF=0, SF=0
03A     WM      0xF0        ; MBR = 0xF0
03C     BRGT    0x040       ; ACC = (0xD6) - (0xF0) = 0xE6          ZF=0, CF=1, OF=1, SF=1
03E     BRLT    0x044       ; ACC = (0xE6) - (0xF0) = 0xF6          ZF=0, CF=1, OF=1, SF=1
040     WB      0x00        ; unreachable
042     WACC                ; unreachable
044     WB      0x03        ; MBR = 0x03
046     WACC                ; ACC = 0x03
```

```
; This part is a controlled loop
048     WB      0x00        ; MBR = 0x00
04A     BRE     0x052       ; ACC = (0x03) - (0x00) = 0x03      ZF=0, CF=0, OF=0, SF=0
                            ; ACC = (0x02) - (0x00) = 0x02      ZF=0, CF=0, OF=0, SF=0
                            ; ACC = (0x01) - (0x00) = 0x01      ZF=0, CF=0, OF=0, SF=0
                            ; ACC = (0x00) - (0x00) = 0x00      ZF=1, CF=0, OF=0, SF=0
                            ; PC <- 0x52 (when ACC == MBR and ZF=1)
04C     WB      0x01        ; MBR = 0x01
04E     SUB                 ; ACC = (0x03) - (0x01) = 0x02      ZF=0, CF=0, OF=0, SF=0
                            ; ACC = (0x02) - (0x01) = 0x01      ZF=0, CF=0, OF=0, SF=0
                            ; ACC = (0x01) - (0x01) = 0x00      ZF=1, CF=0, OF=0, SF=0
050     BR      0x048       ; PC <- 0x048 (loop)
052     EOP
```

Appendix B



TRACS Architecture