<Mini CPU>

Created by CO-OP999

6733185821 Penpitcha Piyawaranont

6733255021 Sirikarn Fugsrimuang

6733284221 Atipat Buranavatanachoke

6733293921 Aitsayaphan Limmuangnil

2110252 Digital Computer Logic

Semester 1 Year 2024

Chulalongkorn University

<Mini CPU>

Introduction & Ideas

Mini CPU created by using Digital.exe to run following op-code. The components M, N, prog IN, reset, progLoad, start, result, and clk are inputs. Done, valid, output, 7-segment, and rRam are outputs. So, we designed to divide the functions into <u>13 parts</u>, and write an ASM chart to check the steps we should follow according to the operations of the CPU.

Start with get input, when progLoad = 1 (StartWrite = 1) will start to **Write** and continue to add the value at write by push clk signal (clk = 1).

Read will activate when start = 1 (startRead = 1). We make control units that loop until opcode = 31, then output (valid will = 1). In addition, when press startRead and clk will start working by pulling the readValue out and reading the value in **pRAM**. Then go check (in **Write Address and Value RAM1**), after pressing startRead, it has to look at readValue and readAddress next (in **read** part) to make readValue1 separate into opcode and operand by putting the operand in [DECODER] and checking each command, which commands will (in **A**) when it matches which commands, it will pull that operand to accA (the value in accA will change to that operand when press clk)
*The value in read will change continuously when working (we didn't put the stop command)

Reset will activate if you press reset, then you press clk. It will work in the first round and get 0, there will be a reset at stage 1. Select resetAddress (in the **reset** part), it will go out to AddressOperand2 (in **Write Address and Value RAM2**) then enter AddressOperand at **rRAM**, the value will be 0 (data path in **Write Address and Value RAM2**). If there is no opcode=11, always keep it 0. If the reset1 and resetAddress buttons are on at the same time, when running, it will overwrite address(operand2)0, store value=0 (in **rRAM**). When you press clk to continue working, value2=0, but resetAddress is 1 because it comes from the command (in **reset** part), it is +1 (in reset part, to make address +1 continuously).

Result will run the values continuously by taking resultAddress (from **result** part) and putting it in resultAddress (in **Write Address and Value RAM2** as well). When you press result1 to print 16 no., it will be sent to AddressOperand. When there is result1, the output will be out, with result0 output = 0 and result1 output = 1, and increase rounds continuously, if we do.

Jump will activate when opcode=12 | equal flag=1(opcode=13) | greater flag=1(opcode=14) | lesser flag=1(opcode=15) | equal or greater flag = 1 (opcode=16). [use AND, OR]. Then, jump to do statement in address followed by operand (in **Read**). *we use opcode in decimal

A will work when opcode=1, 3, 6 , 9, 10, 17-22, 24-28. When opcode=1 it will store operand into accA, opcode=3 accB into accA, opcode=6 registerC into accA, opcode=9 store p-RAM Address-Operand- into accA, opcode=10 store r-RAM Address-Operand- into accA. opcode=17-22 do arithmetic (+,-,*,/,%,^) into A (+,- do in 2's complement), opcode=24 switch bits accA and store in accA, opcode=25-27 [use OR, AND, XOR respectively] of accA and accB then store into accA, opcode=28 accA shift accB and store into accA.[use SHIFT]

B will work when opcode=2 and 4. When opcode=2 it will register operand into accB and when opcode=4 it will register accA into accB. [So, use MUX and OR (opcode 2, 4, reset) before connecting to REG.]

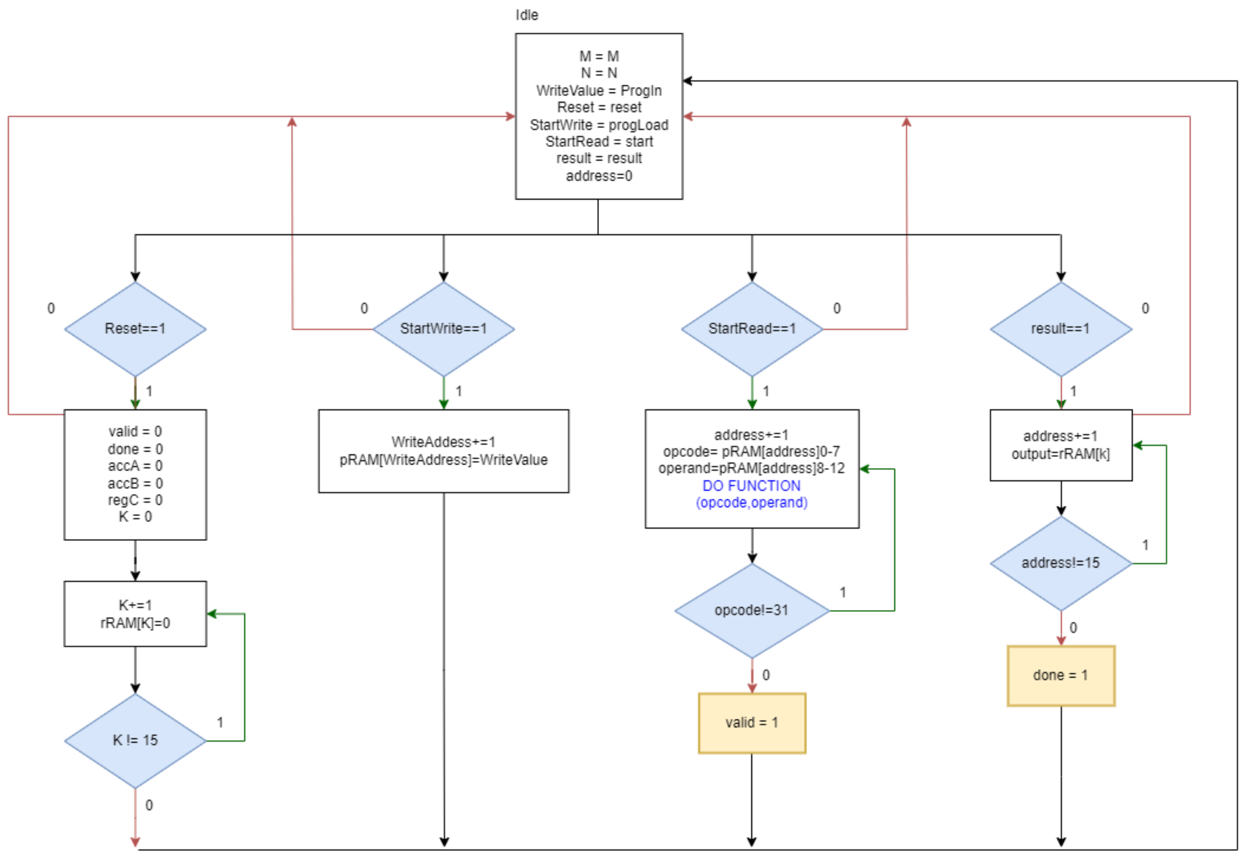
C will work when opcode=5, 7, and 8. When opcode=5, it will register accA into regC. When opcode=7, it will register M into regC. When opcode=8, it will register N into regC. [So, use MUX and OR (opcode 5, 7, 8, reset) before connecting to REG.]

Greater Lesser Equal will compare the values when opcode=23 and then pass them to register the values at equal, greater, or lesser. The opcode=29 is also used to check accA. If there is a prime number, it will also be registered in equal [the equal condition use OR. (opcode23 or 29)].

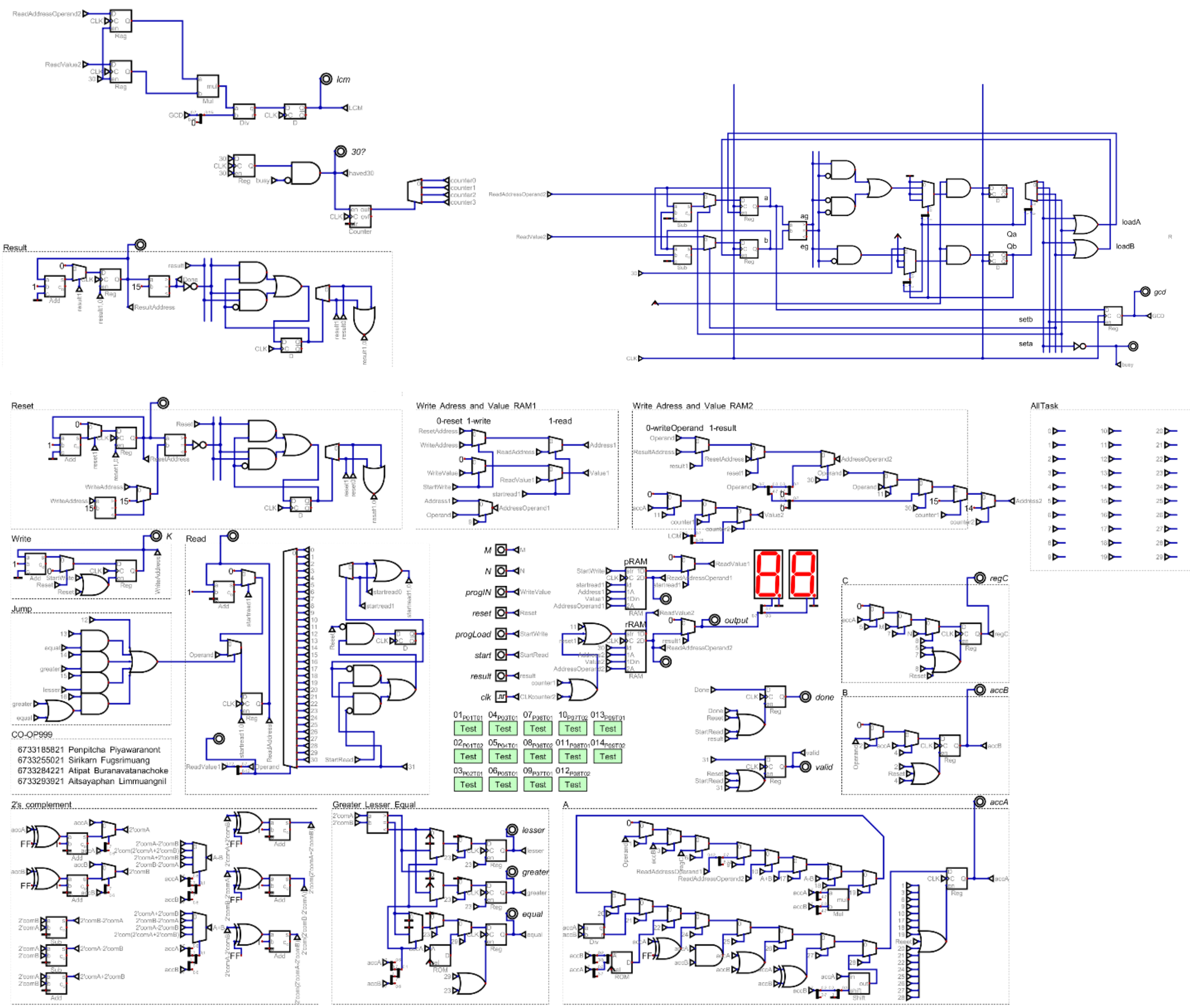
We also add **2's complement** (ALU) to switch bits by xor with FF. To avoid negative value problems.

LCM will work when opcode=30, it will have a state to find the GCD value. When the values are completely found, it will calculate the LCM value using the formula LCM(n,m)=(n*m)/GCD(n,m). When the LCM value is obtained, it will loop to insert it into rRAM address E and address F twice.

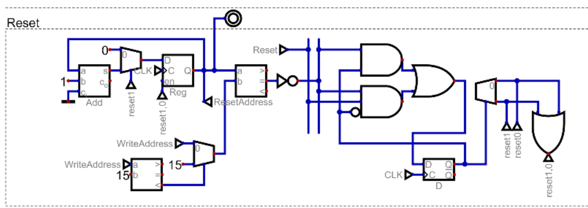
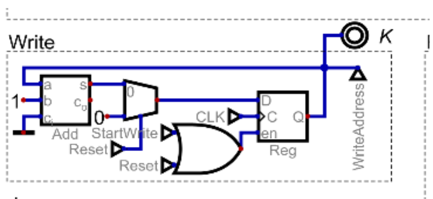
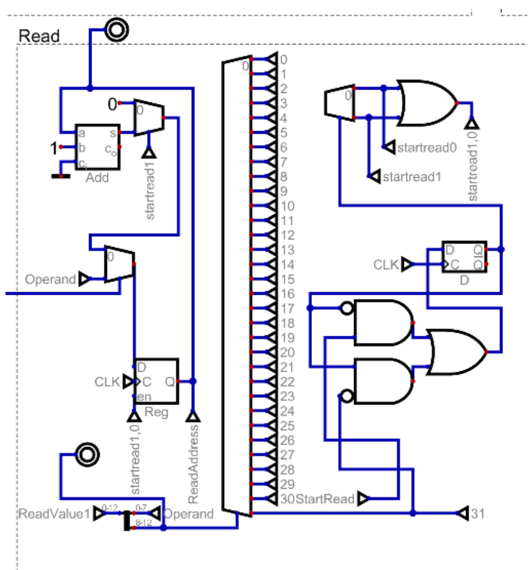
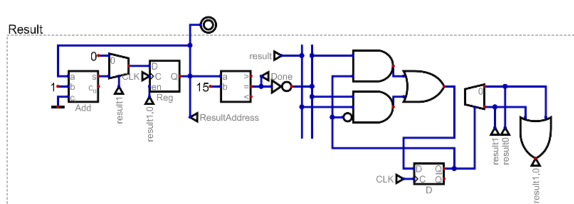
ASM Chart

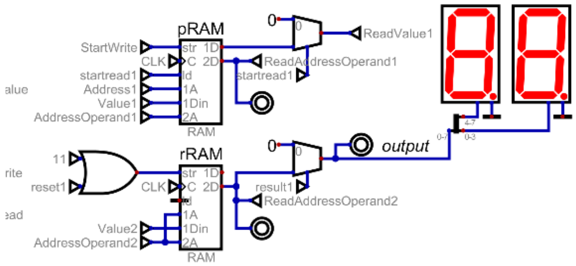
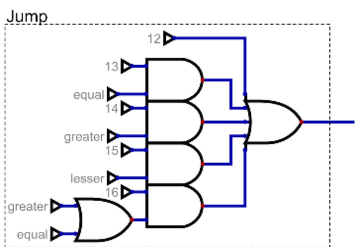
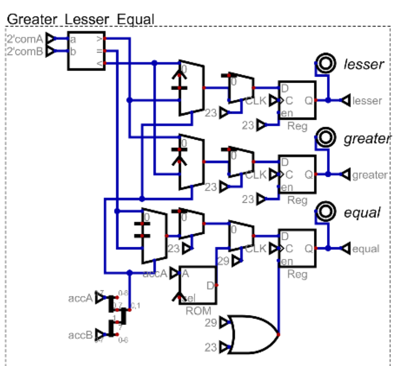
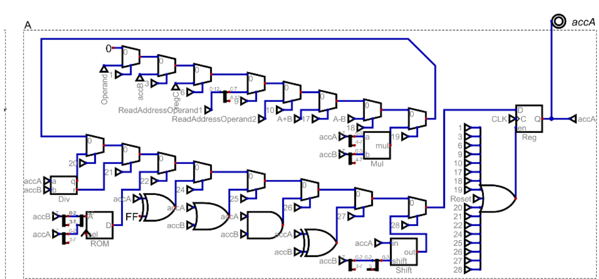

Idle
M = M
N = N
WriteValue = ProgIn
Reset = reset
StartWrite = progLoad
StartRead = start
result = result
address=0
Reset==1
StartWrite==1
StartRead==1
result==1
valid = 0
done = 0
accA = 0
accB = 0
regC = 0
K = 0
WriteAddess+=1
pRAM[WriteAddress]=WriteValue
address+=1
opcode= pRAM[address]0-7
operand=pRAM[address]8-12
DO FUNCTION
(opcode,operand)
address+=1
output=rRAM[k]
K+=1
rRAM[K]=0
K != 15
opcode!=31
valid = 1
address!=15
done = 1

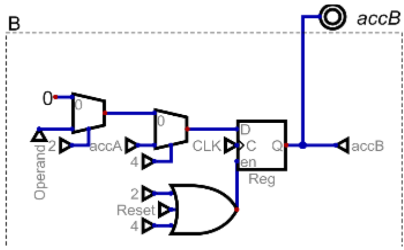
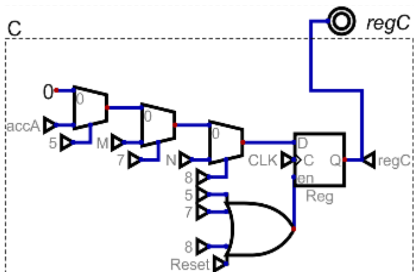
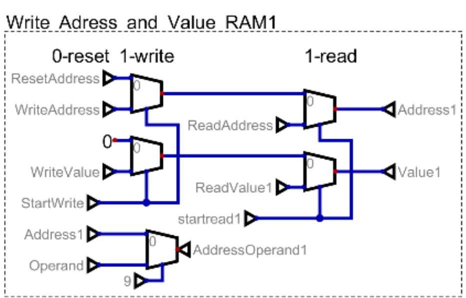
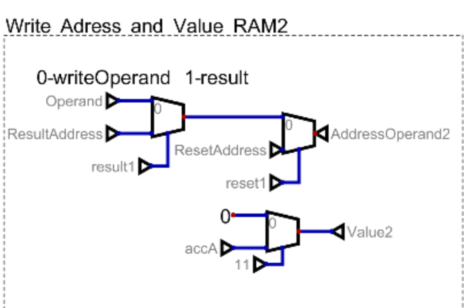
Mini CPU

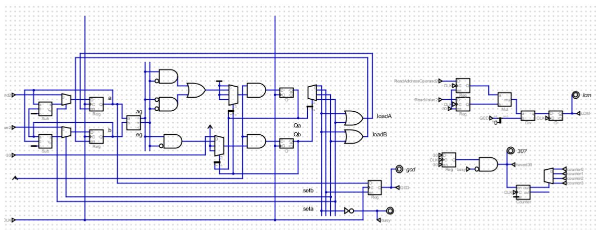

Result
Reset
Write Adress and Value RAM1
Write Adress and Value RAM2
AllTask
Write
Read
Jump
CO-OP999
6733185821 Penpitcha Piyawaranont
6733255021 Sirikarn Fugsrimuang
6733284221 Atipat Buranavatanachoke
6733293921 Aitsayaphan Limmuangnil
2's complement
Greater Lesser Equal
A
B
C
pRAM
rRAM
lcm
30?
gcd
loadA
loadB
setb
seta
regC
accB
accA
done
valid
output
lesser
greater
equal
M
N
progIN
reset
progLoad
start
result
clk

Data paths & Control units

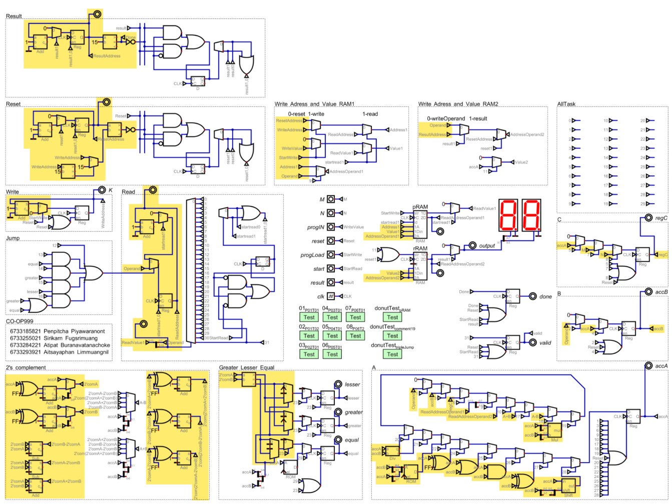
<table>
<tr><th>Functions</th><th>Descriptions</th></tr>
<tr><td></td><td><u>Reset</u>
Data paths
- ResetAddress
- WriteAddress
Control units
- reset0
- reset1
- reset1,0
- reset
- CLK</td></tr>
<tr><td></td><td><u>Write</u>
Data paths
- WriteAddress
Control units
- Reset
- StartWrite
- CLK</td></tr>
<tr><td></td><td><u>Read</u>
Data paths
- Operand
- ReadValue1
Control units
- startread0
- startread1
- startread1,0
- StartRead
- 0-31 (Operand)
- CLK</td></tr>
<tr><td></td><td><u>Result</u>
Data paths
- ResultAddress
- Done
Control units
- Result
- result0
- result1
- result1,0
- CLK</td></tr>
</table>

Functions	Descriptions
	**<u>pRam rRam & Output</u>** **Data paths** - Address1 - value1 - AddressOperand1 - value2 - AddressOperand2 **Control units** - StartWrite - Startread1 - 11 (Operand) - reset1 - CLK
	**<u>Jump</u>** **Data paths** - (No data paths) **Control units** - 12,13,14,15,16 (Operand) - equal - greater - lesser
	**<u>Greater Lesser Equal</u>** **Data paths** - 2'comA - 2'comB - accA - accB - lesser - greater - equal **Control units** - 23,29 (Operand) - CLK
	**<u>Accumulator A (AccA) & ALU</u>** **Data paths** - Operand1 - Operand2 - accA - accB - ALU (A+B, Mul, Div) - Shift - ROM - output register **Control units** - MUXes - control signals (accA, accB) - program counter - CLK

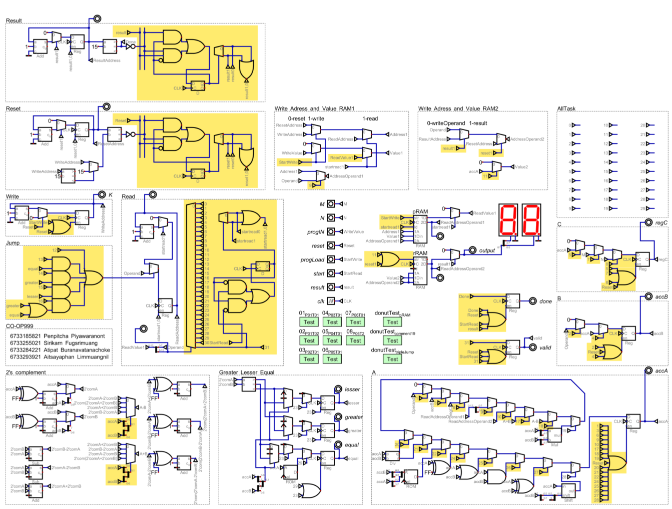
Functions	Descriptions
	**<u>Accumulator B (AccB)</u>** **Data paths** - Operand - accA - accB **Control units** - 2,4 (Operand) - Reset - CLK
	**<u>Register C (RegC)</u>** **Data paths** - accA - regC - M - N **Control units** - 5,7,8 (Operand) - Reset - CLK
	**<u>Write Address and Value RAM1 (pRam)</u>** **Data paths** - WriteAddress - ReadAddress - Address1 - Operand1 - WriteValue - ReadValue1 - Value1 - AddressOperand1 **Control units** - 9 (Operand) - startread1 - startWrite
	**<u>Write Address and Value RAM2 (rRam)</u>** **Data paths** - Operand - ResultAddress - ResetAddress - accA - AddressOperand2 - Value2 **Control units** - 11 (Operand) - result1 - reset1

Functions	Descriptions
	**<u>Future (LCD)</u>** **Data paths** - ReadAddressOperand2 - ReadValue2 - a>b? - a=b? - 30 - busy? **Control unit** - LoadA - LoadB - Seta - setb

Data Paths

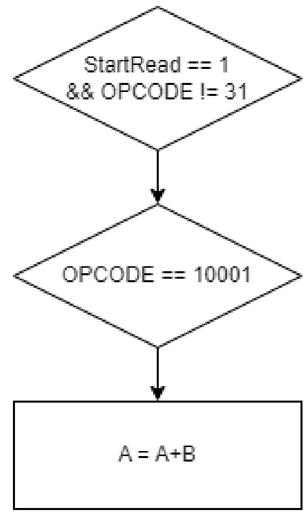


Control Units



Example : OPCODE 10001 (accA ← accA + accB)

OPCODE & OPERAND	FUNCTION
0000100001111	accA ← Operand
0001000110101	accB ← Operand
1000100000000	accA ← accA + accB
0101100000011	RAM_add[Operand] ← accA
1111100000000	STOP



User

1. start
2. load **OPCODE & OPERAND** into **ProgIN**
3. press **ProgLoad** 1 clock
4. If loading is not complete, do **2.** Again
5. press **Start**
6. wait until **valid** show 1
7. press **Result**

CPU (Do Function)

- **OPCODE == 00001**
    1. read OPERAND
    2. accA ← OPERAND
- **OPCODE == 00010**
    1. read OPERAND
    2. accB ← OPERAND
- **OPCODE == 10001**
    1. accA ← accA + accB
- **OPCODE == 01011**
    1. read OPERAND
    2. rRam[OPERAND] ← accA
- **OPCODE == 111111**
    1. Valid = 1
    2. STOP