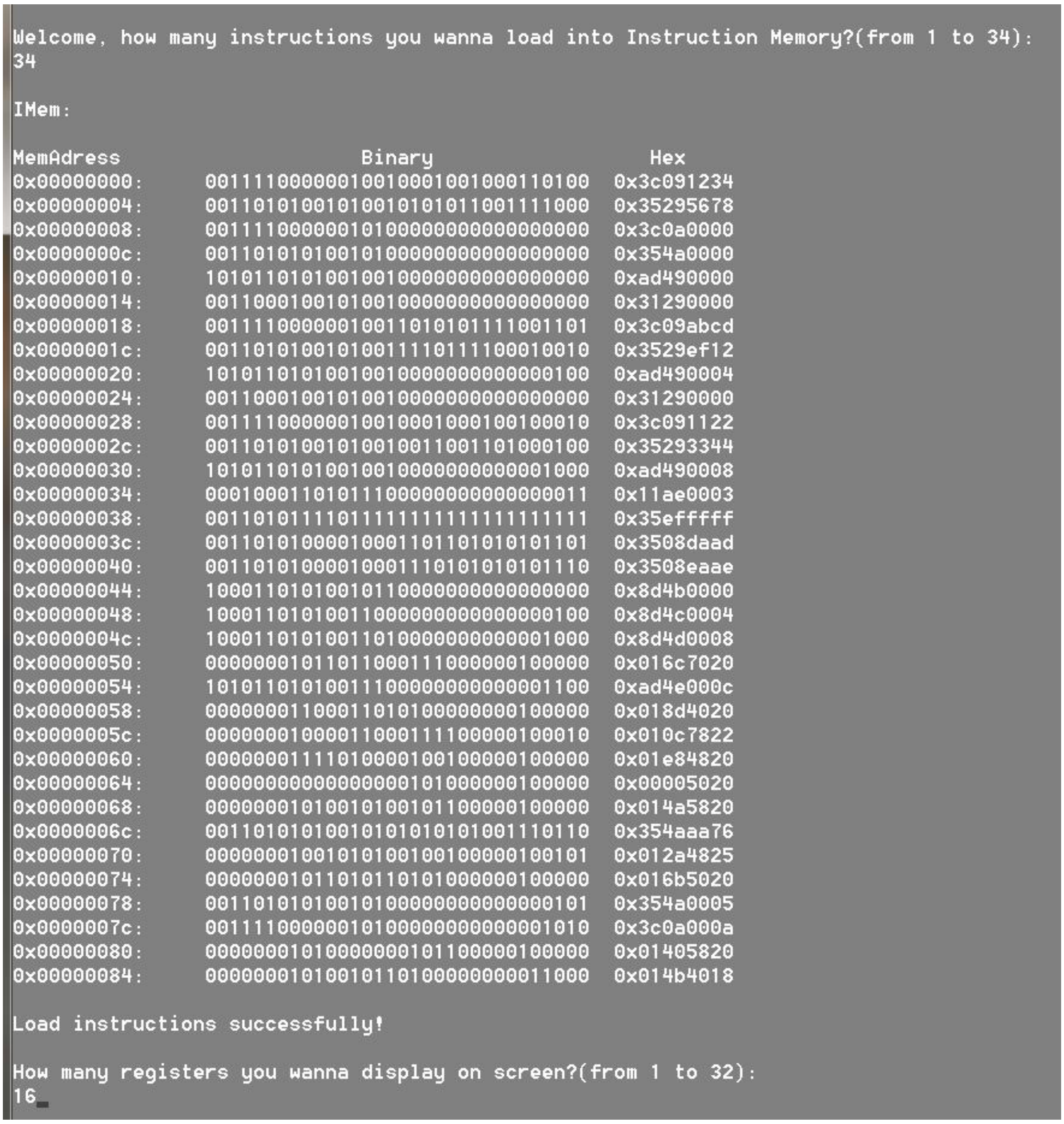
ECE510 Fall 2020

Project 3 - Simulation Result

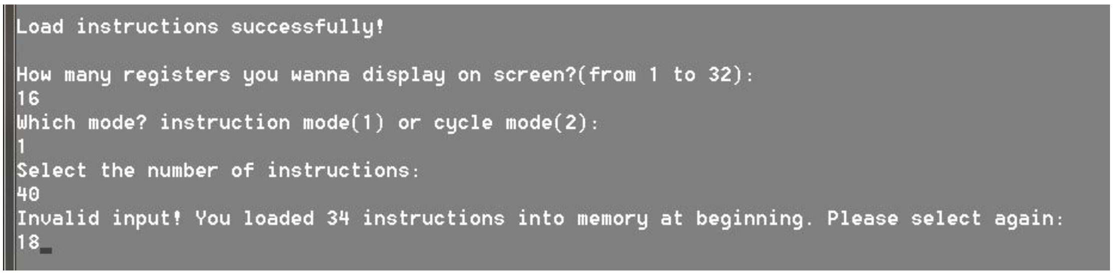
In this section, I'm showing you how my program goes correctly. Some necessary screenshots will be given as follow. And also these calculating result such as utilization and total cycle can be found in other folder. Any question, please reach me out.

Step1: select the number of instruction you wanna load into IMem(34 in this project) and select the number of register you wanna display on screen(16 is enough in this project)



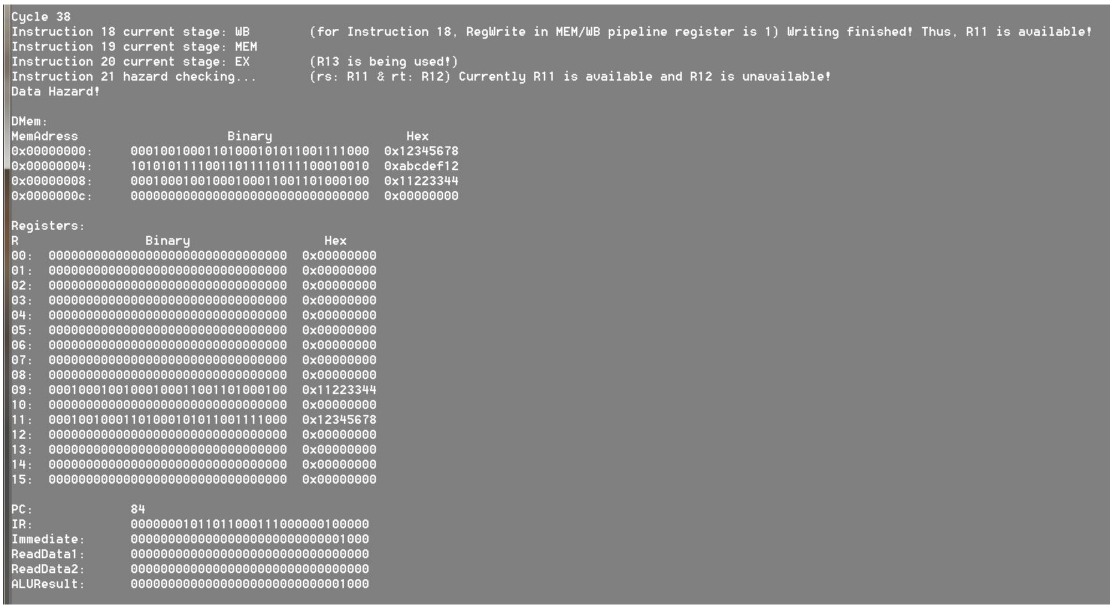
```
Welcome, how many instructions you wanna load into Instruction Memory?(from 1 to 34):
34

IMem:

MemAdress                    Binary                    Hex
0x00000000:     00111100000010010001001000110100  0x3c091234
0x00000004:     00110101001010010101011001111000  0x35295678
0x00000008:     00111100000010100000000000000000  0x3c0a0000
0x0000000c:     00110101010010100000000000000000  0x354a0000
0x00000010:     10101101010010010000000000000000  0xad490000
0x00000014:     00110001001010010000000000000000  0x31290000
0x00000018:     00111100000010011010101111001101  0x3c09abcd
0x0000001c:     00110101001010011110111100010010  0x3529ef12
0x00000020:     10101101010010010000000000000100  0xad490004
0x00000024:     00110001001010010000000000000000  0x31290000
0x00000028:     00111100000010010001000100100010  0x3c091122
0x0000002c:     00110101001010010011001101000100  0x35293344
0x00000030:     10101101010010010000000000001000  0xad490008
0x00000034:     00010001101011100000000000000011  0x11ae0003
0x00000038:     00110101111011111111111111111111  0x35efffff
0x0000003c:     00110101000010001101101010101101  0x3508daad
0x00000040:     00110101000010001110101010101110  0x3508eaae
0x00000044:     10001101010010110000000000000000  0x8d4b0000
0x00000048:     10001101010011000000000000000100  0x8d4c0004
0x0000004c:     10001101010011010000000000001000  0x8d4d0008
0x00000050:     00000001011011000111000000100000  0x016c7020
0x00000054:     10101101010011100000000000001100  0xad4e000c
0x00000058:     00000001100011010100000000100000  0x018d4020
0x0000005c:     00000001000011000111100000100010  0x010c7822
0x00000060:     00000001111010000100100000100000  0x01e84820
0x00000064:     00000000000000000101000000100000  0x00005020
0x00000068:     00000001010010100101100000100000  0x014a5820
0x0000006c:     00110101010010101010101001110110  0x354aaa76
0x00000070:     00000001001010100100100000100101  0x012a4825
0x00000074:     00000001011010110101000000100000  0x016b5020
0x00000078:     00110101010010100000000000000101  0x354a0005
0x0000007c:     00111100000010100000000000001010  0x3c0a000a
0x00000080:     00000001010000000101100000100000  0x01405820
0x00000084:     00000001010010110100000000011000  0x014b4018

Load instructions successfully!

How many registers you wanna display on screen?(from 1 to 32):
16
```

Step2: We should select the mode. Here we choose the first one instruction mode at first. And then select the number of instruction.
What I wanna point out is that there are several error detection in my program, so as you can see, here if we input a 40, It will go like invalid input. Because we only load 34 instructions at beginning. So we have to input again and just execute 18 instructions this time.



```
Load instructions successfully!

How many registers you wanna display on screen?(from 1 to 32):
16
Which mode? instruction mode(1) or cycle mode(2):
1
Select the number of instructions:
40
Invalid input! You loaded 34 instructions into memory at beginning. Please select again:
18
```

And then as you can see the process stops until instruction 18 writes back successfully.

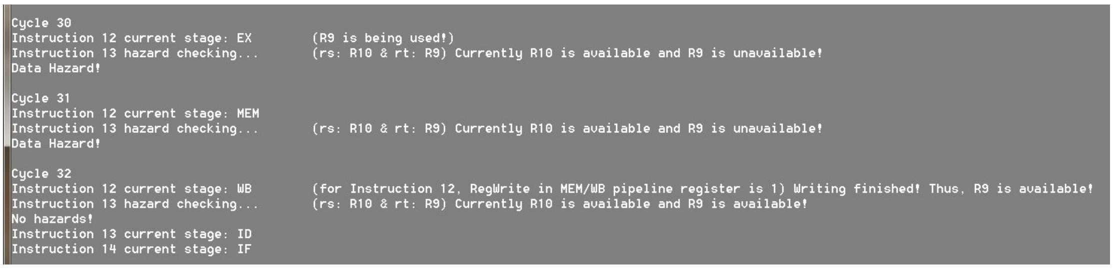


```
Cycle 38
Instruction 18 current stage: WB       (for Instruction 18, RegWrite in MEM/WB pipeline register is 1) Writing finished! Thus, R11 is available!
Instruction 19 current stage: MEM
Instruction 20 current stage: EX       (R13 is being used!)
Instruction 21 hazard checking...      (rs: R11 & rt: R12) Currently R11 is available and R12 is unavailable!
Data Hazard!

DMem:
MemAdress                     Binary                  Hex
0x00000000:     00010010001101000101011001111000  0x12345678
0x00000004:     10101011110011011110111100010010  0xabcdef12
0x00000008:     00010001001000100011001101000100  0x11223344
0x0000000c:     00000000000000000000000000000000  0x00000000

Registers:
R                  Binary                 Hex
00:  00000000000000000000000000000000  0x00000000
01:  00000000000000000000000000000000  0x00000000
02:  00000000000000000000000000000000  0x00000000
03:  00000000000000000000000000000000  0x00000000
04:  00000000000000000000000000000000  0x00000000
05:  00000000000000000000000000000000  0x00000000
06:  00000000000000000000000000000000  0x00000000
07:  00000000000000000000000000000000  0x00000000
08:  00000000000000000000000000000000  0x00000000
09:  00010001001000100011001101000100  0x11223344
10:  00000000000000000000000000000000  0x00000000
11:  00010010001101000101011001111000  0x12345678
12:  00000000000000000000000000000000  0x00000000
13:  00000000000000000000000000000000  0x00000000
14:  00000000000000000000000000000000  0x00000000
15:  00000000000000000000000000000000  0x00000000

PC:             84
IR:             00000001011011000111000000100000
Immediate:      00000000000000000000000000001000
ReadData1:      00000000000000000000000000000000
ReadData2:      00000000000000000000000000000000
ALUResult:      00000000000000000000000000001000
```

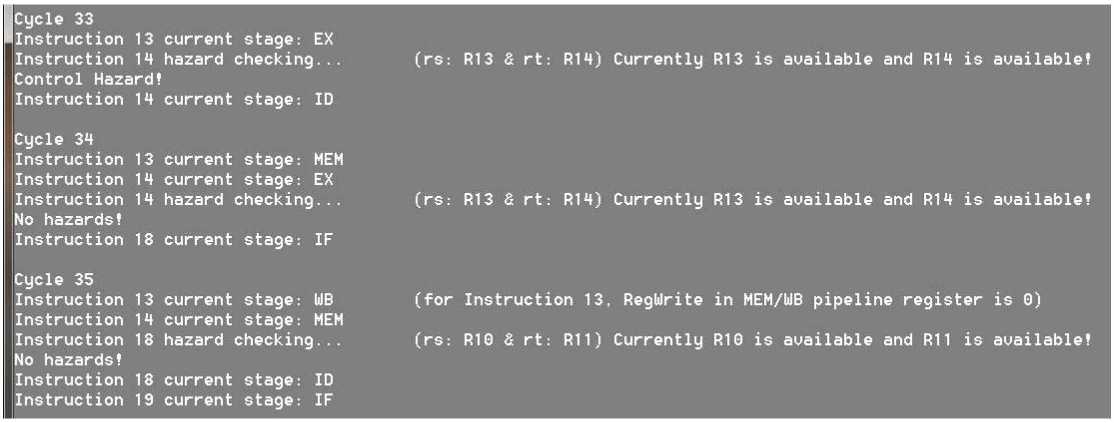
But what I wanna point out is here.

As mentioned before, some data hazards can be detected correctly and we just insert NOPs to deal with it. It just stops fetching and waits until the 12th writes back.

What’s more, since the 14th instruction is BEQ, so you can see my program detects control hazard correctly too. And it goes from 14th instruction straightly to 18th instruction.

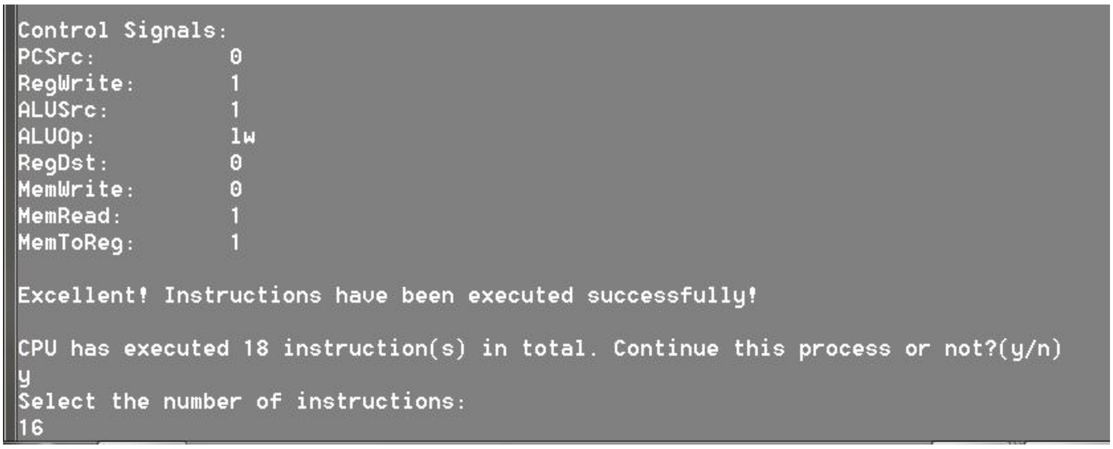


```
Cycle 30
Instruction 12 current stage: EX        (R9 is being used!)
Instruction 13 hazard checking...       (rs: R10 & rt: R9) Currently R10 is available and R9 is unavailable!
Data Hazard!

Cycle 31
Instruction 12 current stage: MEM
Instruction 13 hazard checking...       (rs: R10 & rt: R9) Currently R10 is available and R9 is unavailable!
Data Hazard!

Cycle 32
Instruction 12 current stage: WB        (for Instruction 12, RegWrite in MEM/WB pipeline register is 1) Writing finished! Thus, R9 is available!
Instruction 13 hazard checking...       (rs: R10 & rt: R9) Currently R10 is available and R9 is available!
No hazards!
Instruction 13 current stage: ID
Instruction 14 current stage: IF
```

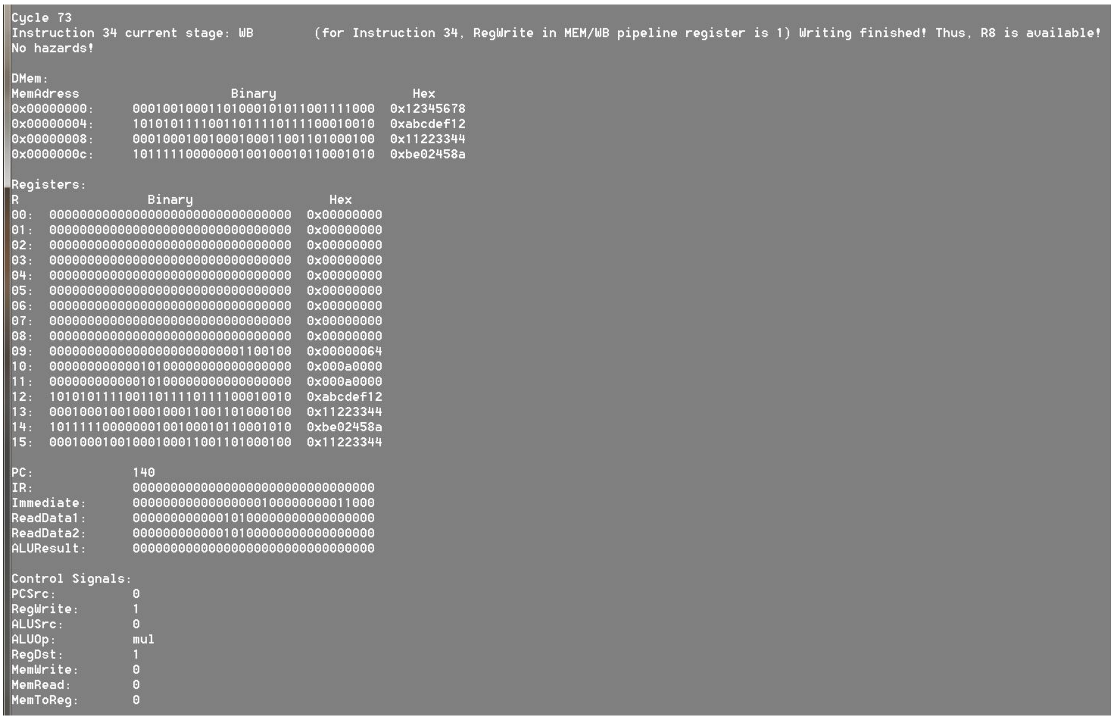


```
Cycle 33
Instruction 13 current stage: EX
Instruction 14 hazard checking...       (rs: R13 & rt: R14) Currently R13 is available and R14 is available!
Control Hazard!
Instruction 14 current stage: ID

Cycle 34
Instruction 13 current stage: MEM
Instruction 14 current stage: EX
Instruction 14 hazard checking...       (rs: R13 & rt: R14) Currently R13 is available and R14 is available!
No hazards!
Instruction 18 current stage: IF

Cycle 35
Instruction 13 current stage: WB        (for Instruction 13, RegWrite in MEM/WB pipeline register is 0)
Instruction 14 current stage: MEM
Instruction 18 hazard checking...       (rs: R10 & rt: R11) Currently R10 is available and R11 is available!
No hazards!
Instruction 18 current stage: ID
Instruction 19 current stage: IF
```

Step3: What’s my program different is here. Since this process is not been canceled yet. Now we have finished 18 instructions in total. If we continue this process and select 16, we will execute all the 34 instructions.

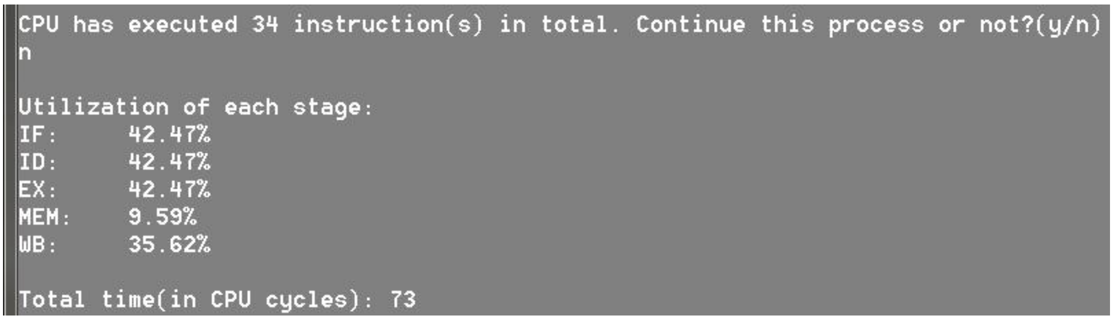


```
Control Signals:
PCSrc:          0
RegWrite:       1
ALUSrc:         1
ALUOp:          lw
RegDst:         0
MemWrite:       0
MemRead:        1
MemToReg:       1

Excellent! Instructions have been executed successfully!

CPU has executed 18 instruction(s) in total. Continue this process or not?(y/n)
y
Select the number of instructions:
16
```



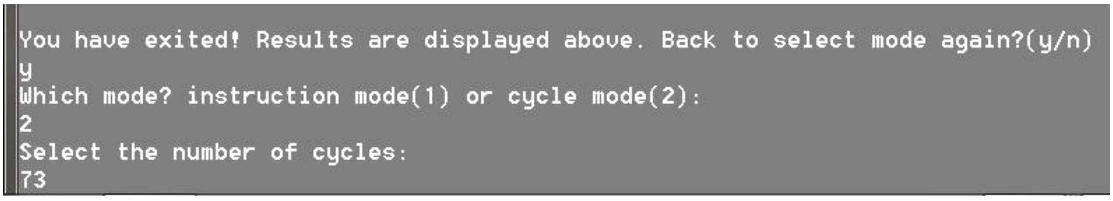
```
Cycle 73
Instruction 34 current stage: WB        (for Instruction 34, RegWrite in MEM/WB pipeline register is 1) Writing finished! Thus, R8 is available!
No hazards!

DMem:
MemAdress                    Binary                  Hex
0x00000000:     00010010001101000101011001111000  0x12345678
0x00000004:     10101011110011011110111100010010  0xabcdef12
0x00000008:     00010001001000100011001101000100  0x11223344
0x0000000c:     10111110000000100100010110001010  0xbe02458a

Registers:
R                   Binary                  Hex
00:  00000000000000000000000000000000  0x00000000
01:  00000000000000000000000000000000  0x00000000
02:  00000000000000000000000000000000  0x00000000
03:  00000000000000000000000000000000  0x00000000
04:  00000000000000000000000000000000  0x00000000
05:  00000000000000000000000000000000  0x00000000
06:  00000000000000000000000000000000  0x00000000
07:  00000000000000000000000000000000  0x00000000
08:  00000000000000000000000000000000  0x00000000
09:  00000000000000000000000001100100  0x00000064
10:  00000000000010100000000000000000  0x000a0000
11:  00000000000010100000000000000000  0x000a0000
12:  10101011110011011110111100010010  0xabcdef12
13:  00010001001000100011001101000100  0x11223344
14:  10111110000000100100010110001010  0xbe02458a
15:  00010001001000100011001101000100  0x11223344

PC:             140
IR:             00000000000000000000000000000000
Immediate:      00000000000000000100000000011000
ReadData1:      00000000000010100000000000000000
ReadData2:      00000000000010100000000000000000
ALUResult:      00000000000000000000000000000000

Control Signals:
PCSrc:          0
RegWrite:       1
ALUSrc:         0
ALUOp:          mul
RegDst:         1
MemWrite:       0
MemRead:        0
MemToReg:       0
```

As you can see, the 34th instruction writes back correctly. If we can exit, the calculating result output will be printed on screen as follow.



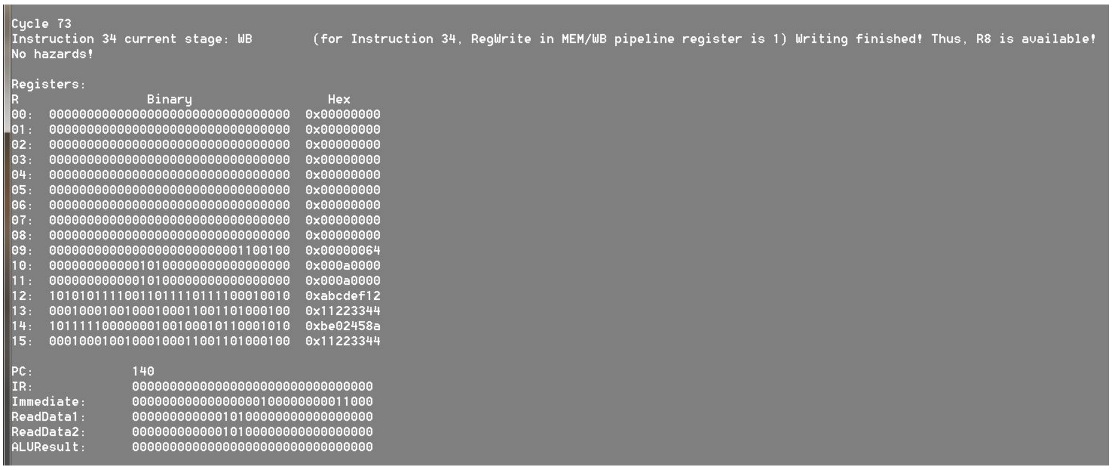
```
CPU has executed 34 instruction(s) in total. Continue this process or not?(y/n)
n

Utilization of each stage:
IF:     42.47%
ID:     42.47%
EX:     42.47%
MEM:    9.59%
WB:     35.62%

Total time(in CPU cycles): 73
```

Step4: What’s more, we can still back to select the mode and start a new process again. We select mode 2 this time and give a 73.

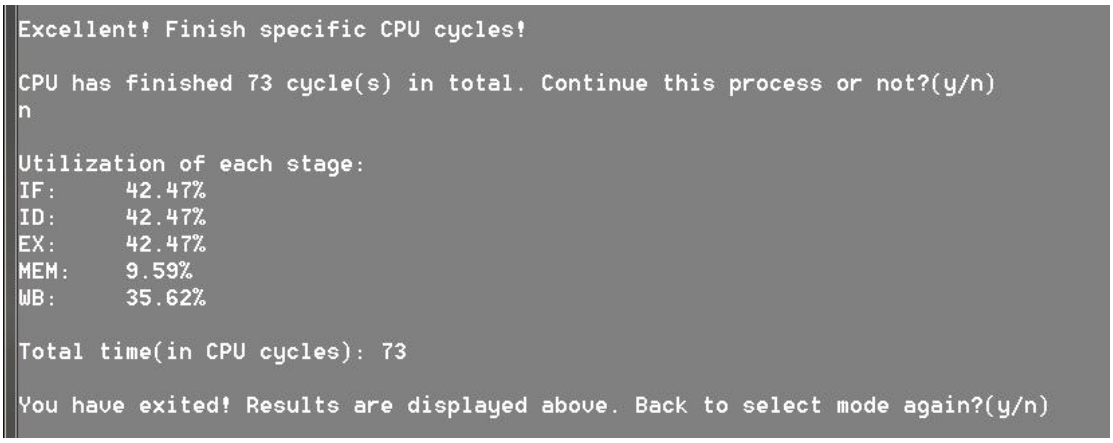


```
You have exited! Results are displayed above. Back to select mode again?(y/n)
y
Which mode? instruction mode(1) or cycle mode(2):
2
Select the number of cycles:
73
```

As you can see, the program stops at Cycle 73 accurately. Also the calculating results like utilization and total time could be printed on screen if we exit.



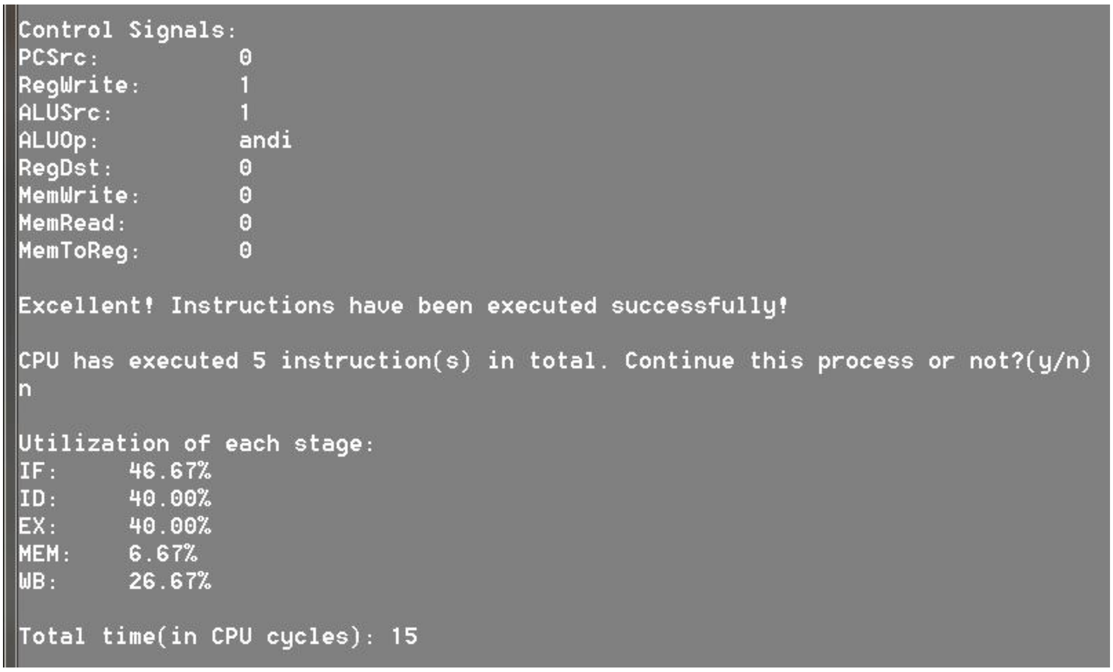
```
Cycle 73
Instruction 34 current stage: WB        (for Instruction 34, RegWrite in MEM/WB pipeline register is 1) Writing finished! Thus, R8 is available!
No hazards!

Registers:
R                 Binary                  Hex
00:  00000000000000000000000000000000  0x00000000
01:  00000000000000000000000000000000  0x00000000
02:  00000000000000000000000000000000  0x00000000
03:  00000000000000000000000000000000  0x00000000
04:  00000000000000000000000000000000  0x00000000
05:  00000000000000000000000000000000  0x00000000
06:  00000000000000000000000000000000  0x00000000
07:  00000000000000000000000000000000  0x00000000
08:  00000000000000000000000000000000  0x00000000
09:  00000000000000000000000001100100  0x00000064
10:  00000000000010100000000000000000  0x000a0000
11:  00000000000010100000000000000000  0x000a0000
12:  10101011110011011110111100010010  0xabcdef12
13:  00010001001000100011001101000100  0x11223344
14:  10111110000000100100010110001010  0xbe02458a
15:  00010001001000100011001101000100  0x11223344

PC:             140
IR:             00000000000000000000000000000000
Immediate:      00000000000000000100000000011000
ReadData1:      00000000000010100000000000000000
ReadData2:      00000000000010100000000000000000
ALUResult:      00000000000000000000000000000000
```



```
Excellent! Finish specific CPU cycles!

CPU has finished 73 cycle(s) in total. Continue this process or not?(y/n)
n

Utilization of each stage:
IF:      42.47%
ID:      42.47%
EX:      42.47%
MEM:     9.59%
WB:      35.62%

Total time(in CPU cycles): 73

You have exited! Results are displayed above. Back to select mode again?(y/n)
```

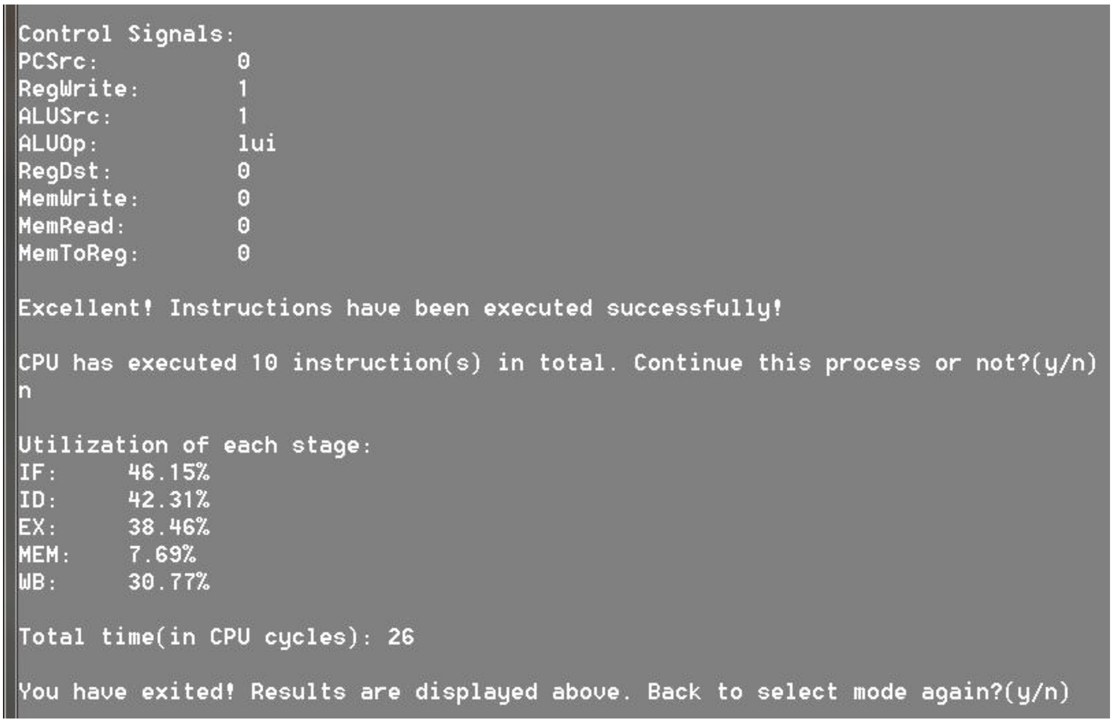
Finally, as a demo, I'll also show you several calculating results output of different instructions input. More calculating results and different operations, please check the code folder and run the program for more details. Thank you for your time!

Mode 1 and 5 instructions:



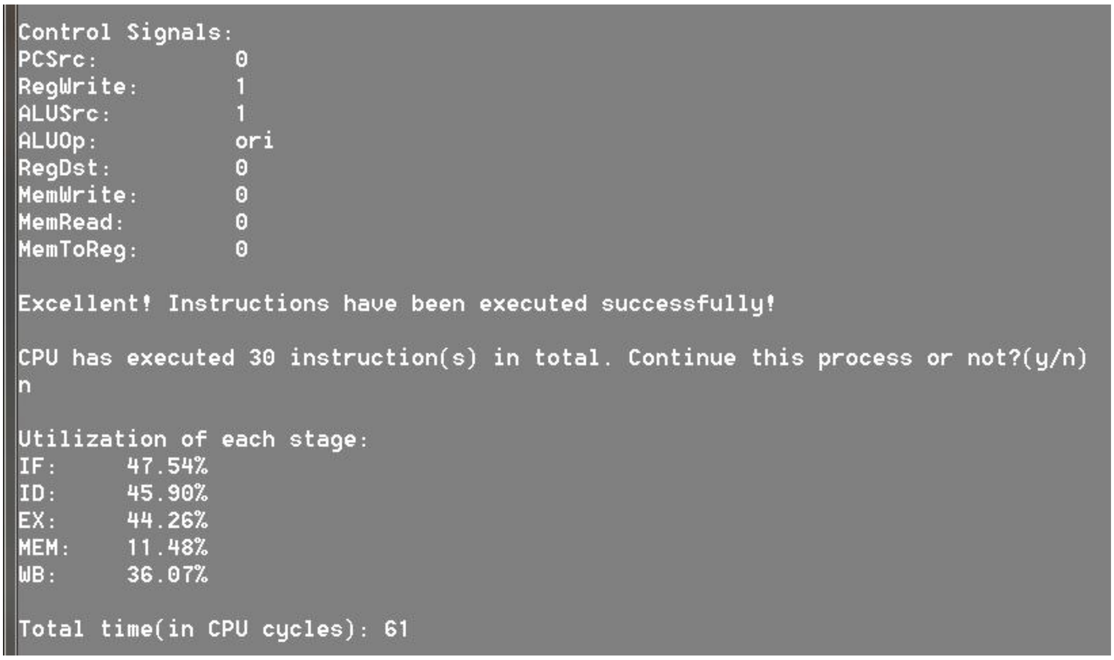
```
Control Signals:
PCSrc:          0
RegWrite:       1
ALUSrc:         1
ALUOp:          andi
RegDst:         0
MemWrite:       0
MemRead:        0
MemToReg:       0

Excellent! Instructions have been executed successfully!

CPU has executed 5 instruction(s) in total. Continue this process or not?(y/n)
n

Utilization of each stage:
IF:     46.67%
ID:     40.00%
EX:     40.00%
MEM:    6.67%
WB:     26.67%

Total time(in CPU cycles): 15
```

Mode 1 and 10 instructions:



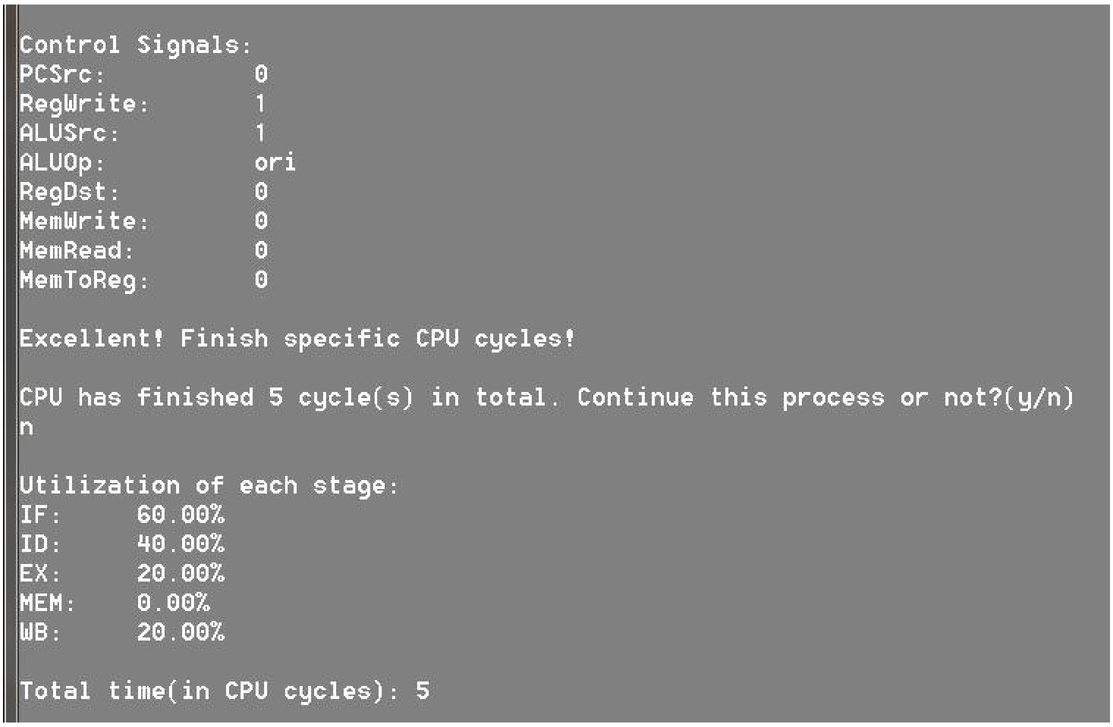
```
Control Signals:
PCSrc:          0
RegWrite:       1
ALUSrc:         1
ALUOp:          lui
RegDst:         0
MemWrite:       0
MemRead:        0
MemToReg:       0

Excellent! Instructions have been executed successfully!

CPU has executed 10 instruction(s) in total. Continue this process or not?(y/n)
n

Utilization of each stage:
IF:     46.15%
ID:     42.31%
EX:     38.46%
MEM:    7.69%
WB:     30.77%

Total time(in CPU cycles): 26

You have exited! Results are displayed above. Back to select mode again?(y/n)
```

Mode 1 and 30 instructions:



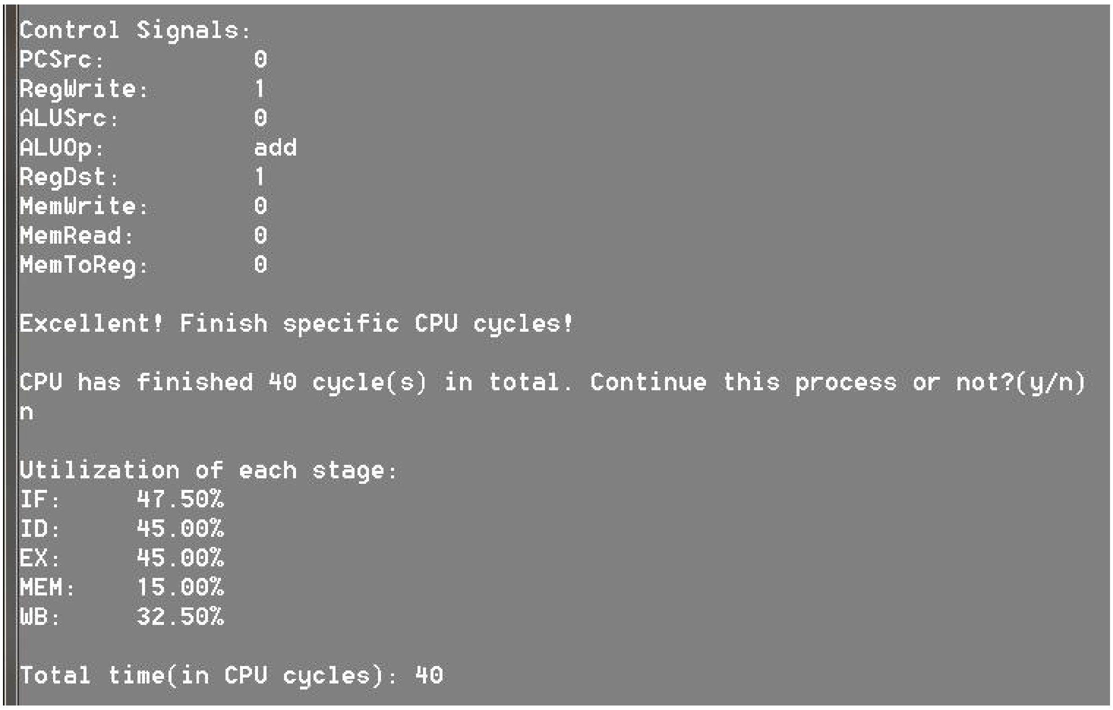
```
Control Signals:
PCSrc:          0
RegWrite:       1
ALUSrc:         1
ALUOp:          ori
RegDst:         0
MemWrite:       0
MemRead:        0
MemToReg:       0

Excellent! Instructions have been executed successfully!

CPU has executed 30 instruction(s) in total. Continue this process or not?(y/n)
n

Utilization of each stage:
IF:     47.54%
ID:     45.90%
EX:     44.26%
MEM:    11.48%
WB:     36.07%

Total time(in CPU cycles): 61
```

Mode 2 and 5 cycles:



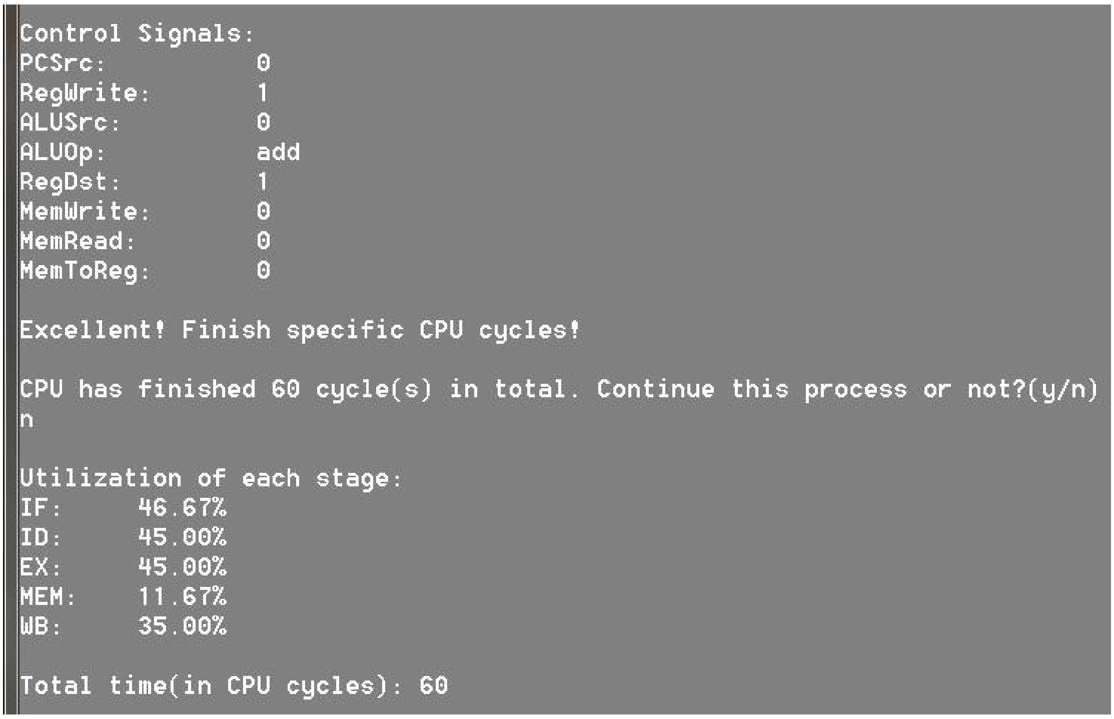
```
Control Signals:
PCSrc:          0
RegWrite:       1
ALUSrc:         1
ALUOp:          ori
RegDst:         0
MemWrite:       0
MemRead:        0
MemToReg:       0

Excellent! Finish specific CPU cycles!

CPU has finished 5 cycle(s) in total. Continue this process or not?(y/n)
n

Utilization of each stage:
IF:     60.00%
ID:     40.00%
EX:     20.00%
MEM:    0.00%
WB:     20.00%

Total time(in CPU cycles): 5
```

Mode 2 and 40 cycles:



```
Control Signals:
PCSrc:          0
RegWrite:       1
ALUSrc:         0
ALUOp:          add
RegDst:         1
MemWrite:       0
MemRead:        0
MemToReg:       0

Excellent! Finish specific CPU cycles!

CPU has finished 40 cycle(s) in total. Continue this process or not?(y/n)
n

Utilization of each stage:
IF:     47.50%
ID:     45.00%
EX:     45.00%
MEM:    15.00%
WB:     32.50%

Total time(in CPU cycles): 40
```

Mode 2 and 60 cycles:



```
Control Signals:
PCSrc:          0
RegWrite:       1
ALUSrc:         0
ALUOp:          add
RegDst:         1
MemWrite:       0
MemRead:        0
MemToReg:       0

Excellent! Finish specific CPU cycles!

CPU has finished 60 cycle(s) in total. Continue this process or not?(y/n)
n

Utilization of each stage:
IF:     46.67%
ID:     45.00%
EX:     45.00%
MEM:    11.67%
WB:     35.00%

Total time(in CPU cycles): 60
```