**Classical CPU Project:**

**CPU Name: Binary Operation Response Gadget (B.O.R.G)**

**Students:** HaoYu Zhang(260499567)

YiFan, Zhou(260533900)

HaiTong, Yang(260453583)

**Section 1: Overview**

We have built our CPU based on the design of a Classic CPU. It contains most of the components of the Classic CPU: BUS, PC, MAR, MBR, IR, Control Unit, 2 General Purpose registers (GP0, GP1), ALU and Clock. In addition, we have added a keypad and a 2 digit display, each connected by their private bus. The input of each of the components is usually connected to a multiplexer controlled by the C.U which can regulate from where each components receive their data.

In order to implement our instructions, we used a 12-tick "timer" for each instruction in the control unit. Thus one cycle is equal to 12 clock ticks and processes one single instruction. Each cycle starts with fetching the instructions (6 ticks) and ends by incrementing the PC(1 tick). Each components of the CPU is linked to the clock and the control unit by 2 different wires. During each tick in the timer, the C.U sends the signal to the targeted part of the CPU in order to allow them to update with the necessary information in order to execute the instruction.

**Section 2: LOGISIM File Breakdown**

The RAM consists of the main memory along with its AR, and DR. It has a 4-bit input coming from MAR and a 8-bit input that's connected to MBR(also 8-bit output to MBR in case of reading from RAM).

The R/W bit can select whether data goes from memory to DR or the reverse by a MUX

The AR is just a 4-bit register with input from MAR & output to the main memory.

The DR is a byte register whose read/write behavior is controlled by the signal from R/W bit.

**Main memory:**

it takes a 4-bit input from AR as the address. Then the address signal goes through a 4 to 16 decoder that serves as an address selector. There are 16 byte-registers in the main memory. They take a 3-bit input which is determined by the clock signal, signal from the decoder as well as the signal from R/W bit. The signal are ANDed when writing to RAM.

**ALU:**

The ALU that we have implemented can perform 4 operation: OR, AND ADD SUB. It take two 1 byte input as operands and output a 1 byte result with a status register. The status register has 4 flags: Negative, Zero, Carrie, Overflow. The ALU is design to perform 2's complement operation. The carry is different from overflow. Overflow is when adding two positive numbers and get a negative result or adding two negative number to get a positive result. Carry is when there is a carry out bit after an operation. To perform a specific action, there is two control bits(control0, crontol1) that is used to determine which operation to perform.

add: control0:0, control1:1  
 sub: control0:1, control1:1  
 Or: control0:0, control1:0  
 And: control01, control1:0

The subtraction uses the control bit control0 together with one of the input byte is connected to a XOR gate that will invert the input bit. Also, it serves as a constant 1 to be added to a full adder to get the inverted inputs to get the two's complement of that input and add it with the other input.

**Display:**

The display takes 8 bits inputs and display the integer number that it represent on two 7-segments display in hexadecimal form. Each left 4 bits and right 4 bits is connected to a single 8 segment display since each 4 bits can represent 1 hexadecimal number. There is also a control bit to either enable the display or disable the display. The way the display is implemented is that each 4 bits will have 16 different combination and each combination will activate certain segment in the 7-segment to display a number.

**Two's Complement:**

we use a two's complement black box because we can't perform two's complement in the ALU. The implementation of two's complement is standard. it invert the input bits and add one to it.

**MAR:**

The MAR is an 4-bit register that receives its input (ADDR) from either the PC or IR.

**PC:**

The PC also is an 4-bit register that receives its input (ADDR) from either the IR (from bz) or from the PC ADDER.

**GR0/GR1:**

Both are 8-bit registers that receive data from either the MBR or ALU-OUT. These registers send the data to ALU-L and ALU-R when an operation needs to be performed.

**MBR:**

the MBR is an 8-bit register that receive data from RAM, GR0, GR1, Keypad and 2's complement module. Its signals are regulated by a multiplexer controlled by the control unit.

**C.U:**

The control unit fetches and performs every instruction. Each instruction cycle takes exactly 12 ticks. The C.U is connected to every components of the CPU. At each tick, the CPU activates the necessary components of the CPU and their multiplexer in order to allow them to be updated.

**I.R:**

The instruction register is an 8-bit register that receives instruction from MBR. The first 4 bits are sent to the C.U to perform the necessary operations and the 4 last bits are sent to the MAR or the PC, depending on the instruction.

**Section 3: The Assembly Language**

**4.1**

The instructions in the project handout

**4.2**

format: the leftmost 3 bits are OP code, the 4th bit is the parameter, the last 4 bits as address

STR Rx, Address = 0010 xxxx // 4 bits for address

STR Ry, Address = 0011 xxxx

LD Rx ,Address = 0100 xxxx

LD Ry, Address= 0101 xxxx

PRT Rx = 1000 ------- //address field not used

PRT Ry= 1001 -------

INP Rx = 1010 --------

INP Ry = 1011 --------

STOP= 0111 ---------

COMP Rx= 1110 ---------

COMP Ry = 1111 ---------

ADD Rx,Ry= 0000 ----

ADD Ry, Rx= 0001----

BZ Rx,Ry,adress 0110 ----

Sub,Rx,Ry 1100----

**4.3**

In the CU, there are 16 times for each instruction. Each timer has a loop of 12 ticks. After each tick, one output wire coming out of the timer is set to 1.

For every instruction, the first 6 ticks are used for fetching. During that 6 ticks:

MAR<--PC

AR<--MAR

DR<---M(MAR)

MBR<--DR

IR<--MBR

The execution during next 5 ticks vary for each instruction.

**STR Rxy, Address:**

MBR<--Rxy

MAR<--Addr

R/W<--1

AR<--MAR

DR<--MBR

M(AR)<--DR

**LD Rxy :**

Address executes similarly as STR

**PRT Rxy:**

display buffer<--Rxy

print

**INP Rxy:**

MBR<- input buffer

Rxy<-MBR

**COMP:**

2's comp module<-Rxy

MBR<-2's comp module

Rxy<-MBR

**sub R0,R1:**  The first tick after the fetch phase of the execution will load R0 to AL and R1 to AR. AOut control is set to 1,1. The next tick, AOut which has the result: R0-R1 will go to R0.

**Bz R01,R01:** The first 2 ticks after the fetch phase will is the same as the sub instruction. If the result is zero the flag zero of ALU will be set to 1 and it will load pc with the address stored in IR.

**add R0,R1**: The first tick after the fetch phase is load r0 to Al and R1 to AR. The next tick , Aout which has the result of addition will be loaded to R0.  
 we could also store Aout to R1.

**STOP:**  There's an AND gate that has clock an one of its input. the wire of CU that execute the Stop instruction is arranged so that the other wire of the and gates is set to 0 so that the cpu clock stops.

**4.4**

**The program:**

we are computing axb, a is in the RAM, b in the input buffer

the idea is add b+b+b...+b, a times

number a decrements by 1 each time until a=00000001

address instructions

.data

15 PRODUCT # store the product

14 a # stores number a

.text

13 stop # stop program

12 prt rx # display the result of the product

11 ld rx,15 # store the product into rx

10 bz rx,ry,0 # jump back to the start address 0

9 ld rx,2 # load a 1 into rx

8 str rx,14 # store the new a

7 sub rx,ry # a-1

6 bz rx,ry,11 # if a is 1,then exit the loop and display

5 ld rx,14 # load a into rx

4 ld ry,2 # load a 1 in ry.

3 str rx,15 # store the result in address 15

2 add rx,ry # add rx, ry. this is number 1 because the instruction is 00000001

1 inp ry # get value from keyboard

0 ld rx,15 # load the value in address 15 into rx

rx=R0, ry=R1 where R0,R1 are the two general purpose registers.