Pipeline CPU by CYGS

Gabrielle Beaudin 260533363 Sorin Muchi 260575810 Cecily Thomas 260499150
Yaron Bachrach 260497961

December 5, 2014

1 Overview of the CPU

1.1 The CPU diagram

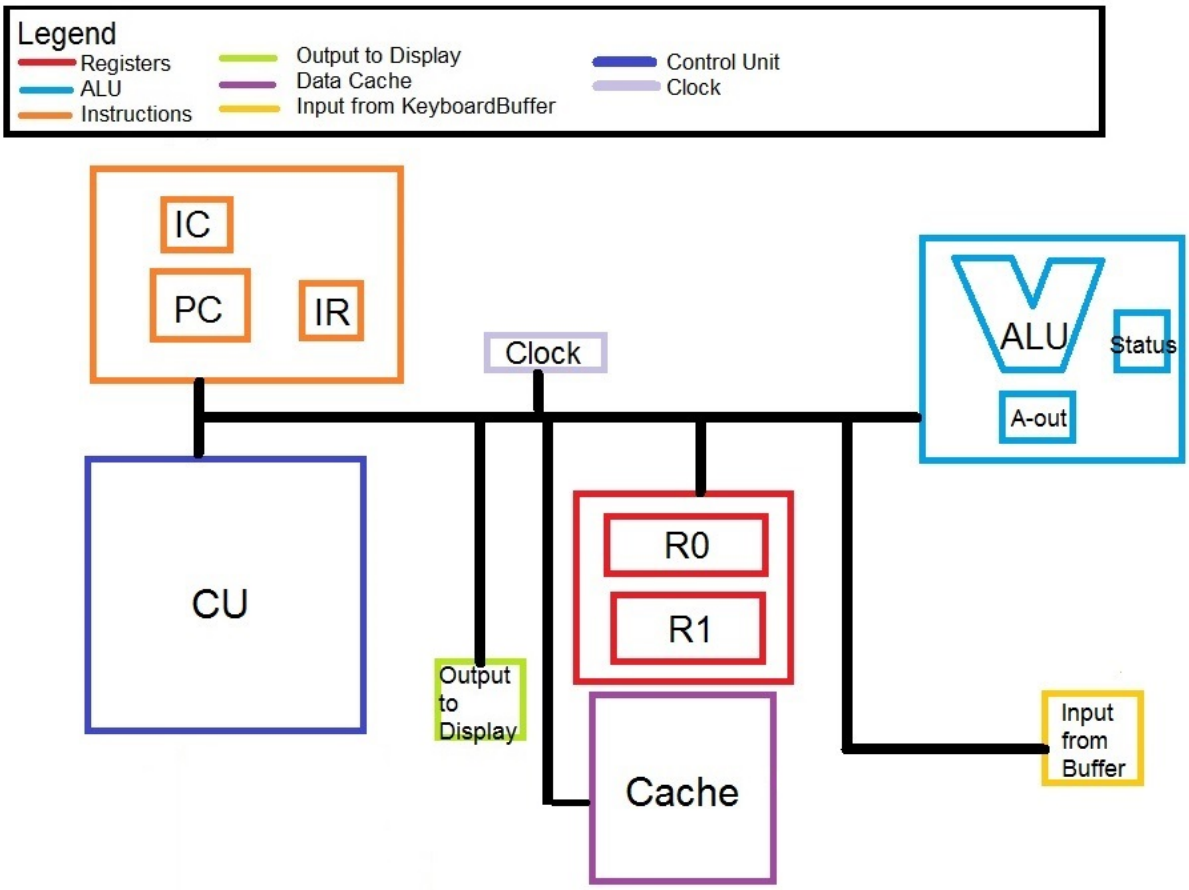

Figure 1: Overview of the CPU Design

2 LOGISIM File Breakdown

Everything.circ uses	CPU.circ
	INPUT.circ
	DISPLAY.circ
CPU.circ uses	ALU.circ
	CU.circ
	DataCache.circ
	REGISTER.circ
	InstructionCache.circ
	EXTRACIRCUIT.circ
PC Increment in EXTRACIRCUIT.circ uses	PC.circ
	ALU.circ

2.1 The CU, PC, IR

The CU as for input the opcode coming from the instruction register. It decodes it and changes the output to match the requested action. For example, if the input is 00010001(ADD 0 1), the Rx output is set to 0, the Ry output is set to 1 and the op-code for the ALU is set to 00.

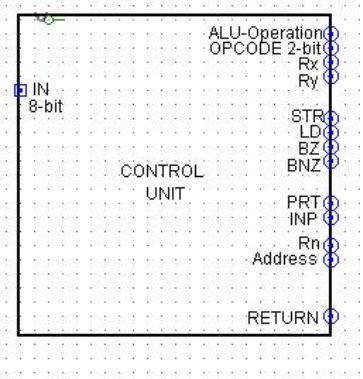

Figure 2: The black box for the Control Unit.

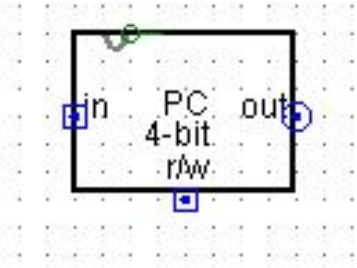

Figure 3: The black box for the Program Counter.

The PC is the same as a register, but uses 4 bits instead of 8 bits. The PC is there to store the address of the next instruction. It is also incremented by 1 after each process is finished. PC increment takes care of incrementing the PC.

The IR is the same as a register. The role of the IR is to store the next instruction.

2.2 The ALU

The ALU takes two inputs. For ADD, it is adding these two input together. For SUB, it is adding the 2s complement of the first input and the second input. For MULT, the inputs need to be 4-bit numbers. It, then, multiple by partial sum. The ALU has a-out and a status register. The a-out stores the answer. The status register stores if there is an overflow(0 if no overflow, 1 if overflow) or if it is zero(0 is not zero, 1 if zero). The ALU has also a R/W input so that when we write in the registers, the a-out cannot be changed.

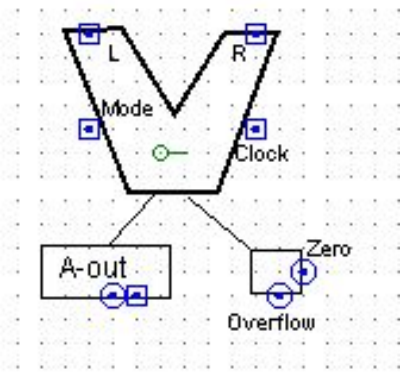

Figure 4: The black box for the Arithmetic logic unit.

2.3 Register

The register is a black box containing 8 D flip-flops, an input for the clock and a input for the read or write. If the input is for read(0), the data stored in the flip-flip is put in the output pins. If the input pins for the register change,

the flip-flops are not affected. If the input is for write(1), the flip-flops are updated with the inputs. However, the output pins are not updated.

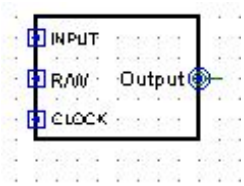

Figure 5: The black box for the components using a register.

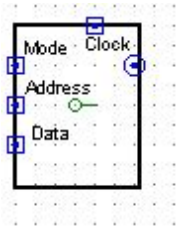

Figure 6: The black box for the component using the cache.

2.4 Data Cache and Instruction Cache

The data cache works like a RAM. It takes a input and if the R/W is write, stores it at the address mention in the address input. If the R/W is read, the data stored at the address mention in the address input is outputted in the output pins.

The instruction cache works the same as the data cache but it is used for storing the instruction.

2.5 The Display

The display decoder take in an 8-bit input and decode it into 7 outputs that corresponds to the a-g inputs for the 7-segment Display. The number 0-9 are displayed and the other numbers display E for error.

2.6 The Input

A 8-bit buffer that store the number inputed by the user.

3 The Assembly Language

Instruction Syntax	Instruction OP-Code	ALU OP-Code	Rx	Ry	Rn	Address (4-bits)	Extra 1-bit	Instruction Binary Format
ADD Rx Ry	000 1	00	0/1	0/1	N/A	N/A	N/A	000100xy
SUB Rx Ry	000 1	01	0/1	0/1	N/A	N/A	N/A	000101xy
MULT Rx Ry	000 1	10	0/1	0/1	N/A	N/A	N/A	000110xy
STR Rx Address	100	N/A	N/A	N/A	0/1	addr	N/A	100naddr
LD Rx Address	101	N/A	N/A	N/A	0/1	addr	N/A	101naddr
BZ Rx Address	110	N/A	N/A	N/A	0/1	addr	N/A	110naddr
BNZ Rx Address	111	N/A	N/A	N/A	0/1	addr	N/A	111naddr
PRT Rx	000 0 0	N/A	N/A	N/A	0/1	N/A	0	000010n0
INP Rx	000 0 1	N/A	N/A	N/A	0/1	N/A	0	000011n0
RETURN	000 000 11	N/A	N/A	N/A	N/A	N/A	N/A	00000011

3.1 Execution of an instruction

3.1.1 ADD,SUB,MULT

For these three operations, the process is the same. The CU output the corresponding op-code that is set to the ALU mode. It also outputs which register is used as Rx and the one used for Ry. These two values then follow a trail of wires which determine the one going into R-left and R-right of the ALU. The ALU that does the operation and the result is stored inside a-out. Then, the result is stored in the register corresponding to Rx.

3.1.2 STR and LD

For store, the CU output 1 for the store output, the 4-bit address and 0 or 1 for Rn. Rn indicate from which register the data will be taken from. The 4-bit address is the cache address where the data needs to be stored. The store black box selects which register and reads the data from that register and writes it at the address in the data cache. Overwriting any data previously there.

For load, the CU output 1 for the load output, the 4-bit address and 0 or 1 for Rn. Rn indicate from which register the data will be store in. The 4-bit address is the cache address where the data will be taken from. The load black box selects which register, reads the data at the address in the data cache and writes it to that register. Overwriting any data previously there.

3.1.3 INP and PRT

For input, the CU output 1 for the input output and output 0 or 1 for Rn. The 8-bit number stored in the buffer is then wrote into the given register.

For print, the CU output 1 for the print output and output 0 or 1 for Rn. Then, it reads the data in the given register and sends the data to the display outside of the CPU.

3.1.4 BZ and BNZ

For branch zero, the CU output 1 for BZ, 0 or 1 for Rn and the address of the instruction to go if true. It reads the value of of the given register and compare it each bit to check if it equal to zero or not. If it is, the PC is updated with the address given. For branch not zero, it is the same thing but it goes to the specified address if the value of the register is not equal to zero.

3.1.5 RETURN

For return, the CU output 1 at the return output. This stop the clock signal in the CPU and stopping all processes.

3.2 Program for the CPU

```
#Setup: we need to put a number into the data cache at address 0,
#and a 1 into the data cache at address 3


# data:
# address 0 holds the number to be multiplied
# address 1 holds the partial sum
# address 2 holds the input/counter
# address 3 holds 1

main:                   # address:
        INP R1          # 0
        STR R1, 2       # 1     save the input number to address 2
        LD R0, 0        # 2     load address 0 into R0
        LD R1, 1        # 3     load address 1 into R1
                        #       (should be zero initially,
                        #       will hold the previous sum afterwards)
        ADD R0, R1      # 4     add R0 to the previous sum
        STR R0, 1       # 5     store the result in address 1
        LD R1, 2        # 6     load the counter into R1
        LD R0, 3        # 7     load 1 into R0
        SUB R0, R1      # 8     decrement the counter
        STR R0, 2       # 9     store the counter back
```

```
BNZ R0, 2          # 10    if the counter is not zero
                   #       go back to instruction at address 2
                   #       if zero, we can display
LD R0, 1           # 11
PRT R0             # 12
RETURN             # 13
```

Address	Binary Instruction
0	00001110
1	10010010
2	10100000
3	10110001
4	00010001
5	10000001
6	10110010
7	10100011
8	00010101
9	10000010
10	11100010
11	10100001
12	00001000
13	00000011

Table 1: The binary instruction to be manually input at the address in the instruction cache.