Faculty of Computing

Lab 5 Tasks

Computer Architecture

Name: Mahnoor Farrukh

SAP ID: 52926

Semester: BSCS-4

Write a detailed explanation of how the Fetch-Decode-Execute cycle works.

Answer: The Fetch-Decode-Execute cycle is the core process that a CPU follows to execute instructions. It consists of three main stages:

Fetch: The CPU fetches the instruction from memory. The Program Counter (PC) holds the address of the next instruction. This address is copied into the Address Register (AR), and the CPU retrieves the instruction from memory and stores it in the Instruction Register (IR). The PC then increments to point to the next instruction.

Decode: The CPU decodes the instruction in the IR. It determines what operation needs to be performed, and which data is involved.

Execute: The CPU performs the operation. If it's an arithmetic or logic operation, the Arithmetic Logic Unit (ALU) processes the data. Results are stored in the Accumulator (AC) or another register.

Use simple instruction as an example and describe each step.

Answer: Let's take the instruction: ADD 5 (add the value at memory location 5 to the accumulator).

Fetch:
PC holds the address of the instruction. Let's say PC = 10.
The address (10) is copied to the AR.
The CPU fetches the instruction ADD 5 from memory location 10 and places it into the IR.
PC increments to 11 (next instruction).

Decode:
The CPU decodes ADD 5. It recognizes this as an ADD operation and identifies 5 as the memory location holding the data.

Execute:
The AR is set to 5, and the data from memory location 5 is loaded into the Data Register (DR).
The ALU adds this value to the Accumulator (AC).
The result remains in the AC for future use or output.

Explain the role of PC, AR, IR, AC, and DR in your own words.

Answer:
Program Counter (PC): It keeps track of the next instruction that is to be executed.

Address Register (AR): Holds the memory address currently being accessed — either for instructions or data.

Instruction Register (IR): Temporarily holds the fetched instruction so the CPU can decode and execute it.

Accumulator (AC): Stores intermediate results and final output of arithmetic and logic operations.

Data Register (DR): Temporarily holds data fetched from memory or waiting to be written back to memory.

What is the function of the Arithmetic Logic Unit (ALU) in CPU operations? How does ALU interact with registers and memory?

Answer: The ALU performs all arithmetic and logical operations inside the CPU. It handles basic math like addition, subtraction, multiplication, and division, as well as logic comparisons like AND, OR ,NOT, etc.

For example:

If the instruction is ADD, the ALU adds two numbers — one from the AC and the other from the DR — and stores the result back into the AC.

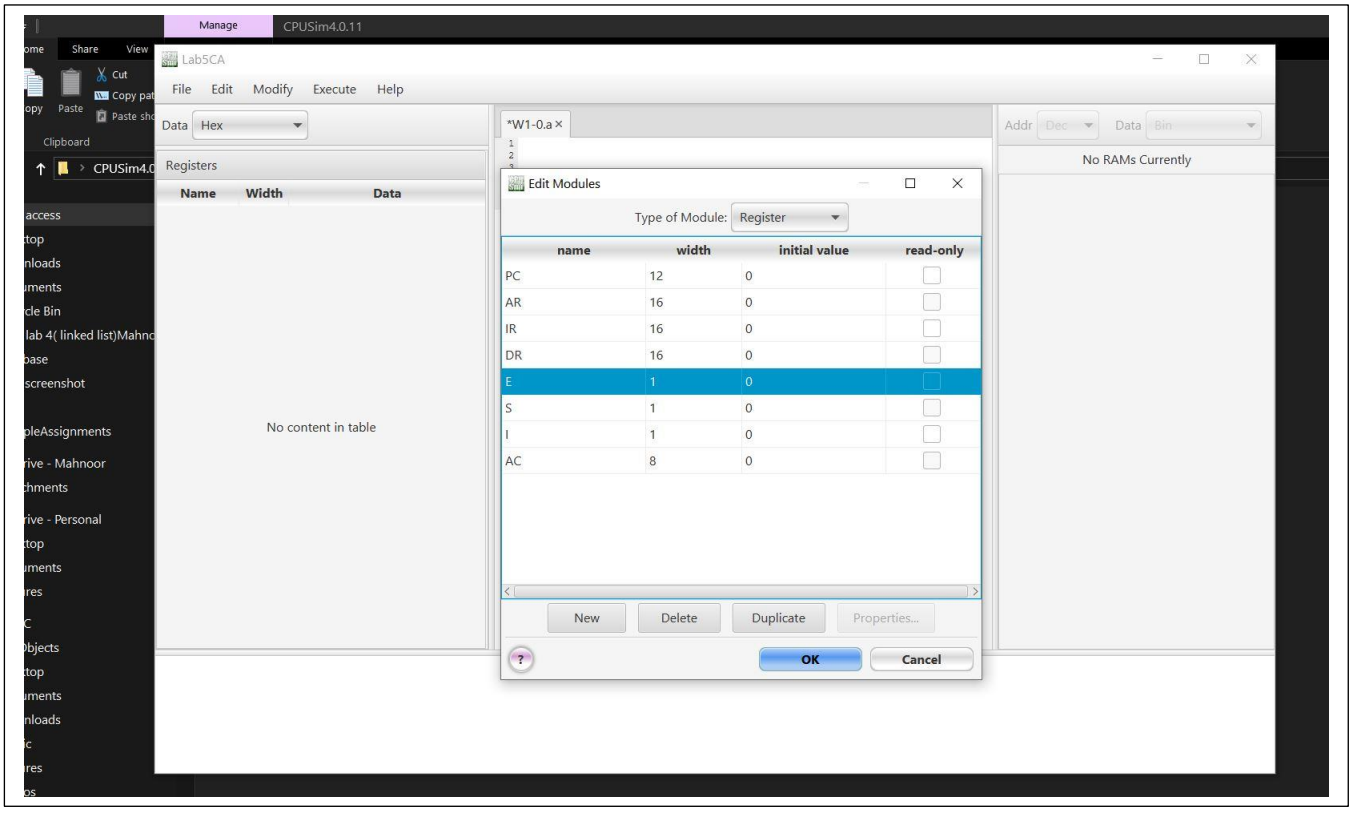
The ALU receives data from registers like the AC and DR. After processing, it sends the result back to the AC or another register for temporary storage. If needed, the CPU can move the data from the register back to memory for long-term storage.

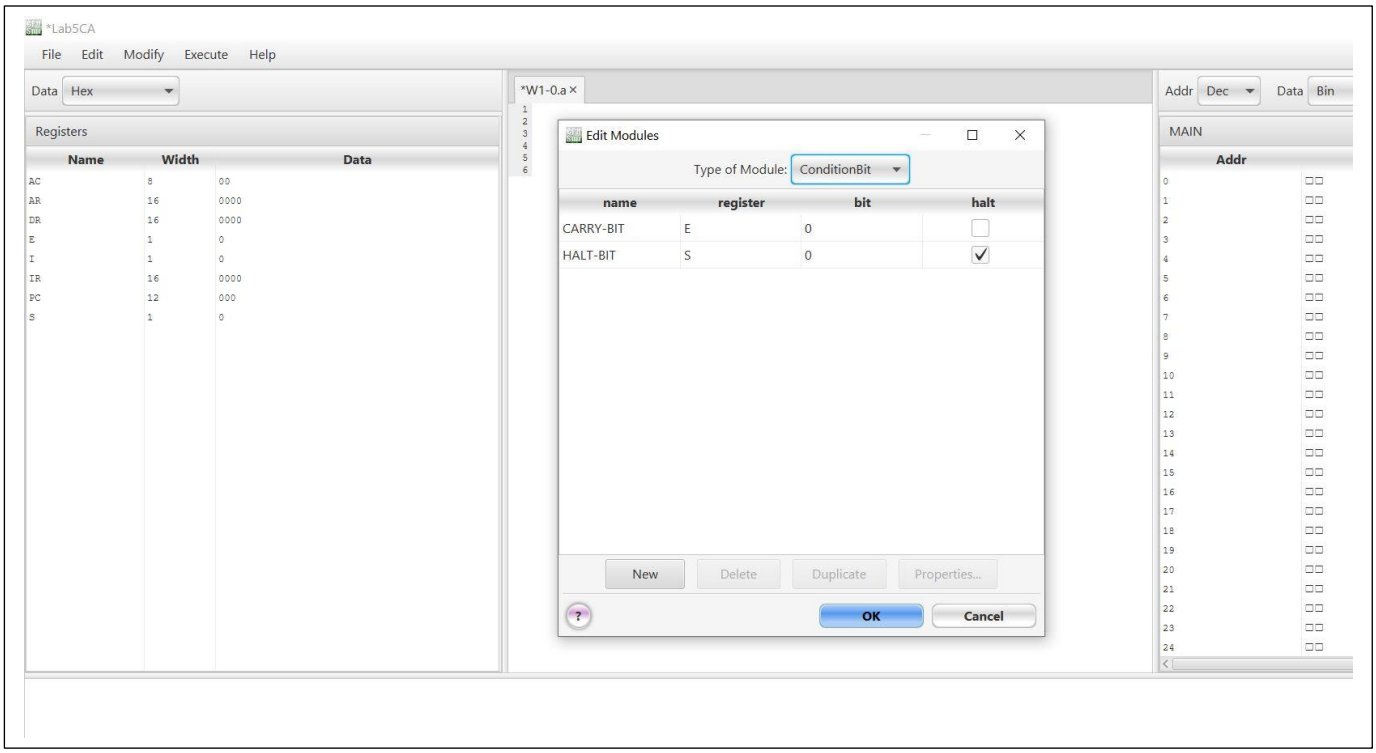
For example:

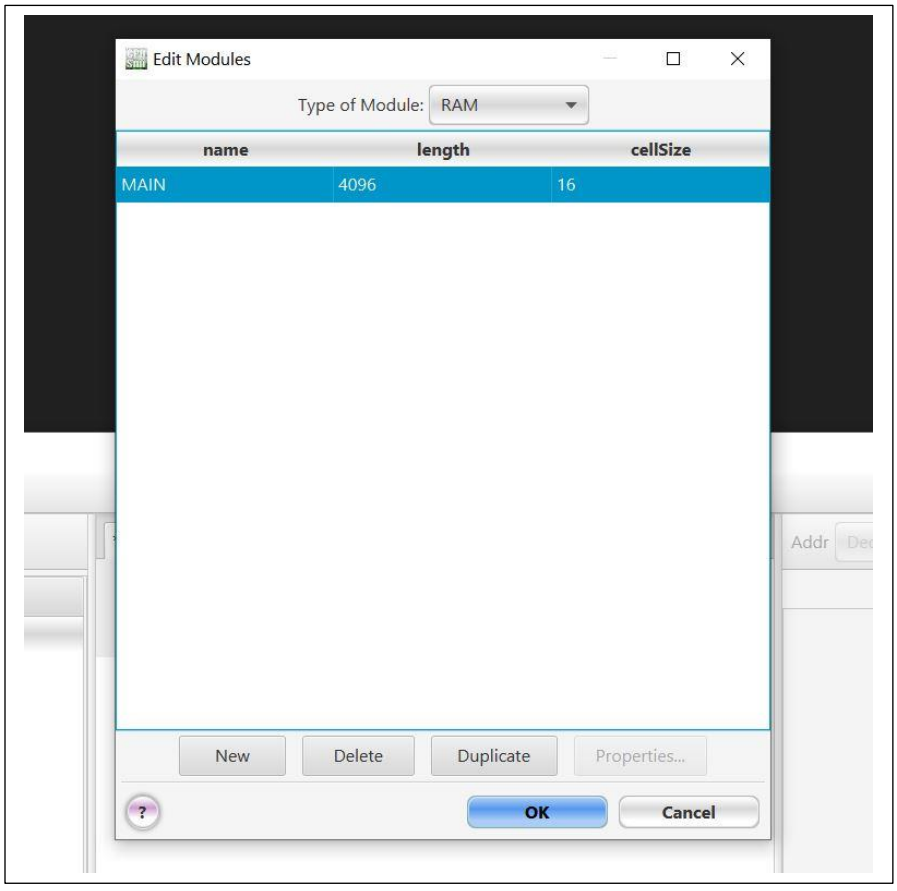
If we perform ADD 5, the ALU fetches the value from memory, processes it with the AC value, and updates the AC with the result.

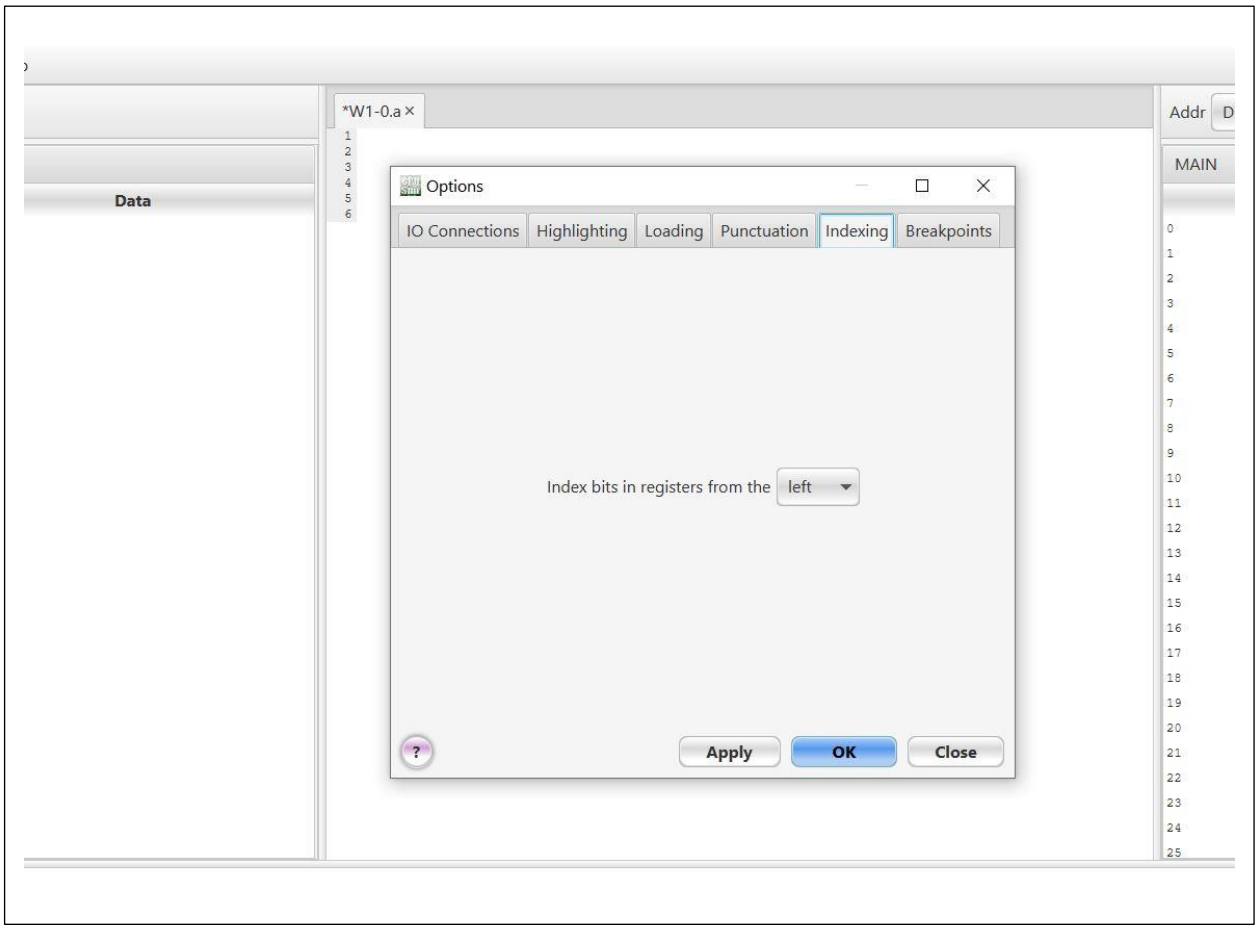
For comparisons (e.g., IF A = B), the ALU checks the values and sets status flags (like zero or carry) for conditional operations.

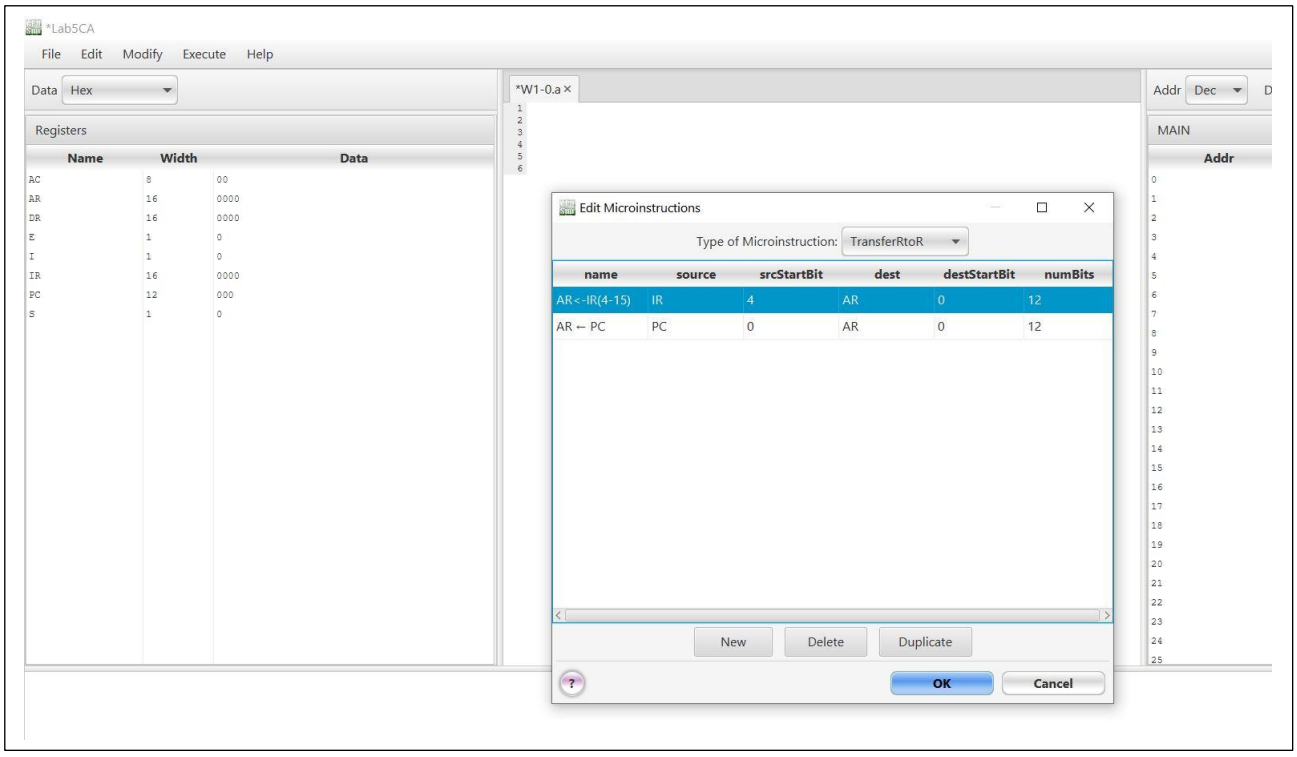
Create a new base machine and change the bit width of a register (e.g., make AC 8-bit instead of 16-bit)

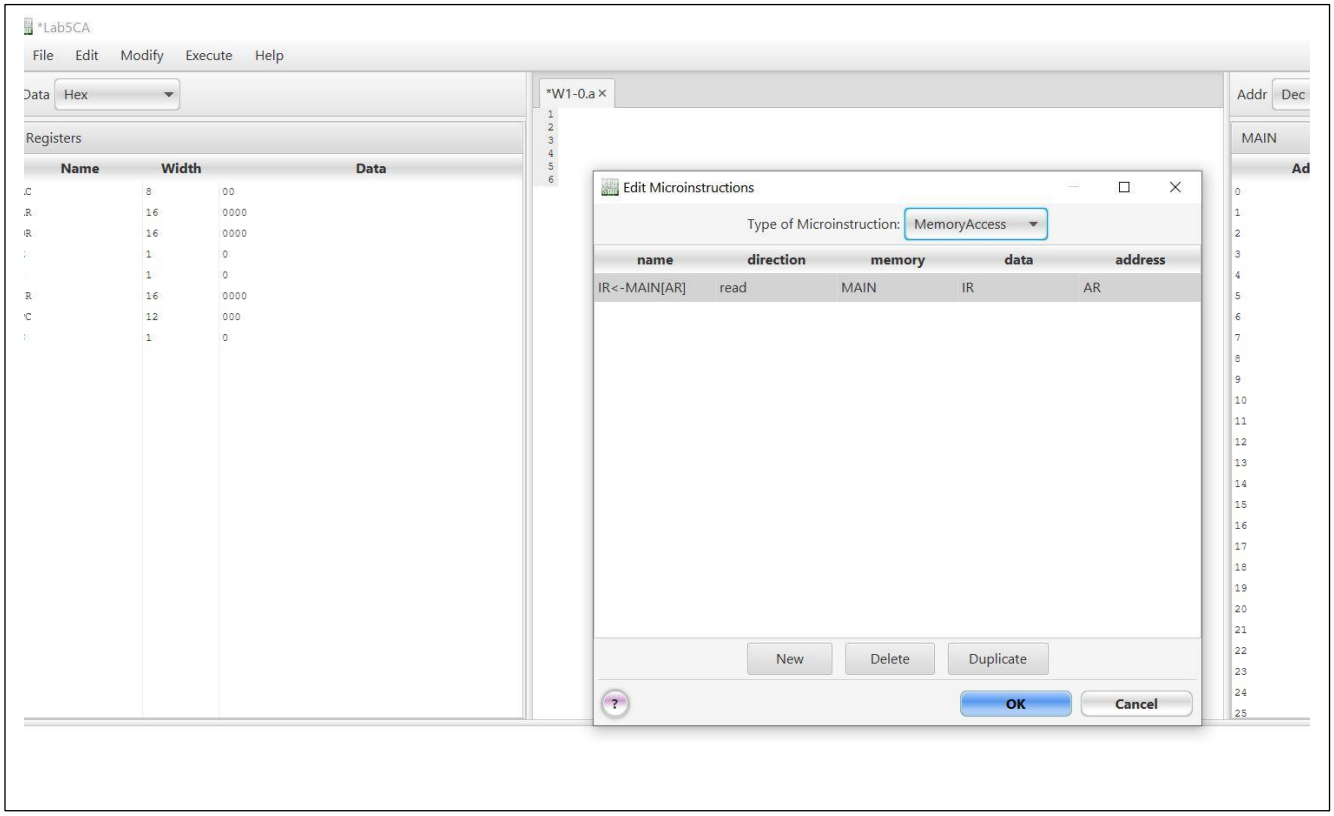


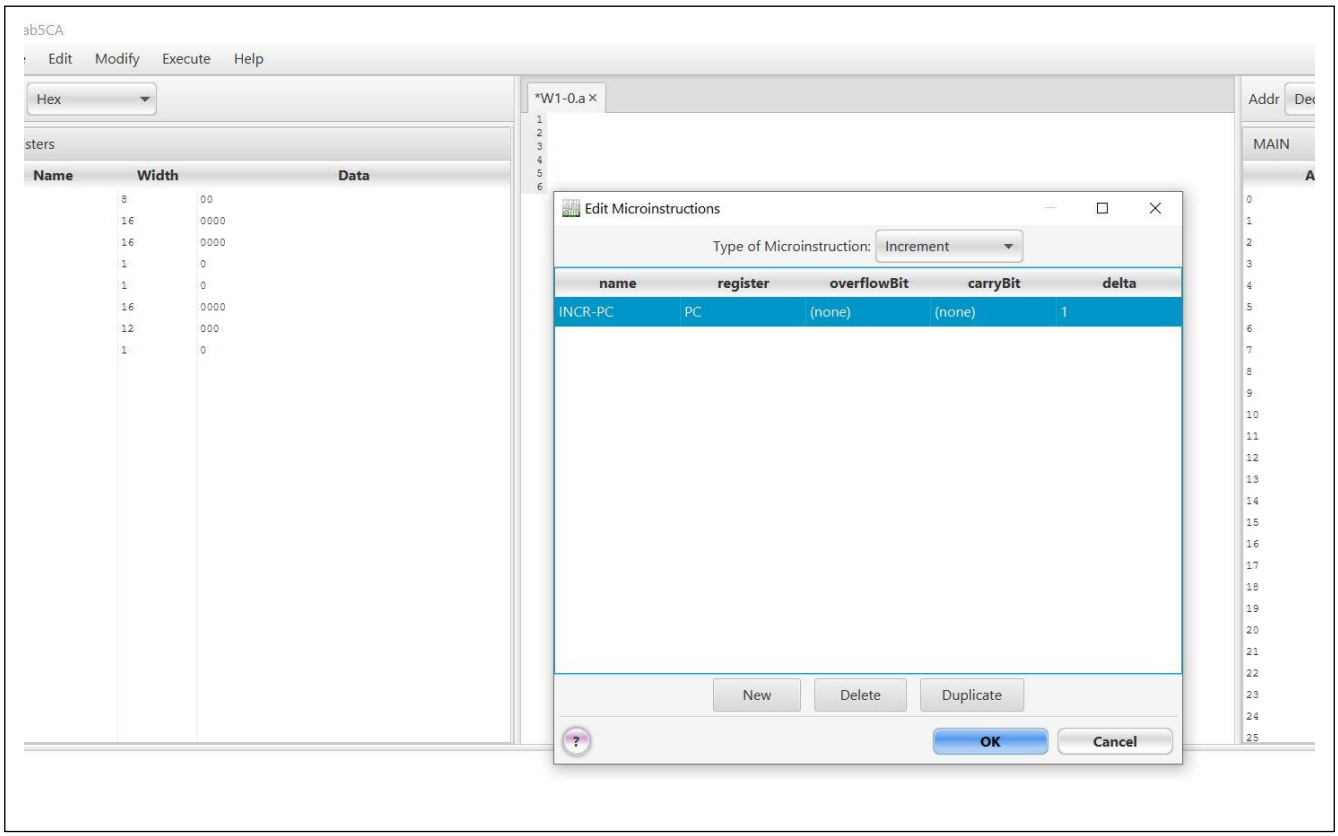

*Lab5CA
File
Edit
Modify
Execute
Help
Data
Hex
Registers
Name
Width
Data
AC
8
00
AR
16
0000
DR
16
0000
E
1
0
I
1
0
IR
16
0000
PC
12
000
S
1
0
*W1-0.a
Edit Modules
Type of Module:
ConditionBit
name
register
bit
halt
CARRY-BIT
E
0
HALT-BIT
S
0
New
Delete
Duplicate
Properties...
OK
Cancel
Addr
Dec
Data
Bin
MAIN
Addr

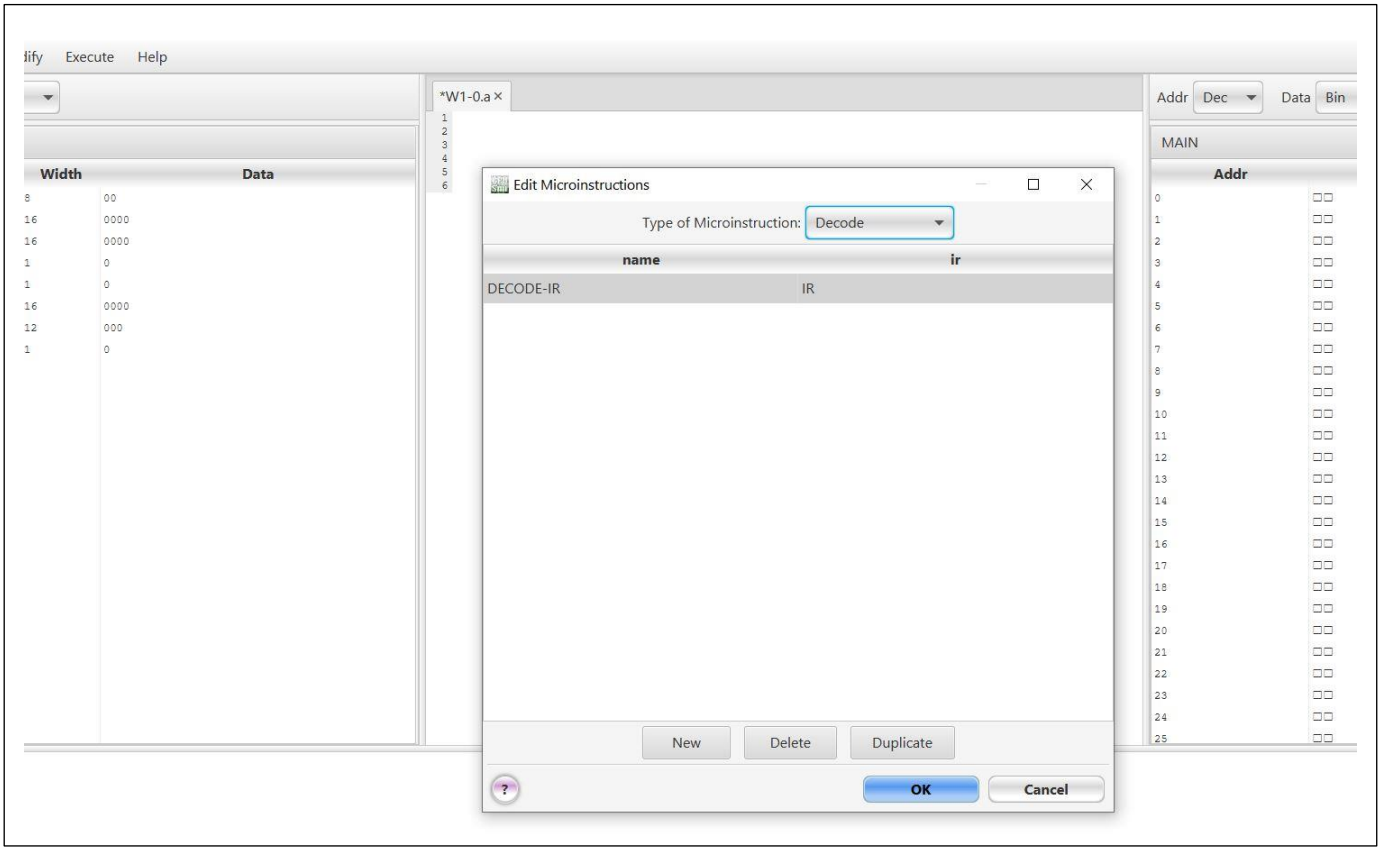

Edit Modules
Type of Module:
RAM
name
length
cellSize
MAIN
4096
16
New
Delete
Duplicate
Properties...
OK
Cancel

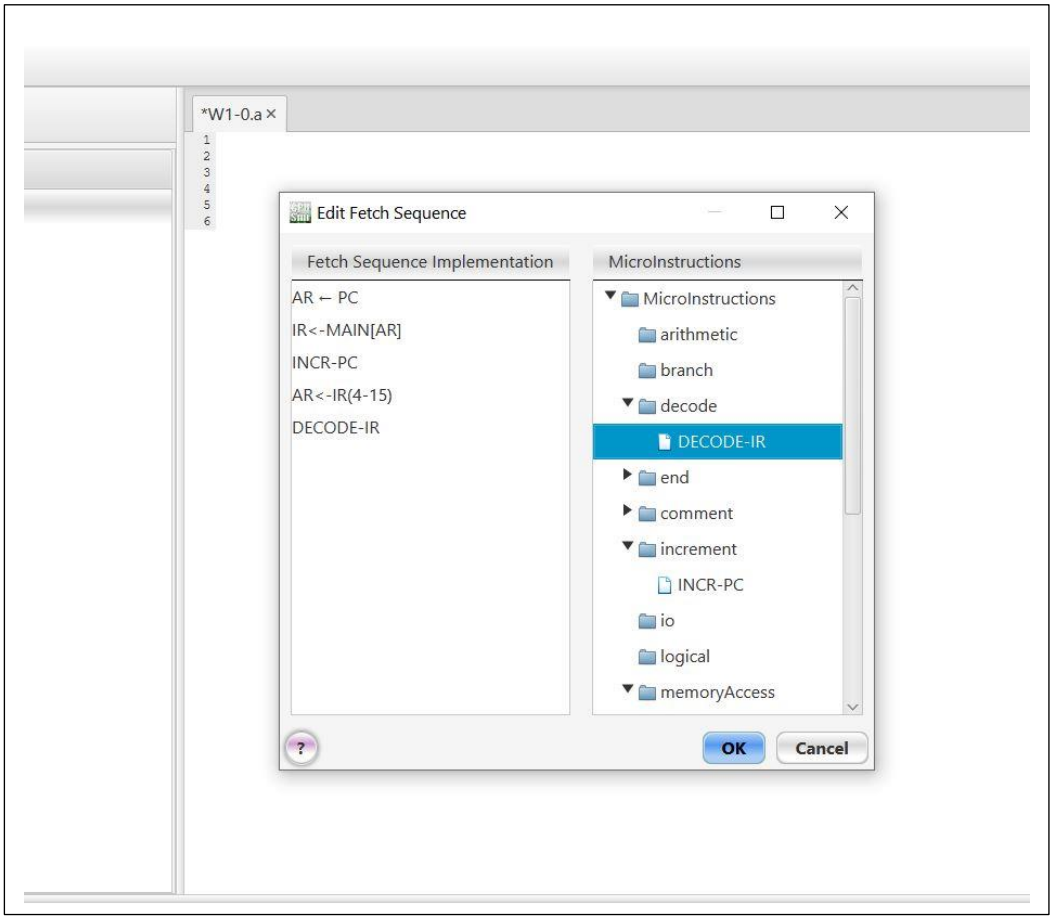

*W1-0.a
Data
Addr
MAIN
Options
IO Connections
Highlighting
Loading
Punctuation
Indexing
Breakpoints
Index bits in registers from the
left
Apply
OK
Close


*Lab5CA
File
Edit
Modify
Execute
Help
Data
Hex
Registers
Name
Width
Data
AC 8 00
AR 16 0000
DR 16 0000
E 1 0
I 1 0
IR 16 0000
PC 12 000
S 1 0
*W1-0.a
Addr
Dec
MAIN
Addr
Edit Microinstructions
Type of Microinstruction:
TransferRtoR
name
source
srcStartBit
dest
destStartBit
numBits
AR<-IR(4-15)
IR
4
AR
0
12
AR ← PC
PC
0
AR
0
12
New
Delete
Duplicate
OK
Cancel


*Lab5CA
File Edit Modify Execute Help
Hex
Registers
Name Width Data
*W1-0.a
Addr Dec
MAIN
Edit Microinstructions
Type of Microinstruction: MemoryAccess
name direction memory data address
IR<-MAIN[AR] read MAIN IR AR
New Delete Duplicate
OK Cancel


Edit Microinstructions
Type of Microinstruction: Increment
name register overflowBit carryBit delta
INCR-PC PC (none) (none) 1
New Delete Duplicate
OK Cancel


lify
Execute
Help
*W1-0.a ×
Width
Data
8
00
16
0000
16
0000
1
0
1
0
16
0000
12
000
1
0
Addr
Dec
Data
Bin
MAIN
Addr
Edit Microinstructions
Type of Microinstruction:
Decode
name
ir
DECODE-IR
IR
New
Delete
Duplicate
OK
Cancel


*W1-0.a ×
Edit Fetch Sequence
Fetch Sequence Implementation
AR ← PC
IR<-MAIN[AR]
INCR-PC
AR<-IR(4-15)
DECODE-IR
MicroInstructions
MicroInstructions
arithmetic
branch
decode
DECODE-IR
end
comment
increment
INCR-PC
io
logical
memoryAccess
OK
Cancel