Microcode decoder & processing control circuit

neverfear.org

Table of Contents

Overview of architecture

It's important to note a few things about my architecture before understanding the circuit diagram.

1. The word size of the circuit is 8 bits
2. There are 3 buses
    a. Instruction bus which is 12 bits wide
    b. Control bus which is 16 bits wide (although currently only 9 bits are used)
3. Output bus which is 8 bits wide
4. The microcode instruction set is currently configured with 8 instruction OpCodes
5. Instructions are 12 bits wide
    a. The first 4 bits is the OpCode
    b. The second 8 bits is operand X
6. Not all instructions require X (namely any arithmetic instruction does not require X)
7. Operands to arithmetic instructions are always assumed to be present in the registers named RegA (first operand) and RegB (second operand)
8. RAM addressing is always direct
9. The output register is referred to as operand O in the instruction set
10. With the current control line specification, only a max of 4 ALU functions may be set (since as you'll see there is only 2 ALU function bits allocated for each operand currently)

The instruction set

Given below is a table of available instructions that contains a description of the instruction and required parameters.

Instruction	OpCode	Requires X?	Description
Load X -> A	0000	Yes	Load the literal value X into register A
Load X -> B	0001	Yes	Load the literal value X into register B
Add A B	0010	No	Add the values in register A and B
Sub B A	0011	No	Subtract the value in register B from the value in register A
Load M(X) -> A	0100	Yes	Load the value at memory address X into register A
Load M(X) -> B	0101	Yes	Load the value at memory address X into register B
Load O -> M(X)	0110	Yes	Load the value in register O into memory at address X
Inc A	0111	No	Increment the value in register A by 1

Control signals

The OpCode is passed into a programmable read only memory bank and outputs the µCode control signals for each instruction. Below is a table of currently available control signals and their effects.

		Effect when signal is	
Control Line #	**General Description**	**0**	**1**
0	Select input from X	The value of X is used in the circuit	The value of X is not used
1	Register A access	Output value in register A	Write input value to register A
2	Register B access	Output value in register B	Write input value to register B
3	Register O access	Output value in register O	Write input value to register O
4	Memory bank M access	Write input value to M at address X	Read value of memory M at address X
5	Select use of operand M	Select use of component M	M is not used
6	ALU function bit 0	ALU function map address bit	
7	ALU function bit 1	ALU function map address bit	
8	Select use of FuncMap	Select use of FuncMap	FuncMap is not used

Given below is a table of OpCodes and their control signals

		Control Line #								
Instruction	**OpCode**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Load X -> A	0000	0	1	0	0	1	1	0	0	1
Load X -> B	0001	0	0	1	0	1	1	0	0	1
Add A B	0010	1	0	0	1	1	1	0	0	0
Sub B A	0011	1	0	0	1	1	1	1	0	0
Load M(X) -> A	0100	1	1	0	0	1	0	0	0	1
Load M(X) -> B	0101	1	0	1	0	1	0	0	0	1
Load O -> M(X)	0110	1	1	1	0	0	0	0	0	1
Inc A	0111	1	0	0	1	1	1	0	1	0

ALU functions

Given below is the function map table. There are currently only 3 defined ALU operations

		ALU Mode Select Bits				
Operation	**FuncMap Address**	0	1	2	3	Cn
Add	00	1	0	0	1	1
Subtract	01	0	1	1	0	0
Increment	10	0	0	0	0	0

An example simulation

The simulation program

To demonstrate this circuit let's similar a simple "program" that demonstrates the use of all of the instructions. Let's imagine our program is this:

```
Load 0x04 -> A
Inc A
Load O -> M(0x00)
Load 0x02 -> B
Load M(0x00) -> A
Sub B A
Load O -> M(0x01)
Load 0x02 -> A
Load M(0x01) -> B
Add A B
Load O -> M(0x02)
```

What does the simulation program do?

The program does this:

1. Load RegA with 0x04
2. Increment A by 1 and place the result in the output register (the output value will be 0x05)
3. Save the output value into memory at address 0x00
4. Load RegB with 0x02
5. Load RegA with the value in memory at address 0x00 (our previously calculated value of 0x05)
6. Subtract 0x02 from 0x05 (the output value will be 0x03)
7. Save the output value into memory at address 0x01
8. Load RegA with 0x03
9. Load RegB with the value in memory at address 0x01 (our previously calculated value of 0x03)
10. Add 0x03 to 0x03 (the output value will be 0x06)
11. Save the output value into memory at address 0x02

Running the simulation

To run this program in CircuitMaker follow this procedure:

1. Load 0x04 -> A
    a. Set the component labelled “OpCode” to the value 0
    b. Set the component labelled “ArgX1” to 0
    c. Set the component labelled “ArgX2” to 4
    d. Run simulation
    e. Stop simulation
2. Inc A
    a. Set the component labelled “OpCode” to the value 7
    b. Run simulation
    c. Stop simulation
3. Load O -> M(0x00)
    a. Set the component labelled “OpCode” to the value 6
    b. Set the component labelled “ArgX1” to 0
    c. Set the component labelled “ArgX2” to 0
    d. Run simulation
    e. Stop simulation
4. Load 0x02 -> B
    a. Set the component labelled “OpCode” to the value 1
    b. Set the component labelled “ArgX1” to 0
    c. Set the component labelled “ArgX2” to 2
    d. Run simulation
    e. Stop simulation
5. Load M(0x00) -> A
    a. Set the component labelled “OpCode” to the value 4
    b. Set the component labelled “ArgX1” to 0
    c. Set the component labelled “ArgX2” to 0
    d. Run simulation
    e. Stop simulation
6. Sub B A
    a. Set the component labelled “OpCode” to the value 3
    b. Run simulation
    c. Stop simulation
7. Load O -> M(0x01)
    a. Set the component labelled “OpCode” to the value 6
    b. Set the component labelled “ArgX1” to 0
    c. Set the component labelled “ArgX2” to 1
    d. Run simulation
    e. Stop simulation
8. Load 0x03 -> A
    a. Set the component labelled “OpCode” to the value 0
    b. Set the component labelled “ArgX1” to 0
    c. Set the component labelled “ArgX2” to 3
    d. Run simulation

    e. Stop simulation
9. Load M(0x01) -> B
    a. Set the component labelled “OpCode” to the value 5
    b. Set the component labelled “ArgX1” to 0
    c. Set the component labelled “ArgX2” to 1
    d. Run simulation
    e. Stop simulation
10. Add A B
    a. Set the component labelled “OpCode” to the value 2
    b. Run simulation
    c. Stop simulation
11. Load O -> M(0x02)
    a. Set the component labelled “OpCode” to the value 6
    b. Set the component labelled “ArgX1” to 0
    c. Set the component labelled “ArgX2” to 2
    d. Run simulation
    e. Finish simulation

The memory component at the end of this program will look like this:

Memory Address	Value
0x00	0x05
0x01	0x03
0x02	0x06