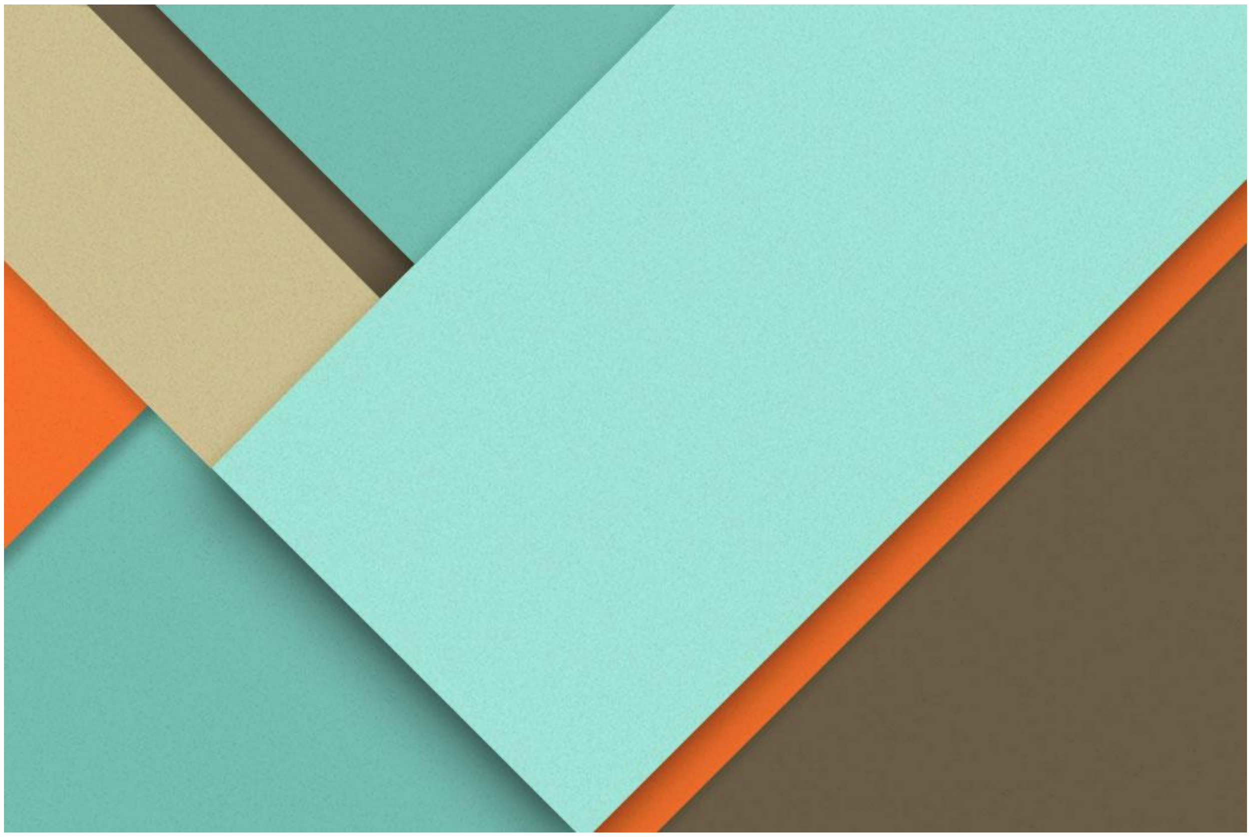


Project Report

09/03/22

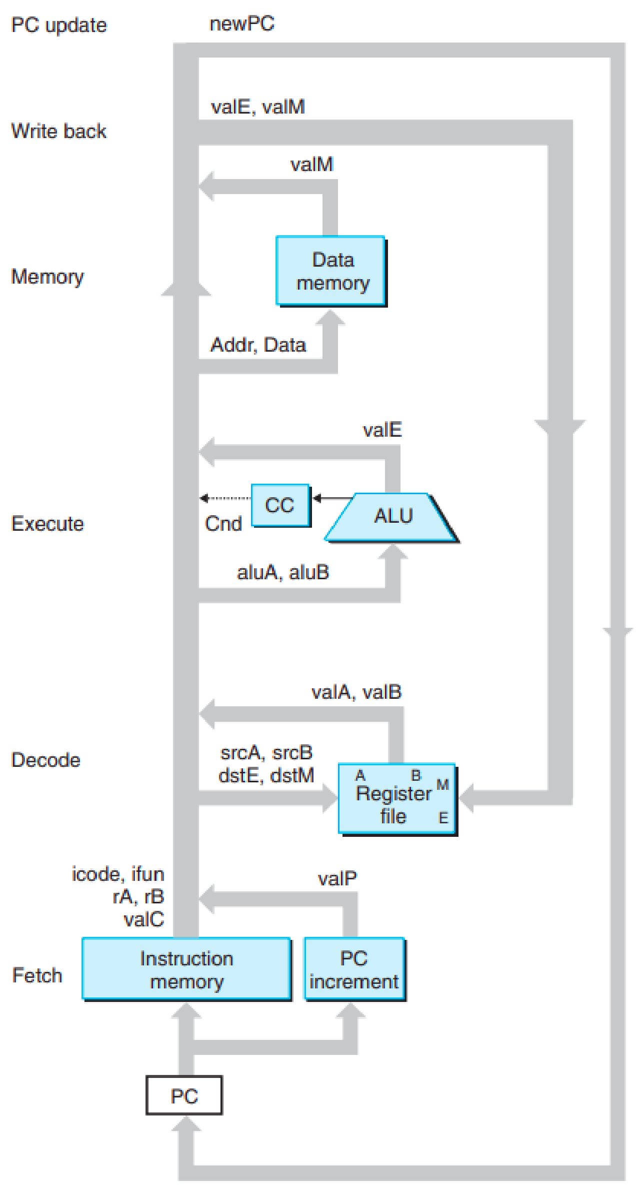
—

Srujana Vanka - 2020102005

Shreeya Singh - 2020102011

SEQ Design

To implement a Y86-64 processor, we describe a processor called SEQ (for “sequential” processor). On each clock cycle, SEQ performs all the steps required to process a complete instruction. This would require a very long cycle time, however, and so the clock rate would be unacceptably low. Our purpose in developing SEQ is to provide a first step toward our ultimate goal of implementing an efficient pipelined processor.

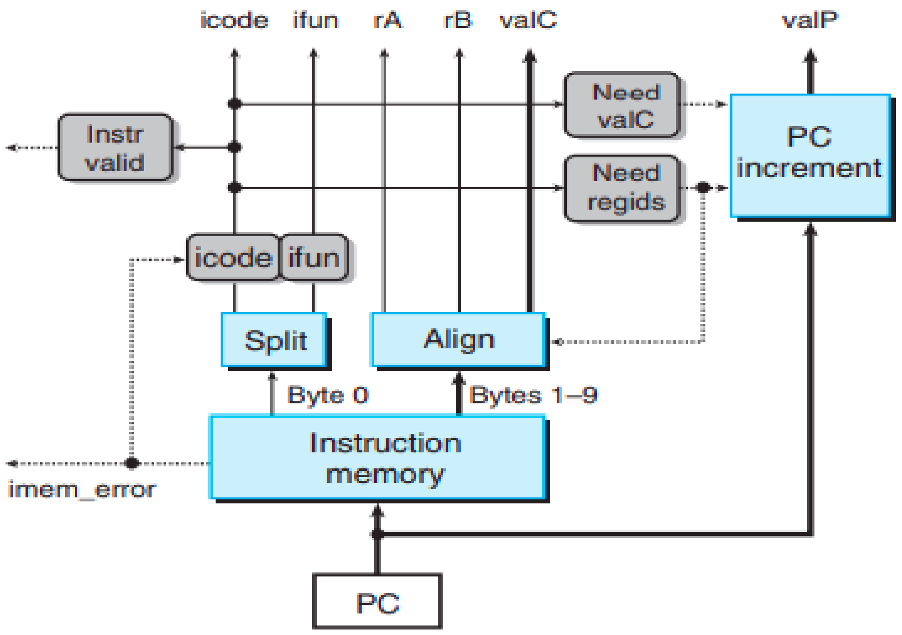


The processor architecture can be broadly divided into 5 main modules described below:

- Fetch
- Decode and Writeback
- Execute
- Memory
- PC update

Fetch module

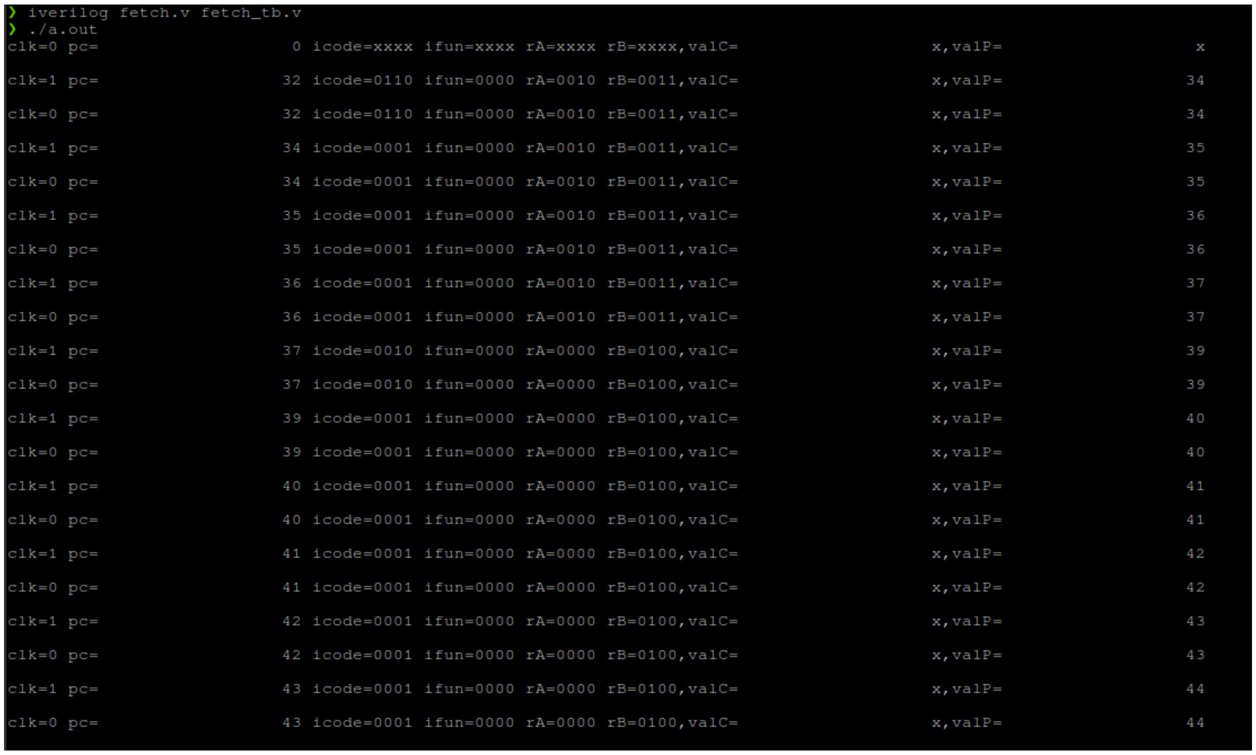
SEQ Fetch stage:



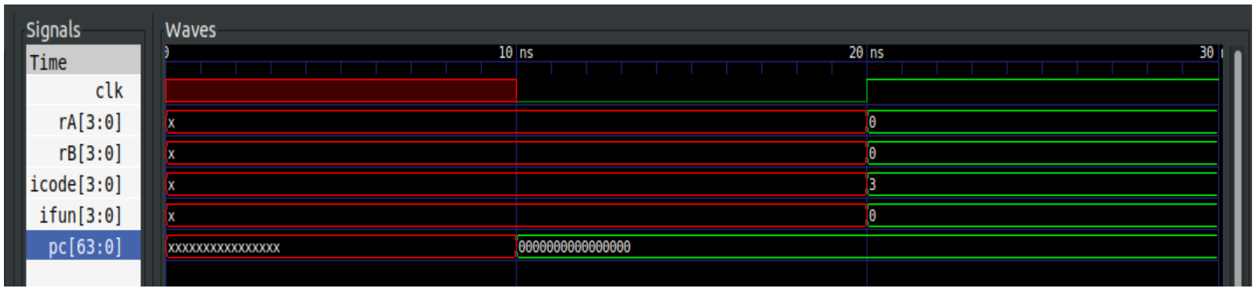
- The fetch stage reads the bytes of an instruction from memory, using the program counter (PC) as the memory address.
- Now, when this step ends, the PC points to the next instruction to be read in the next cycle and the cycle continues like this to fetch all the instructions.
- From the instruction it extracts the two 4-bit portions of the instruction specifier byte, referred to as icode (the instruction code) and ifun (the instruction function).

- It possibly fetches a register specifier byte, giving one or both of the register operand specifiers rA and rB. It also possibly fetches an 8-byte constant word valC. It computes valP to be the address of the instruction following the current one in sequential order. That is, valP equals the value of the PC plus the length of the fetched instruction.The instruction memory reads the bytes of an instruction using the program counter register as an address.

Results



```
❯ iverilog fetch.v fetch_tb.v
❯ ./a.out
clk=0 pc=                    0 icode=xxxx ifun=xxxx rA=xxxx rB=xxxx,valC=                  x,valP=                   x

clk=1 pc=                   32 icode=0110 ifun=0000 rA=0010 rB=0011,valC=                  x,valP=                  34

clk=0 pc=                   32 icode=0110 ifun=0000 rA=0010 rB=0011,valC=                  x,valP=                  34

clk=1 pc=                   34 icode=0001 ifun=0000 rA=0010 rB=0011,valC=                  x,valP=                  35

clk=0 pc=                   34 icode=0001 ifun=0000 rA=0010 rB=0011,valC=                  x,valP=                  35

clk=1 pc=                   35 icode=0001 ifun=0000 rA=0010 rB=0011,valC=                  x,valP=                  36

clk=0 pc=                   35 icode=0001 ifun=0000 rA=0010 rB=0011,valC=                  x,valP=                  36

clk=1 pc=                   36 icode=0001 ifun=0000 rA=0010 rB=0011,valC=                  x,valP=                  37

clk=0 pc=                   36 icode=0001 ifun=0000 rA=0010 rB=0011,valC=                  x,valP=                  37

clk=1 pc=                   37 icode=0010 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  39

clk=0 pc=                   37 icode=0010 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  39

clk=1 pc=                   39 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  40

clk=0 pc=                   39 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  40

clk=1 pc=                   40 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  41

clk=0 pc=                   40 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  41

clk=1 pc=                   41 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  42

clk=0 pc=                   41 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  42

clk=1 pc=                   42 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  43

clk=0 pc=                   42 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  43

clk=1 pc=                   43 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  44

clk=0 pc=                   43 icode=0001 ifun=0000 rA=0000 rB=0100,valC=                  x,valP=                  44
```

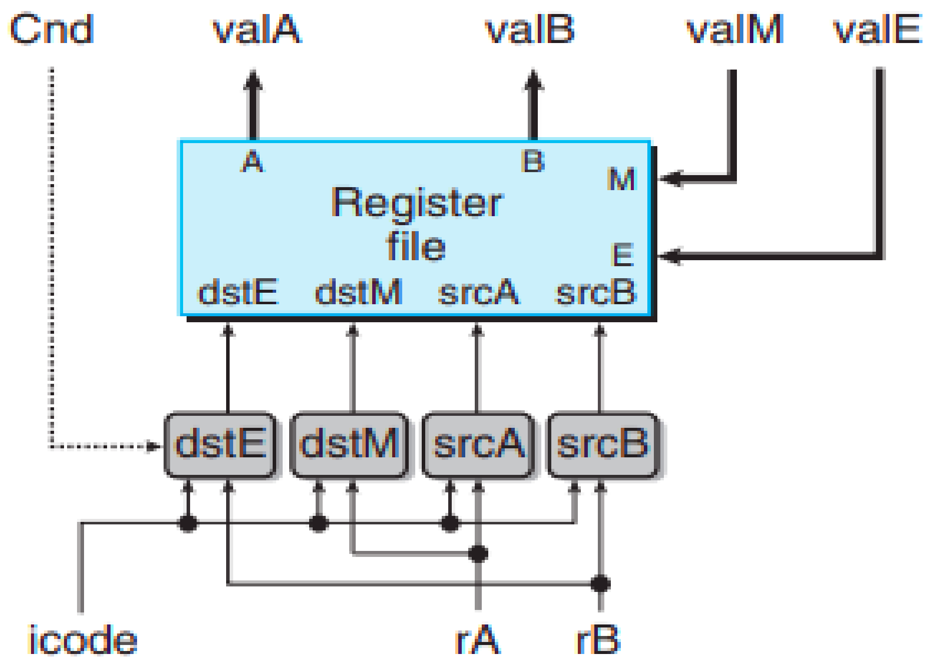


Decode and Writeback module

- The decode module is used to interpret the encoded instruction that is provided in the instruction register at a given clock cycle.

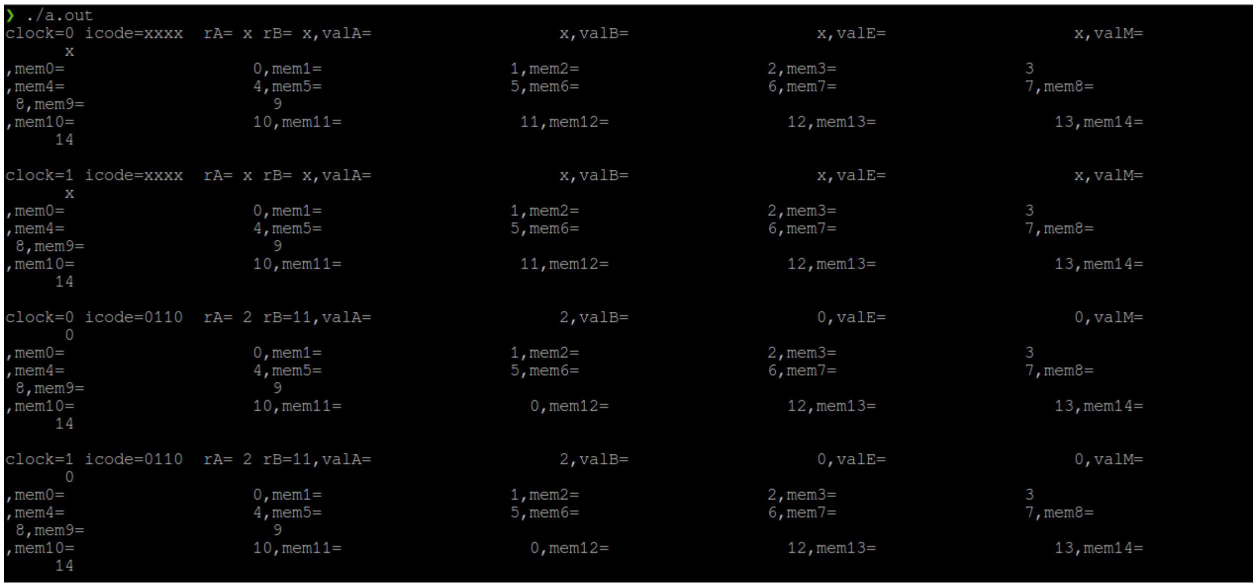
-> For pipeline we implement Decode and Writeback stages together

SEQ Decode and Writeback stage:

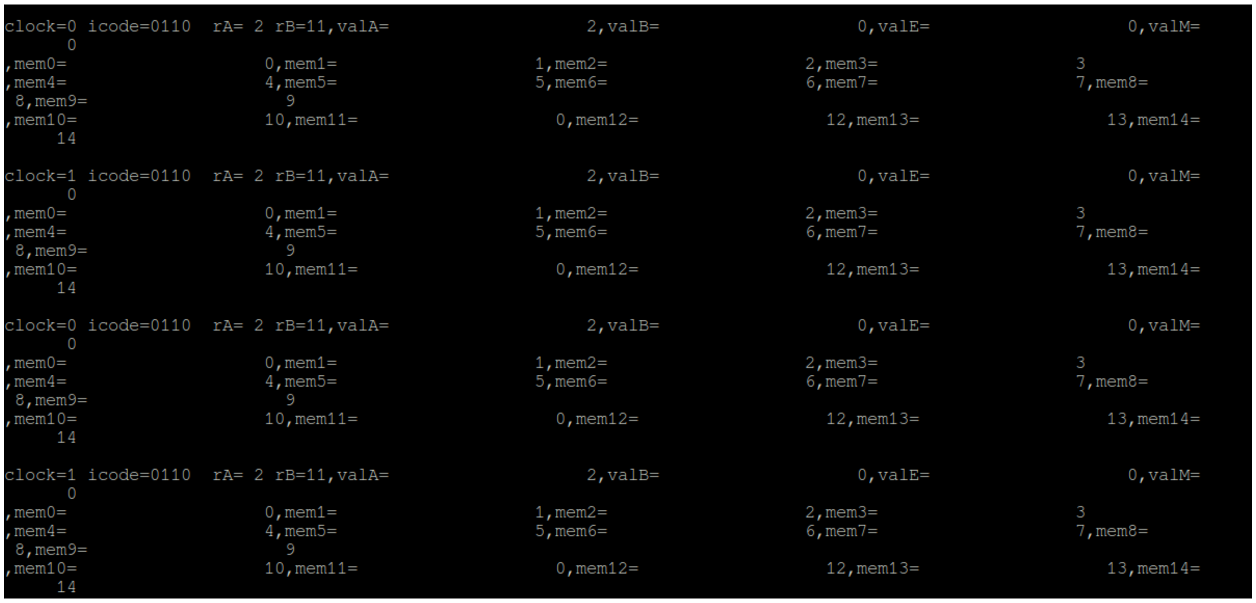


- These two stages (decode and write-back) are combined because they both access the register file.
- The register file has two read ports, A and B, via which register values valA and valB are read simultaneously.
- The instruction fields are decoded to generate register identifiers for four addresses (two read and two write) used by the register file.

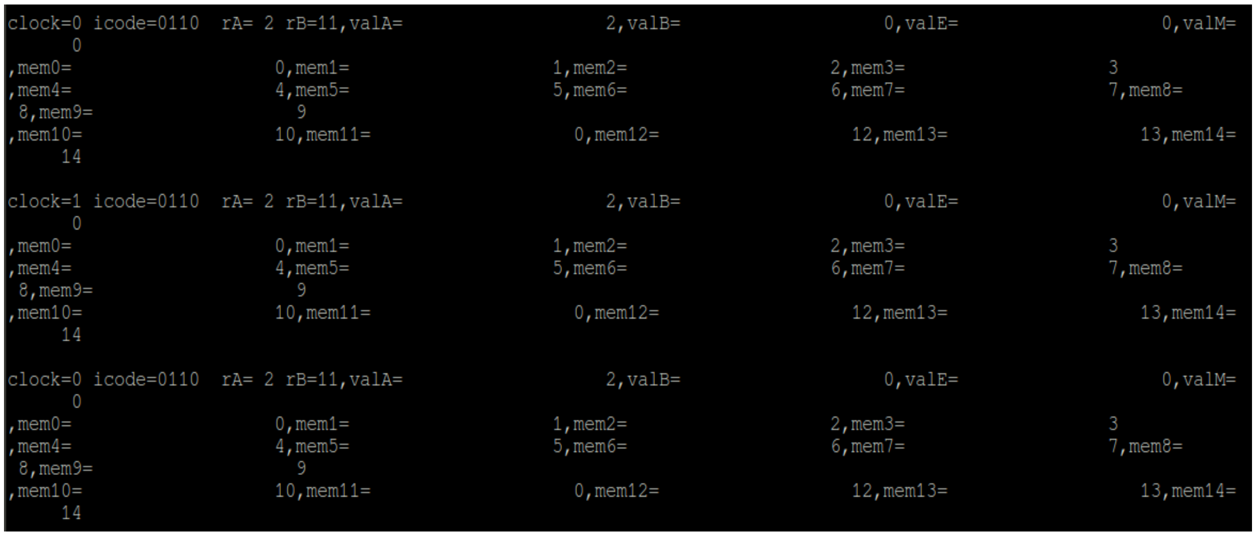
Results



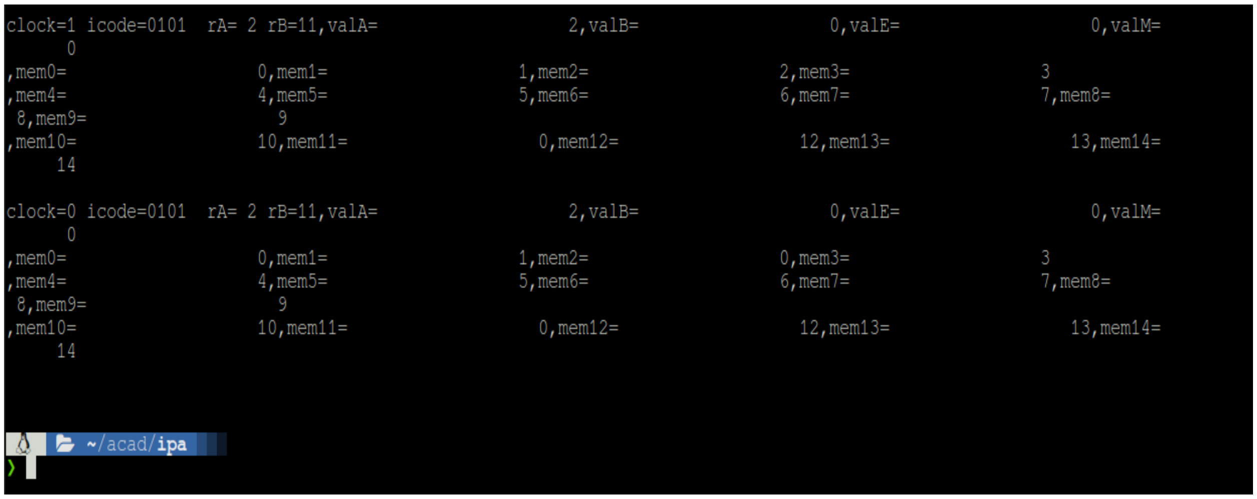
```
❯ ./a.out
clock=0 icode=xxxx  rA= x rB= x,valA=                   x,valB=                   x,valE=                   x,valM=
      x
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                  11,mem12=                  12,mem13=                  13,mem14=
     14

clock=1 icode=xxxx  rA= x rB= x,valA=                   x,valB=                   x,valE=                   x,valM=
      x
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                  11,mem12=                  12,mem13=                  13,mem14=
     14

clock=0 icode=0110  rA= 2 rB=11,valA=                   2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                   0,mem12=                  12,mem13=                  13,mem14=
     14

clock=1 icode=0110  rA= 2 rB=11,valA=                   2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                   0,mem12=                  12,mem13=                  13,mem14=
     14
```



```
clock=0 icode=0110  rA= 2 rB=11,valA=                   2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                   0,mem12=                  12,mem13=                  13,mem14=
     14

clock=1 icode=0110  rA= 2 rB=11,valA=                   2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                   0,mem12=                  12,mem13=                  13,mem14=
     14

clock=0 icode=0110  rA= 2 rB=11,valA=                   2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                   0,mem12=                  12,mem13=                  13,mem14=
     14

clock=1 icode=0110  rA= 2 rB=11,valA=                   2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                   0,mem12=                  12,mem13=                  13,mem14=
     14
```



```
clock=0 icode=0110  rA= 2 rB=11,valA=                  2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                  0,mem12=                  12,mem13=                  13,mem14=
     14

clock=1 icode=0110  rA= 2 rB=11,valA=                  2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                  0,mem12=                  12,mem13=                  13,mem14=
     14

clock=0 icode=0110  rA= 2 rB=11,valA=                  2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                  0,mem12=                  12,mem13=                  13,mem14=
     14
```

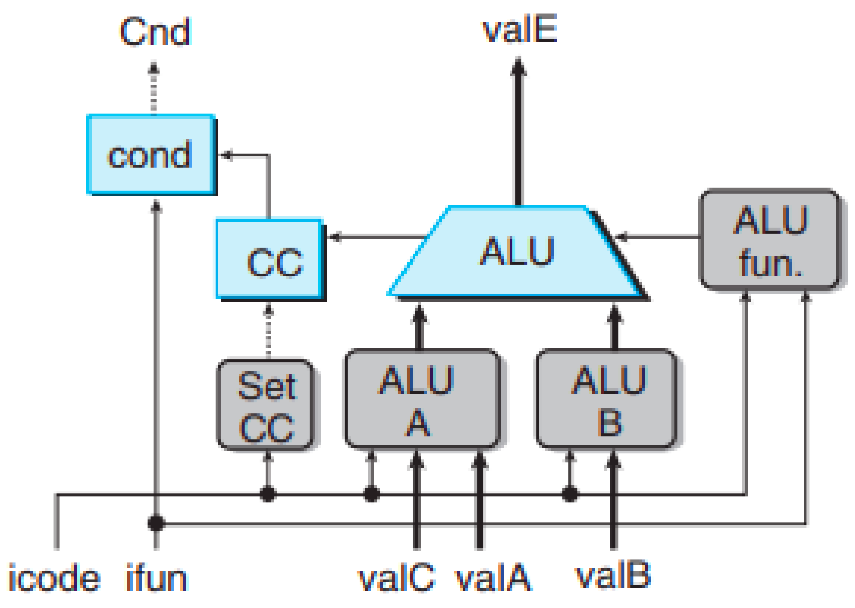


```
clock=1 icode=0101  rA= 2 rB=11,valA=                  2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   2,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                  0,mem12=                  12,mem13=                  13,mem14=
     14

clock=0 icode=0101  rA= 2 rB=11,valA=                  2,valB=                   0,valE=                   0,valM=
      0
,mem0=                   0,mem1=                   1,mem2=                   0,mem3=                   3
,mem4=                   4,mem5=                   5,mem6=                   6,mem7=                   7,mem8=
 8,mem9=                   9
,mem10=                  10,mem11=                  0,mem12=                  12,mem13=                  13,mem14=
     14


  ~/acad/ipa
❯
```

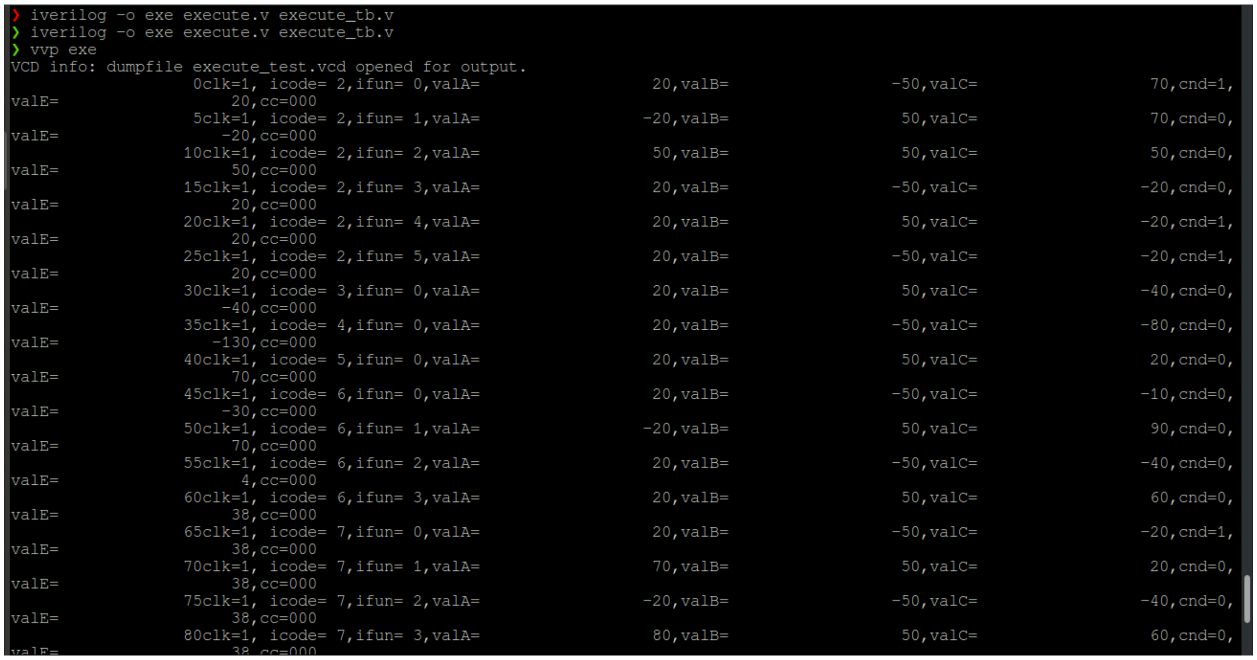
Execute module

- The execute module carries out the functional operations which are implemented by ALU for different purposes according to the instruction type.
- The decoded data is sent as control signals to the functional units and then the required instructions are carried out before being transferred to the ALU.
- The ALU output becomes the signal valE.

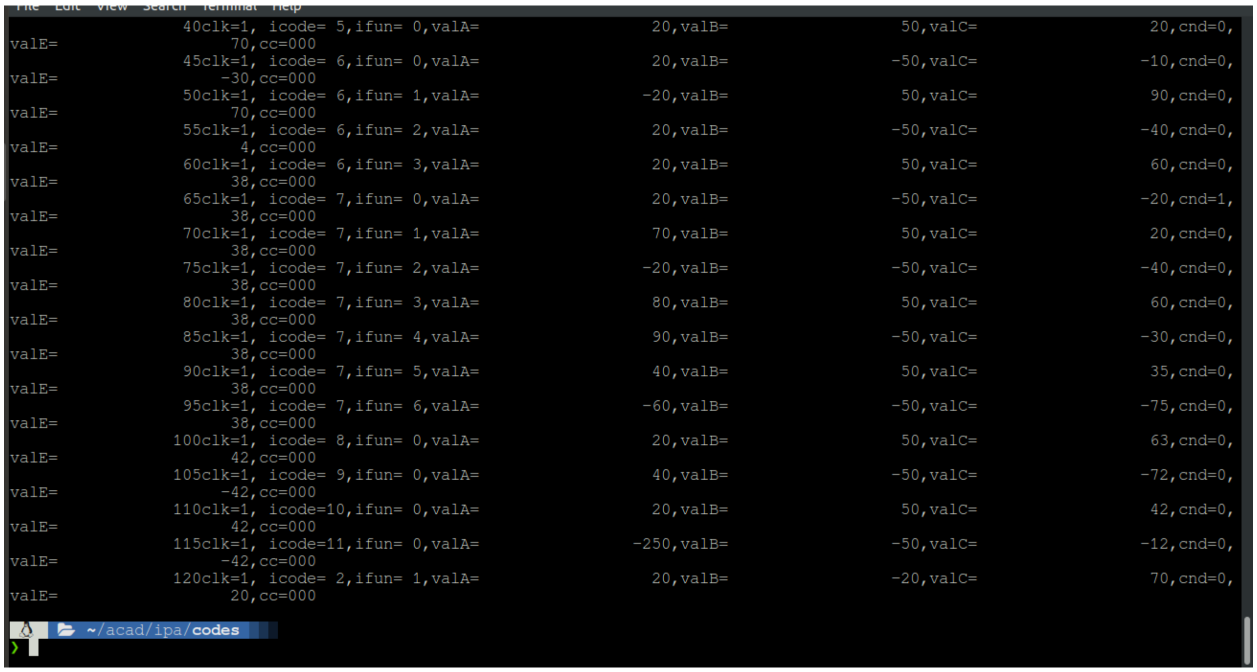
SEQ Execute stage:



Results



```
❯ iverilog -o exe execute.v execute_tb.v
❯ iverilog -o exe execute.v execute_tb.v
❯ vvp exe
VCD info: dumpfile execute_test.vcd opened for output.
                   0clk=1, icode= 2,ifun= 0,valA=               20,valB=                -50,valC=                  70,cnd=1,
valE=                   20,cc=000
                   5clk=1, icode= 2,ifun= 1,valA=              -20,valB=                 50,valC=                  70,cnd=0,
valE=                  -20,cc=000
                  10clk=1, icode= 2,ifun= 2,valA=               50,valB=                 50,valC=                  50,cnd=0,
valE=                   50,cc=000
                  15clk=1, icode= 2,ifun= 3,valA=               20,valB=                -50,valC=                 -20,cnd=0,
valE=                   20,cc=000
                  20clk=1, icode= 2,ifun= 4,valA=               20,valB=                 50,valC=                 -20,cnd=1,
valE=                   20,cc=000
                  25clk=1, icode= 2,ifun= 5,valA=               20,valB=                -50,valC=                 -20,cnd=1,
valE=                   20,cc=000
                  30clk=1, icode= 3,ifun= 0,valA=               20,valB=                 50,valC=                 -40,cnd=0,
valE=                  -40,cc=000
                  35clk=1, icode= 4,ifun= 0,valA=               20,valB=                -50,valC=                 -80,cnd=0,
valE=                 -130,cc=000
                  40clk=1, icode= 5,ifun= 0,valA=               20,valB=                 50,valC=                  20,cnd=0,
valE=                   70,cc=000
                  45clk=1, icode= 6,ifun= 0,valA=               20,valB=                -50,valC=                 -10,cnd=0,
valE=                  -30,cc=000
                  50clk=1, icode= 6,ifun= 1,valA=              -20,valB=                 50,valC=                  90,cnd=0,
valE=                   70,cc=000
                  55clk=1, icode= 6,ifun= 2,valA=               20,valB=                -50,valC=                 -40,cnd=0,
valE=                    4,cc=000
                  60clk=1, icode= 6,ifun= 3,valA=               20,valB=                 50,valC=                  60,cnd=0,
valE=                   38,cc=000
                  65clk=1, icode= 7,ifun= 0,valA=               20,valB=                -50,valC=                 -20,cnd=1,
valE=                   38,cc=000
                  70clk=1, icode= 7,ifun= 1,valA=               70,valB=                 50,valC=                  20,cnd=0,
valE=                   38,cc=000
                  75clk=1, icode= 7,ifun= 2,valA=              -20,valB=                -50,valC=                 -40,cnd=0,
valE=                   38,cc=000
                  80clk=1, icode= 7,ifun= 3,valA=               80,valB=                 50,valC=                  60,cnd=0,
valE=                   38,cc=000
```

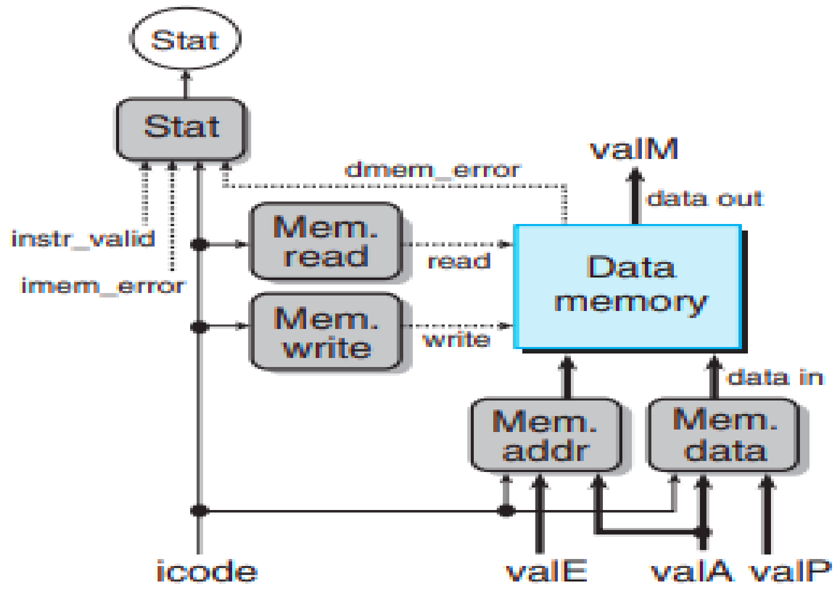


```
                  40clk=1, icode= 5,ifun= 0,valA=               20,valB=                 50,valC=                  20,cnd=0,
valE=                   70,cc=000
                  45clk=1, icode= 6,ifun= 0,valA=               20,valB=                -50,valC=                 -10,cnd=0,
valE=                  -30,cc=000
                  50clk=1, icode= 6,ifun= 1,valA=              -20,valB=                 50,valC=                  90,cnd=0,
valE=                   70,cc=000
                  55clk=1, icode= 6,ifun= 2,valA=               20,valB=                -50,valC=                 -40,cnd=0,
valE=                    4,cc=000
                  60clk=1, icode= 6,ifun= 3,valA=               20,valB=                 50,valC=                  60,cnd=0,
valE=                   38,cc=000
                  65clk=1, icode= 7,ifun= 0,valA=               20,valB=                -50,valC=                 -20,cnd=1,
valE=                   38,cc=000
                  70clk=1, icode= 7,ifun= 1,valA=               70,valB=                 50,valC=                  20,cnd=0,
valE=                   38,cc=000
                  75clk=1, icode= 7,ifun= 2,valA=              -20,valB=                -50,valC=                 -40,cnd=0,
valE=                   38,cc=000
                  80clk=1, icode= 7,ifun= 3,valA=               80,valB=                 50,valC=                  60,cnd=0,
valE=                   38,cc=000
                  85clk=1, icode= 7,ifun= 4,valA=               90,valB=                -50,valC=                 -30,cnd=0,
valE=                   38,cc=000
                  90clk=1, icode= 7,ifun= 5,valA=               40,valB=                 50,valC=                  35,cnd=0,
valE=                   38,cc=000
                  95clk=1, icode= 7,ifun= 6,valA=              -60,valB=                -50,valC=                 -75,cnd=0,
valE=                   38,cc=000
                 100clk=1, icode= 8,ifun= 0,valA=               20,valB=                 50,valC=                  63,cnd=0,
valE=                   42,cc=000
                 105clk=1, icode= 9,ifun= 0,valA=               40,valB=                -50,valC=                 -72,cnd=0,
valE=                  -42,cc=000
                 110clk=1, icode=10,ifun= 0,valA=               20,valB=                 50,valC=                  42,cnd=0,
valE=                   42,cc=000
                 115clk=1, icode=11,ifun= 0,valA=             -250,valB=                -50,valC=                 -12,cnd=0,
valE=                  -42,cc=000
                 120clk=1, icode= 2,ifun= 1,valA=               20,valB=                -20,valC=                  70,cnd=0,
valE=                   20,cc=000

 ~/acad/ipa/codes
❯
```

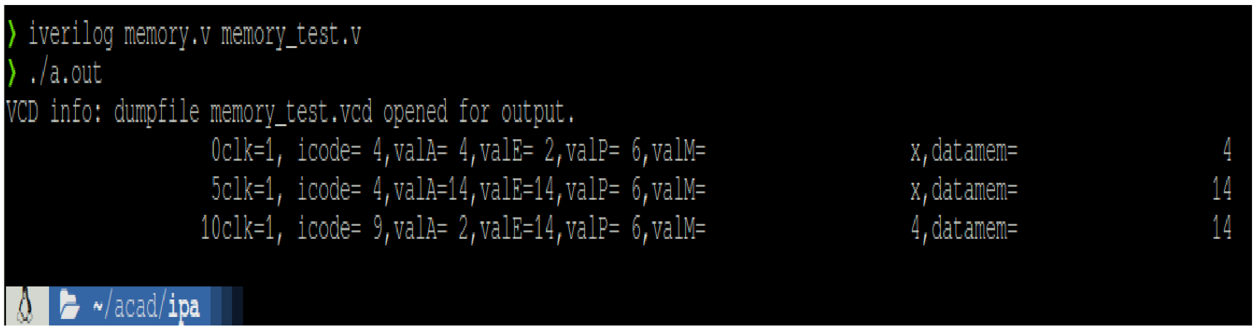
Memory module

- This is the final module used while developing this processor design.
- The memory stage has the task of either reading or writing program data.
- Two control blocks generate the values for the memory.
- Two other blocks generate the control signals indicating whether to perform a read or a write operation

SEQ memory stage:



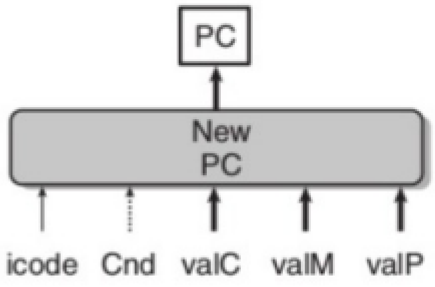
Results



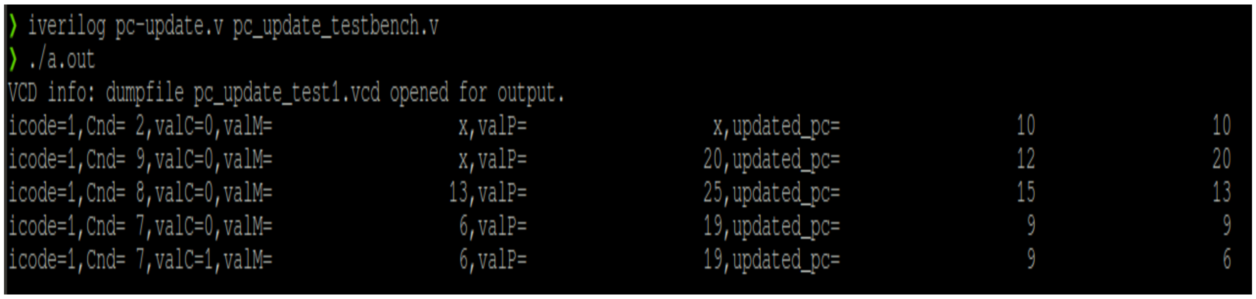
```
❯ iverilog memory.v memory_test.v
❯ ./a.out
VCD info: dumpfile memory_test.vcd opened for output.
                   0clk=1, icode= 4,valA= 4,valE= 2,valP= 6,valM=                    x,datamem=                   4
                   5clk=1, icode= 4,valA=14,valE=14,valP= 6,valM=                    x,datamem=                  14
                  10clk=1, icode= 9,valA= 2,valE=14,valP= 6,valM=                    4,datamem=                  14

~/acad/ipa
```

PC update

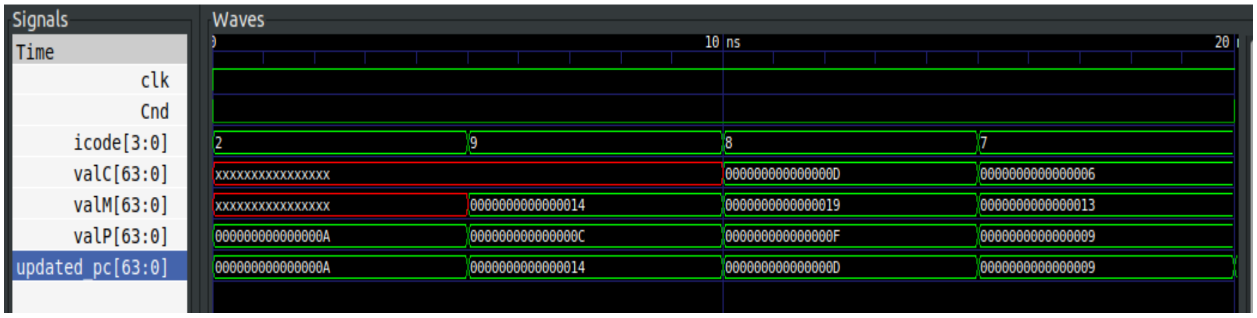
- Inputs to this module are clk,icode,cnd,valC,valM,valP.
- This module outputs a PC updated signal.
- Depending on the type of instruction the program counter value is updated using valC, valM, valP and it is shown at the output.
- For some instructions like jump it will check with cnd signal and then depending on the signal it updates the program counter.



Results

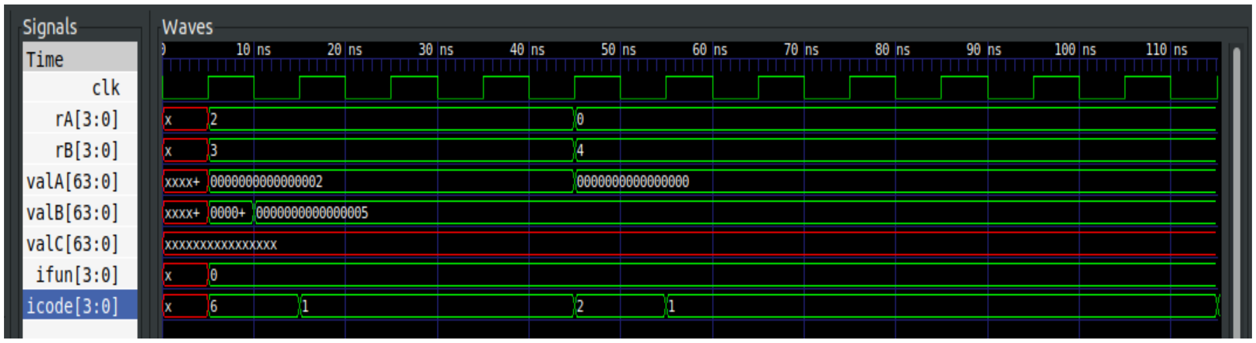


```
❯ iverilog pc-update.v pc_update_testbench.v
❯ ./a.out
VCD info: dumpfile pc_update_test1.vcd opened for output.
icode=1,Cnd= 2,valC=0,valM=                    x,valP=                    x,updated_pc=                   10                   10
icode=1,Cnd= 9,valC=0,valM=                    x,valP=                   20,updated_pc=                   12                   20
icode=1,Cnd= 8,valC=0,valM=                   13,valP=                   25,updated_pc=                   15                   13
icode=1,Cnd= 7,valC=0,valM=                    6,valP=                   19,updated_pc=                    9                    9
icode=1,Cnd= 7,valC=1,valM=                    6,valP=                   19,updated_pc=                    9                    6
```



Sequential Processor Stage:

Testbench results:



Processor Architecture with 5 Stage Pipelined based on the structure of Y86 Processor

To develop a processor architecture design based on the Y86 ISA using Verilog.

As a transitional step toward a pipelined design, we slightly rearrange the order of the five stages in SEQ so that the PC update stage comes at the beginning of the clock cycle, rather than at the end.

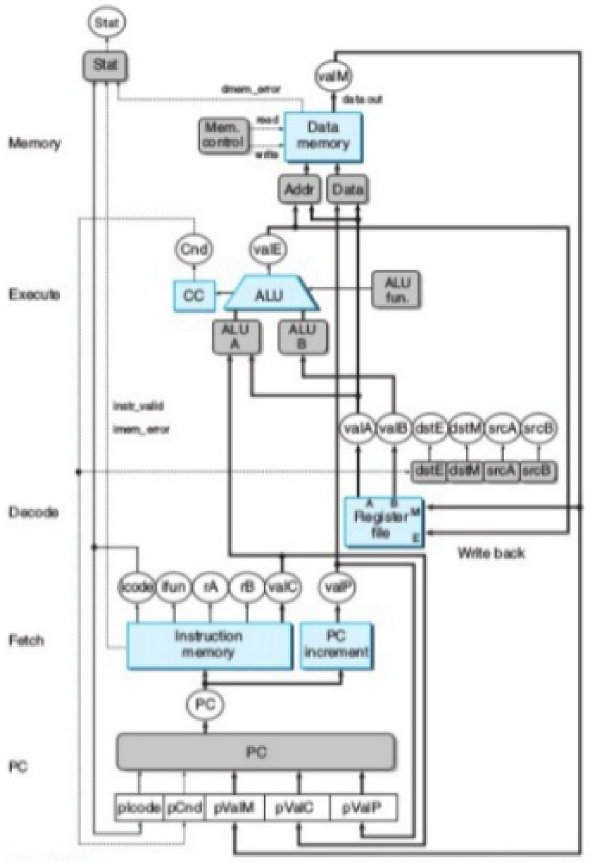


Figure 4.40 **SEQ+ hardware structure.** Shifting the PC computation from the end of the clock cycle to the beginning makes it more suitable for pipelining.

The pipeline registers are labeled as follows:

● F holds a predicted value of the program counter, as will be discussed shortly.

● D sits between the fetch and decode stages. It holds information about the most recently fetched instruction for processing by the decode stage.

● E sits between the decode and execute stages. It holds information about the most recently decoded instruction and the values read from the register file for processing by the execute stage.

● M sits between the execute and memory stages. It holds the results of the most recently executed instruction for processing by the memory stage. It also holds information about branch conditions and branch targets for processing conditional jumps.

● W sits between the memory stage and the feedback paths that supply the computed results to the register file for writing and the return address to the PC selection logic when completing a ret instruction

Our sequential implementations SEQ only process one instruction at a time, and so there are unique values for signals such as valC, srcA, and valE. In our pipelined design, there will be multiple versions of these values associated with the different instructions flowing through the system. I have adopted a naming scheme where a signal stored in a pipeline register can be uniquely identified by prefixing its name with that of the pipe register written in uppercase. For example, the four status codes are named D_stat, E_stat, M_stat, and W_stat.

Fetch module

- The fetch stage reads the bytes of an instruction from memory, using the program counter (PC) as the memory address.
- Now, when this step ends, the PC points to the next instruction to be read in the next cycle and the cycle continues like this to fetch all the instructions.
- From the instruction it extracts the two 4-bit portions of the instruction specifier byte, referred to as icode (the instruction code) and ifun (the instruction function).

- It holds a predicted value of the program counter.
- Instr_valid - Checks if this byte corresponds to a legal Y86-64 instruction. This signal is used to detect illegal instruction.
- Need_regids - Checks if this instruction includes a register specifier byte? -----> bool need_regids = icode in { IRRMOVQ, IOPQ, IPUSHQ, IPOPQ, IIRMOVQ, IRMMOVQ, IMRMOVQ };
- Need_valC - Checks if this instruction includes a constant word -----> bool need_valC = icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IJXX, ICALL };

Decode module

- The decode module is used to interpret the encoded instruction that is provided in the instruction register at a given clock cycle.
- It sits between the fetch and decode stages. It holds information about the most recently fetched instruction for processing by the decode stage.
- The instruction fields are decoded to provide register identifiers for four addresses, two read and two write, that the register file uses.
- valA and valB become signals that are read from the register file.
- valE and valM which are the 2 write back values are the data for the writes.

Execute module

- This module creates ALU used in our processor.
- I.e, It performs operation add, subtract, and, or exclusive-or on inputs aluA and aluB based on the setting of the alufun signal.
- The ALU output becomes the signal valE.
- The operands are listed with aluB first, followed by aluA to make sure the subq instruction subtracts valA from valB.
- Three condition codes are generated—zero, sign, and overflow every time it operates.

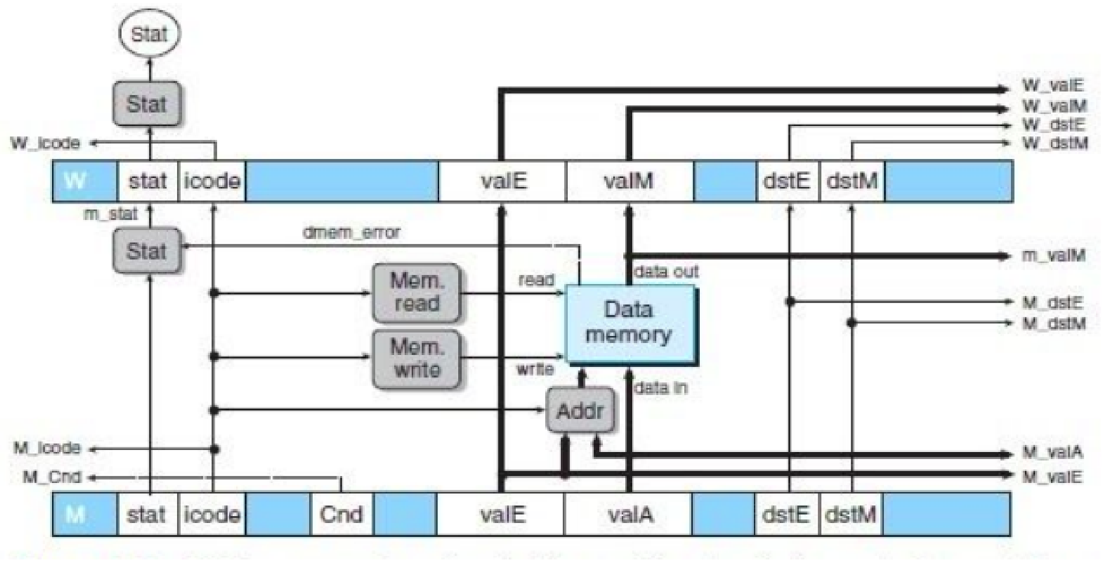
cond.v

The unit uses a combination of the condition codes and the function code to determine whether a conditional branch or data transfer should take place

cc.v

Condition Code block - Our ALU generates the three signals on which the condition codes are based—zero, sign, and overflow—every time it operates. We also have “Set CC,” which determines whether or not to update the condition codes, has signals m_stat and W_stat as inputs.

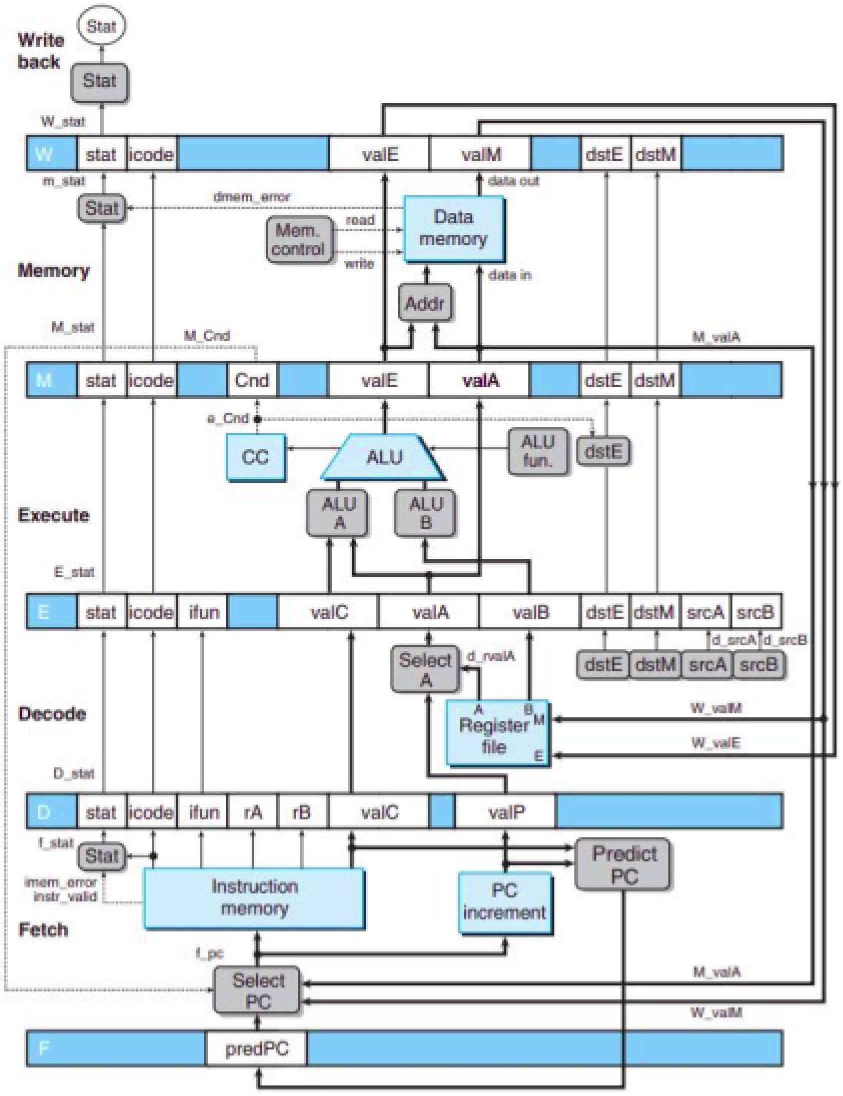
Memory module



- This is the final module used while developing this processor design.
- Sits between the execute and memory stages.
- The memory stage has the task of either reading or writing program data.
- Two control blocks generate the values for the memory.

- When a read operation is performed, the data memory generates the value valM.
- Two other blocks generate the control signals indicating whether to perform a read or a write operation
- Multiple versions of these values will be connected with the various instructions going through the system in our pipelined design.

Entire Flowchart



Instructions and Features Supported By Processor:

- halt
- nop
- rrmovq
- irmovq
- rmmovq
- mrmovq
- OPq
- jXX
- cmovXX
- call
- ret
- pushq
- popq

Features - 64-bit Y86 5-staged pipelined processor architecture Clock pulse has a period of 10 and hence the frequency is 1/10 = 0.1 Instruction memory has size of 8*1024 Data memory has a size of 64*8192

DATA HAZARDS

Data hazard resistance can be checked by changing the instruction file to instr.mem.

CHALLENGES ENCOUNTERED

Some of the Challenges encountered were working with pipelined registers and implementing the jump instruction properly. Implementing the call and ret instruction was also very challenging.

INSTRUCTIONS TO RUN THE CODE FILE

1. Download the codes.
2. Run the following command in the terminal:

```
$ iverilog -o processor processor_test.v processor.v
```

3. Then run:

```
$ vvp processor
```

4. To get the output waveforms, run:

```
$ gtkwave processor_test.vcd
```

TESTING

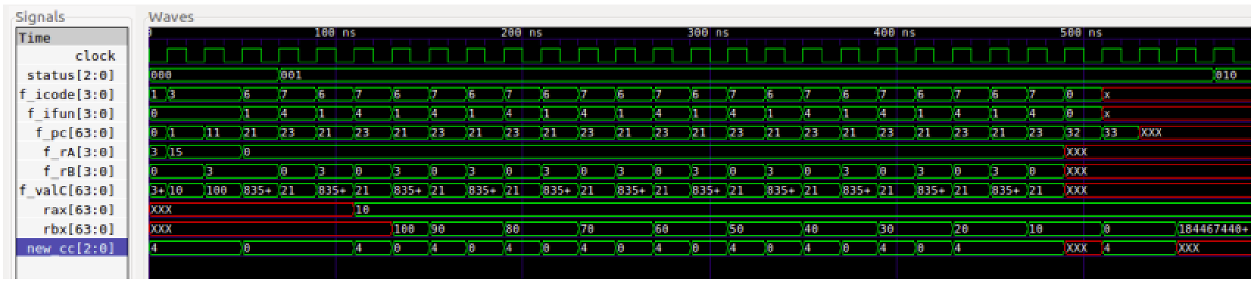
Code to find HCF of two numbers in C++

```
#include<bits/stdc++.h>

using namespace std;

int gcd(int a, int b)

{

if (b == 0)

return a;

return gcd(b, a % b);

}

int main()

{

13

int a=24,b=64;

cout<<gcd(a,b)<<endl;

}
```

Code to find HCF of two numbers in Assembly

```
rmovq 24, %rax

irmovq 64, %rbx

fun1:

rrmovq %rbx, %rcx

subq %rax, %rcx

jg .fun3

jl .fun2

halt

fun2:

subq %rax, %rbx

jmp .fun1

fun3:

rrmovq %rax, %rcx

rrmovq %rbx, %rax

rrmovq %rbx, %rbx

jmp .fun2
```

GTKwave output for HCF



This finds the gcd of 24 and 64 which comes out to be 8 and is stored in rbx. All the parameters are by their names (standard).