ICS-LAB1 REPORT

PB20000096 潘廷岳

Part1.version L

At the beginning,I thought maybe I need to judge the number's plus or minus characteristic.But then,I found it could be simplified to 5 lines as follows:

```
.ORIG x3000 ; start the program at location x3000
;
ADD R1, R1, #0 ;
LOOP BRz LOOP1 ; do it again if the counter is not yet zero
ADD R7, R7, R0 ; add R1 to sum(R7)
ADD R1, R1, #-1 ;
BRnzp LOOP ;
;
LOOP1 HALT ; halt
.END
```

It could be proved as following mathematical justification:

```
assumption: x*y MOD (2^16)=a;

Then:   (2^16-x)*y MOD (2^16) = (-x)*y MOD (2^16)   = -a
```

So,5 lines is enough.

Translate it to machine language is:

```
0011 0000 0000 0000 ;.ORIG x3000 ;
;
0001 001 001 1 00000 ;ADD R1, R1, #0 ;
0000 010 000000011 ;LOOP BRz LOOP1 ;
0001 111 111 0 00 000 ;ADD R7, R7, R0 ;
0001 001 001 1 11111;ADD R1, R1, #-1 ;
0000 111 111111100 ;BRnzp LOOP ;
;
1111 0000 00100101 ;LOOP1 HALT ;
```

This program could deal with any condition.

Part2.version P

*Initial version

Being inspired by Fast-power method to calculate a^b,I designed an algorithm to calculate a*b.The assembly language of it is:

```
.ORIG x3000 ; start the program at location x3000
; R5,is used for checking the bits of the R1 one by one
; R6 to judge the plus or minus characteristic
; R7, to be used for returning the sum
;
ADD R5, R5, #1 ;
ADD R1, R1, x0 ;//Check the symbol of R1;
BRn LOOP1 ;
LOOP1_B ;
;
LOOP3 ADD R1, R1, #0 ; //Check if R1 is zero
BRnz LOOP_Fns ; //Finish the calculate
AND R4, R1, R5 ; //Check the bits of the R1 one by one
BRp LOOP2;
LOOP2_B ;
;
ADD R0, R0, R0 ; //R0<<1
ADD R5, R5, R5 ; //R5<<1
BRnzp LOOP3 ;
;
;
LOOP1 NOT R1, R1 ; //Turn to positive number
ADD R1, R1, #1 ;
ADD R6, R6, #1 ;
BRnzp LOOP1_B ;
;
;
LOOP2 ADD R7, R7, R0 ;
NOT R3, R5 ;//subtraction the R5 from the R1
ADD R3, R3, #1 ;
ADD R1, R1, R3 ;
BRnzp LOOP2_B;
;
LOOP_Fns ;
AND R6, R6, #1;
BRz LOOP4;//If the result shou be negative ,then R7=(-R7)
NOT R7, R7 ;
ADD R7, R7, #1 ;
LOOP4 HALT ; halt
.END
```

The running row is mainly decided by the |R0|'s highest 1 bit(e.x |5| and |-5| is 101,it's highest 1 bit's place is 3) .Assumpt it's h,and it has q bit1,then we need to execute about (a+5*(h-q)+12*h+4) lines to finish the calculation(if R1>=0,then a=3;else a=8).

We could find that this version could deal with some little number 'sproblem,but not so time-consume-stable(e.x R2=2^{16}-1)

*Final version

```
.ORIG x3000 ; start the program at location x3000
; R5,is used for checking the bits of the R1 one by one
; R6 to judge the plus or minus characteristic
; R7, to be used for returning the sum
;
ADD R5, R5, #1 ;
ADD R1, R1, x0 ;//Check the symbol of R1;
BRn LOOP1 ;
LOOP1_B ;
;
LOOP3 BRz LOOP_Fns ; //Finish the calculate
```

```
AND R4, R1, R5 ; //Check the bits of the R1 one by one
BRz LOOP2;
ADD R7, R7, R0;
LOOP2 ADD R0, R0, R0 ; //R0<<1
ADD R5, R5, R5 ; //R5<<1
BRnzp LOOP3 ;
;
LOOP1 NOT R1, R1 ; //Turn to positive number
ADD R1, R1, #1 ;
ADD R6, R6, #1 ;
BRnzp LOOP1_B ;
;
LOOP_Fns ;
AND R6, R6, #1;
BRz LOOP4;//If the result shou be negative ,then R7=(-R7)
NOT R7, R7 ;
ADD R7, R7, #1 ;
LOOP4 HALT ; halt
.END
```

To stable the running time,I delete the process on substracting R5 from R1,and change the finishing condition ,it could bring me less calculation.Now,we need about 127 lines on average to complete the calculation.

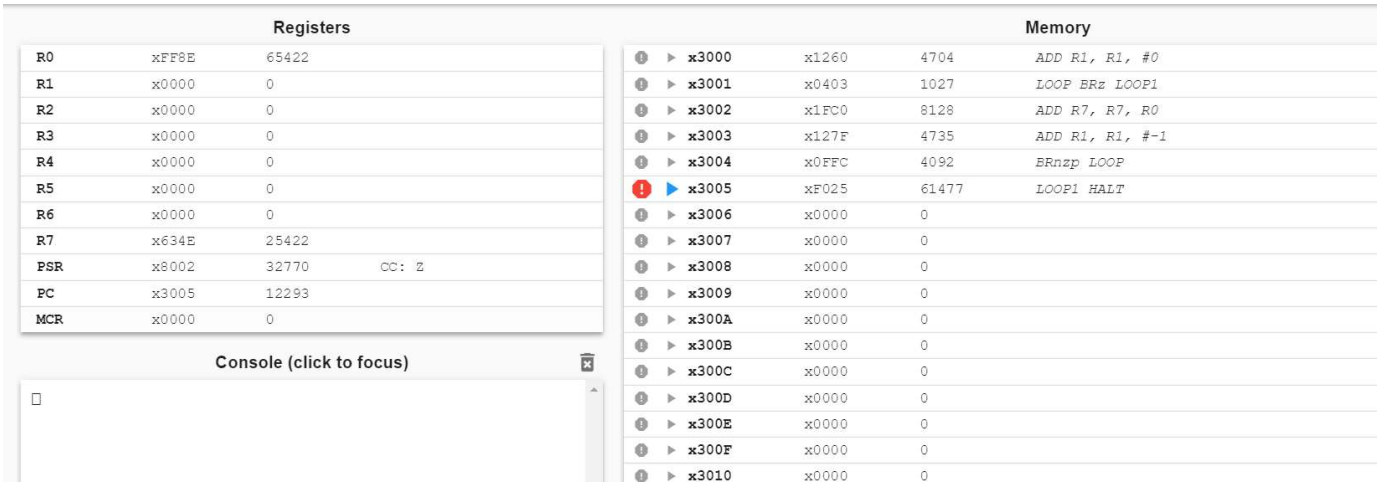
Part3.Answer-verification

To verify the answer,I wrote a cpp-file to verify the answer from machine language.The cpp-program's main construction as follows:

```
Mod=1<<16;
while(1)
{
    cin>>a>>b;
    if(a==b && a==-1) break;
    cout<<(a*b)%Mod<<endl;
}
```

Part4.Picture display

L_version(-223*-114)



Registers

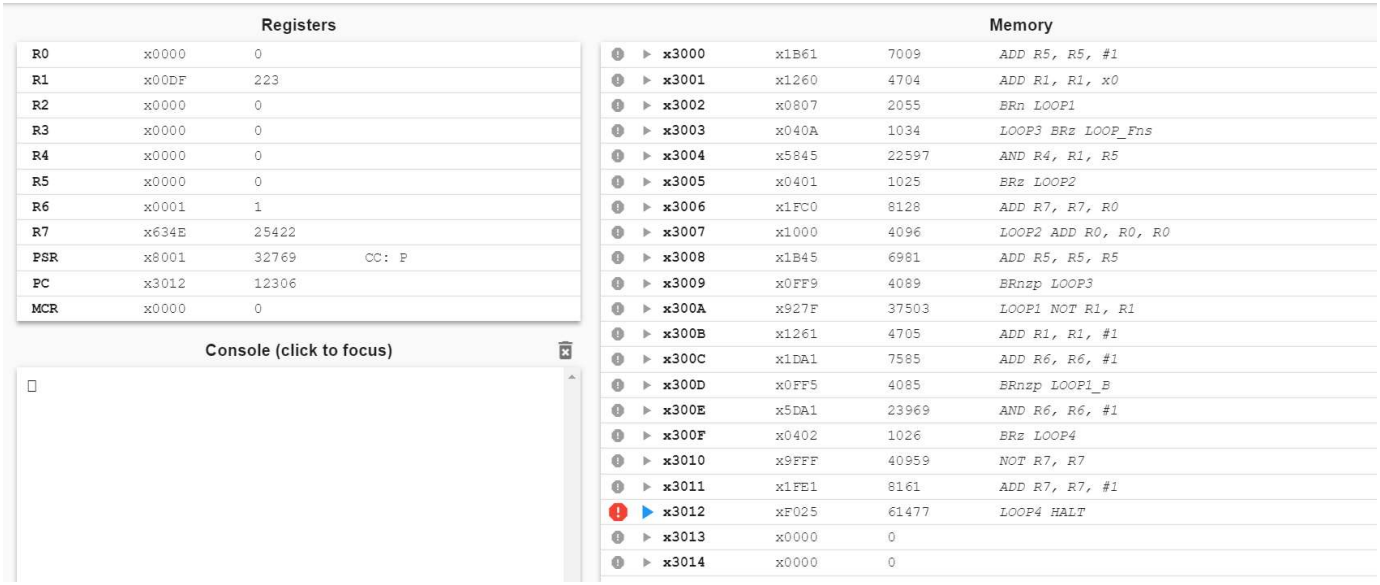
R0	xFF8E	65422	
R1	x0000	0	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x634E	25422	
PSR	x8002	32770	CC: Z
PC	x3005	12293	
MCR	x0000	0	

Console (click to focus)

Memory

x3000	x1260	4704	ADD R1, R1, #0
x3001	x0403	1027	LOOP BRz LOOP1
x3002	x1FC0	8128	ADD R7, R7, R0
x3003	x127F	4735	ADD R1, R1, #-1
x3004	x0FFC	4092	BRnzp LOOP
x3005	xF025	61477	LOOP1 HALT
x3006	x0000	0	
x3007	x0000	0	
x3008	x0000	0	
x3009	x0000	0	
x300A	x0000	0	
x300B	x0000	0	
x300C	x0000	0	
x300D	x0000	0	
x300E	x0000	0	
x300F	x0000	0	
x3010	x0000	0	

P_version(-223*-114)



Registers

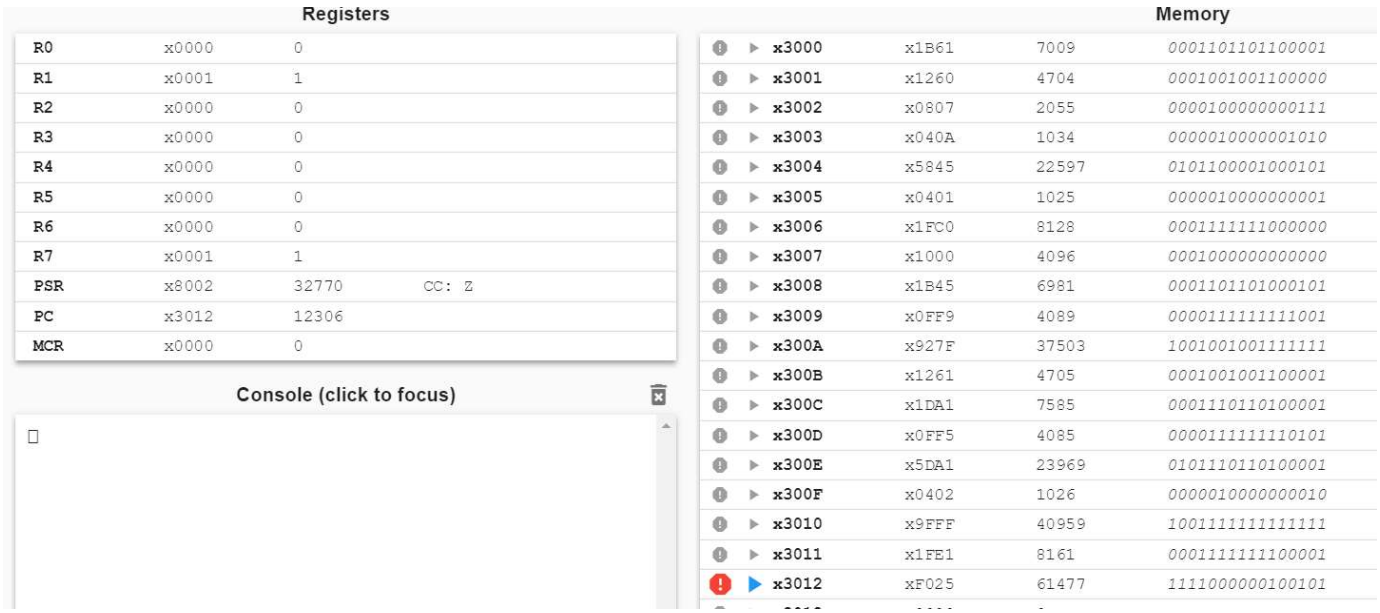
R0	x0000	0	
R1	x00DF	223	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0001	1	
R7	x634E	25422	
PSR	x8001	32769	CC: P
PC	x3012	12306	
MCR	x0000	0	

Console (click to focus)

Memory

x3000	x1B61	7009	ADD R5, R5, #1
x3001	x1260	4704	ADD R1, R1, x0
x3002	x0807	2055	BRn LOOP1
x3003	x040A	1034	LOOP3 BRz LOOP_Fns
x3004	x5845	22597	AND R4, R1, R5
x3005	x0401	1025	BRz LOOP2
x3006	x1FC0	8128	ADD R7, R7, R0
x3007	x1000	4096	LOOP2 ADD R0, R0, R0
x3008	x1B45	6981	ADD R5, R5, R5
x3009	x0FF9	4089	BRnzp LOOP3
x300A	x927F	37503	LOOP1 NOT R1, R1
x300B	x1261	4705	ADD R1, R1, #1
x300C	x1DA1	7585	ADD R6, R6, #1
x300D	x0FF5	4085	BRnzp LOOP1_B
x300E	x5DA1	23969	AND R6, R6, #1
x300F	x0402	1026	BRz LOOP4
x3010	x9FFF	40959	NOT R7, R7
x3011	x1FE1	8161	ADD R7, R7, #1
x3012	xF025	61477	LOOP4 HALT
x3013	x0000	0	
x3014	x0000	0	

P*version*bit(1*1)



Registers

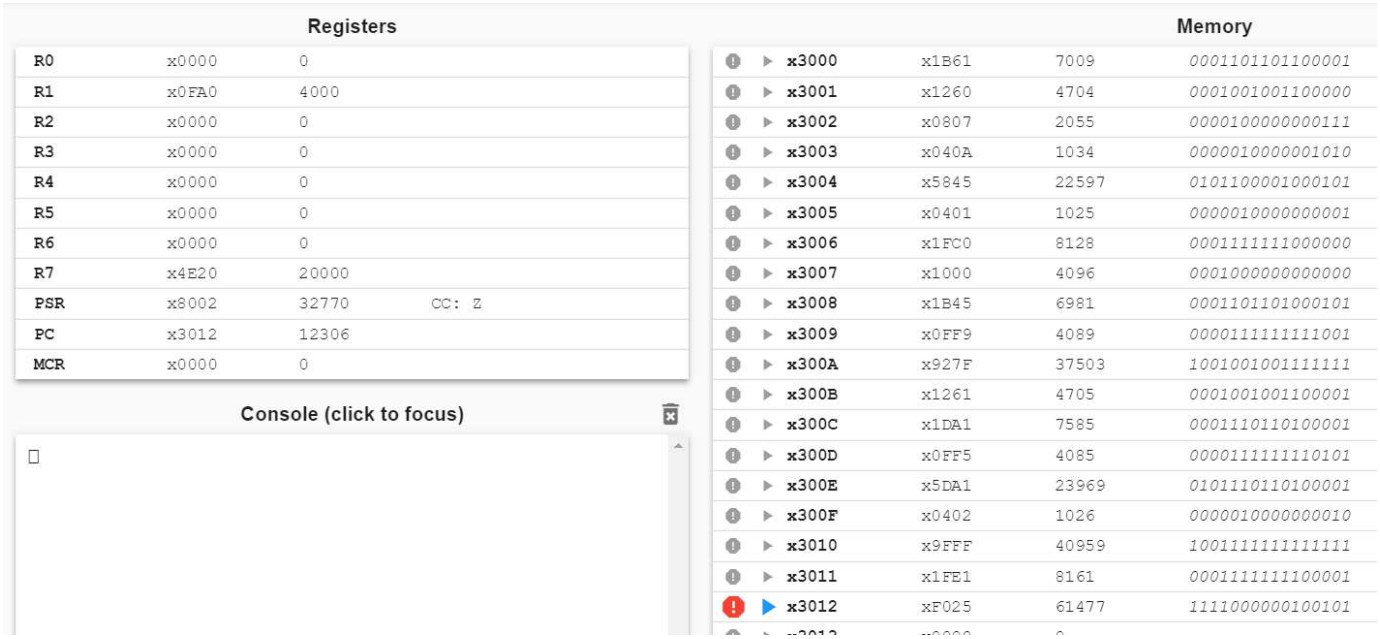
R0	x0000	0	
R1	x0001	1	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0001	1	
PSR	x8002	32770	CC: Z
PC	x3012	12306	
MCR	x0000	0	

Console (click to focus)

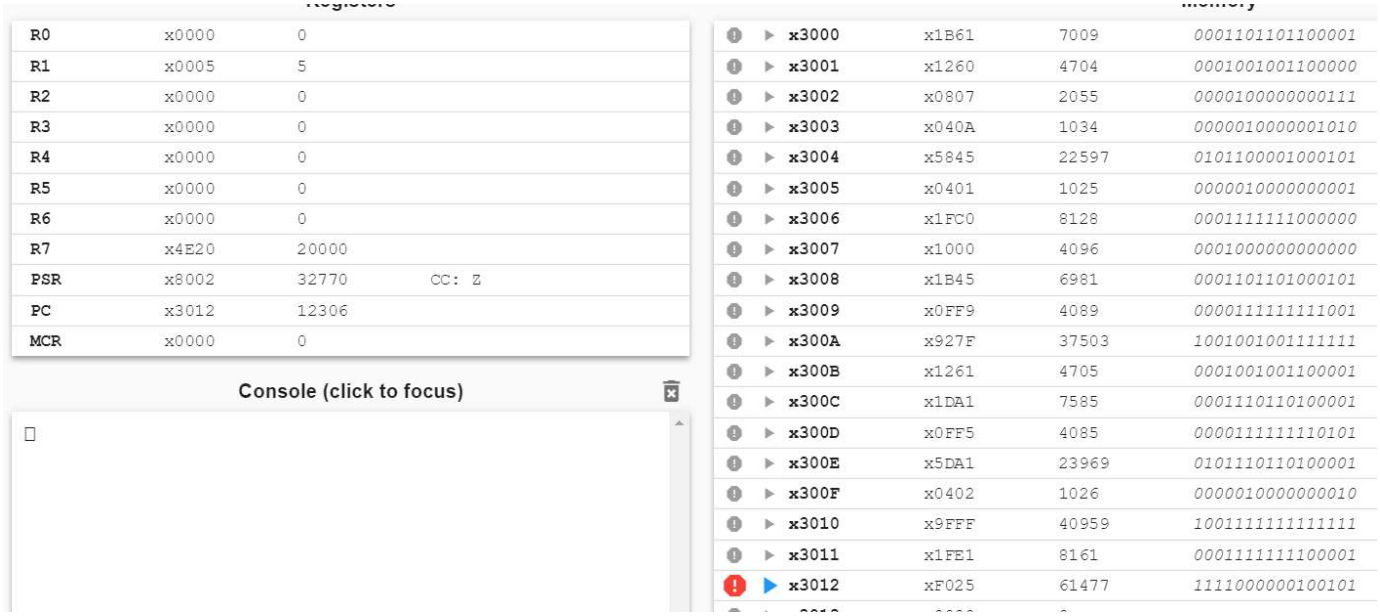
Memory

x3000	x1B61	7009	0001101101100001
x3001	x1260	4704	0001001001100000
x3002	x0807	2055	0000100000000111
x3003	x040A	1034	0000010000001010
x3004	x5845	22597	0101100001000101
x3005	x0401	1025	0000010000000001
x3006	x1FC0	8128	0001111111000000
x3007	x1000	4096	0001000000000000
x3008	x1B45	6981	0001101101000101
x3009	x0FF9	4089	0000111111111001
x300A	x927F	37503	1001001001111111
x300B	x1261	4705	0001001001100001
x300C	x1DA1	7585	0001110110100001
x300D	x0FF5	4085	0000111111110101
x300E	x5DA1	23969	0101110110100001
x300F	x0402	1026	0000010000000010
x3010	x9FFF	40959	1001111111111111
x3011	x1FE1	8161	0001111111100001
x3012	xF025	61477	1111000000100101

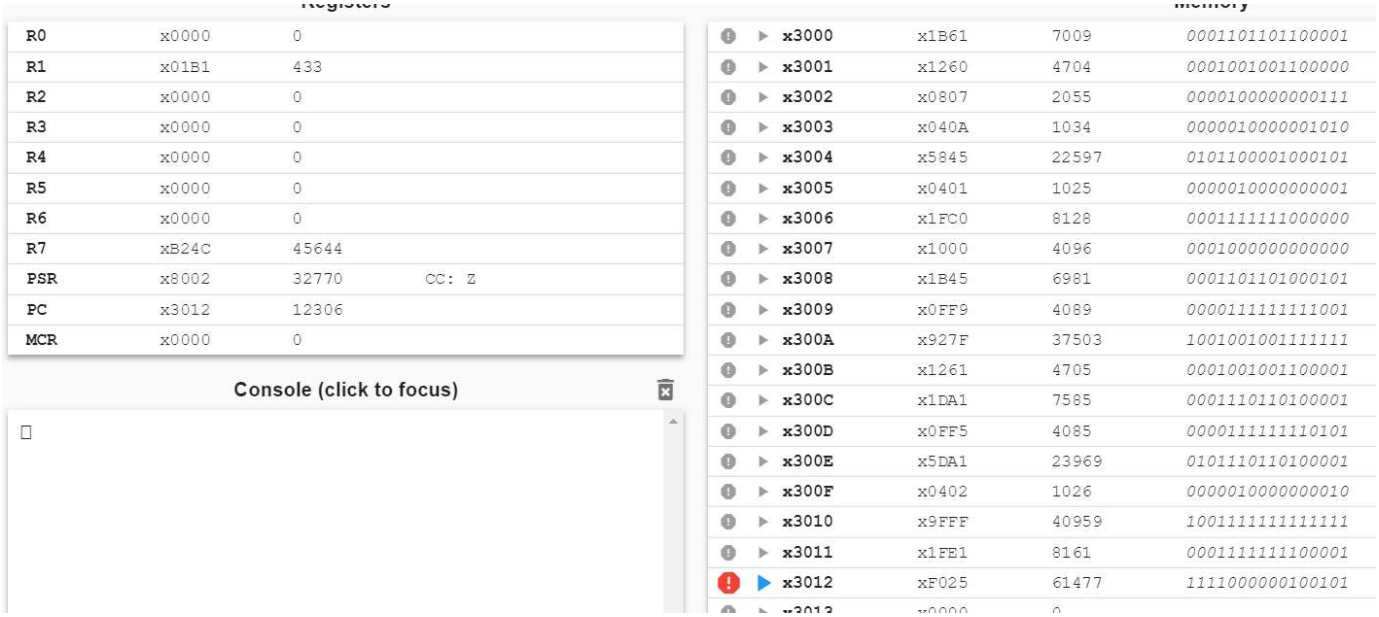
P*version*bit(5*4000)



P*version*bit(4000*5)



P*version*bit(-500*433)

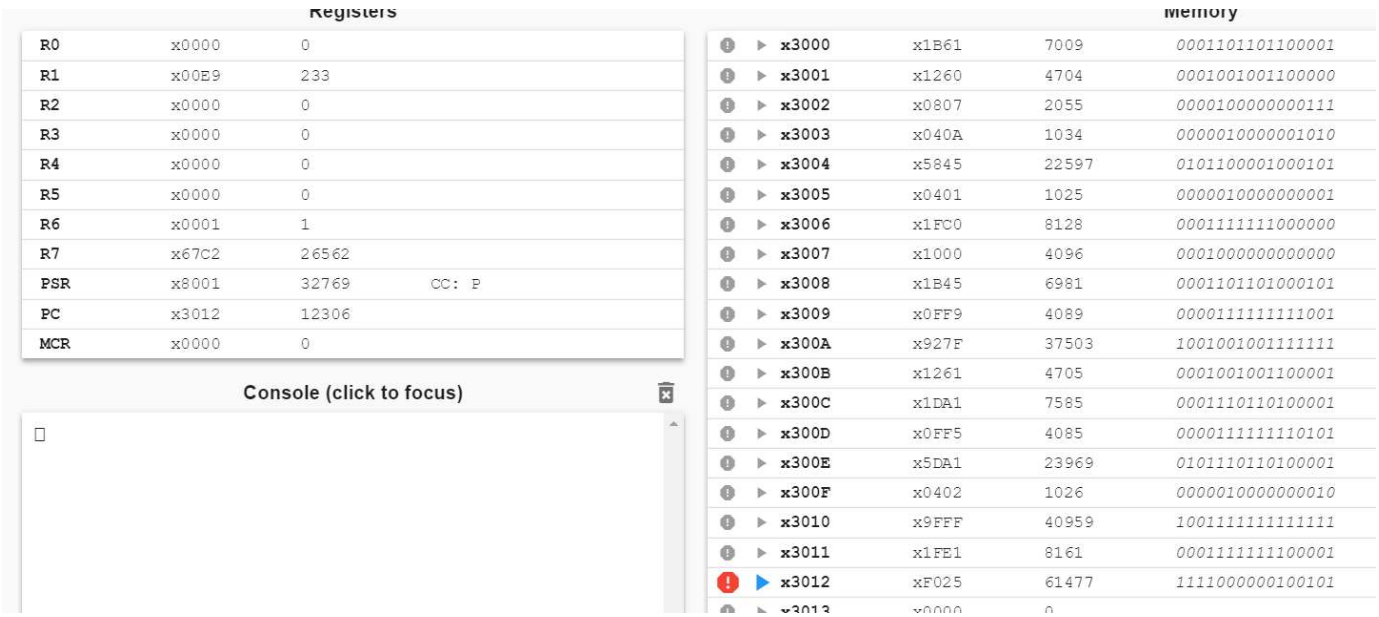


Registers			
R0	x0000	0	
R1	x01B1	433	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	xB24C	45644	
PSR	x8002	32770	CC: Z
PC	x3012	12306	
MCR	x0000	0	

Console (click to focus)

Memory			
x3000	x1B61	7009	0001101101100001
x3001	x1260	4704	0001001001100000
x3002	x0807	2055	0000100000000111
x3003	x040A	1034	0000010000001010
x3004	x5845	22597	0101100001000101
x3005	x0401	1025	0000010000000001
x3006	x1FC0	8128	0001111111000000
x3007	x1000	4096	0001000000000000
x3008	x1B45	6981	0001101101000101
x3009	x0FF9	4089	0000111111111001
x300A	x927F	37503	1001001001111111
x300B	x1261	4705	0001001001100001
x300C	x1DA1	7585	0001110110100001
x300D	x0FF5	4085	0000111111110101
x300E	x5DA1	23969	0101110110100001
x300F	x0402	1026	0000010000000010
x3010	x9FFF	40959	1001111111111111
x3011	x1FE1	8161	0001111111100001
x3012	xF025	61477	1111000000100101

P*version*bit(-114*-233)



Registers			
R0	x0000	0	
R1	x00E9	233	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0001	1	
R7	x67C2	26562	
PSR	x8001	32769	CC: P
PC	x3012	12306	
MCR	x0000	0	

Console (click to focus)

Memory			
x3000	x1B61	7009	0001101101100001
x3001	x1260	4704	0001001001100000
x3002	x0807	2055	0000100000000111
x3003	x040A	1034	0000010000001010
x3004	x5845	22597	0101100001000101
x3005	x0401	1025	0000010000000001
x3006	x1FC0	8128	0001111111000000
x3007	x1000	4096	0001000000000000
x3008	x1B45	6981	0001101101000101
x3009	x0FF9	4089	0000111111111001
x300A	x927F	37503	1001001001111111
x300B	x1261	4705	0001001001100001
x300C	x1DA1	7585	0001110110100001
x300D	x0FF5	4085	0000111111110101
x300E	x5DA1	23969	0101110110100001
x300F	x0402	1026	0000010000000010
x3010	x9FFF	40959	1001111111111111
x3011	x1FE1	8161	0001111111100001
x3012	xF025	61477	1111000000100101