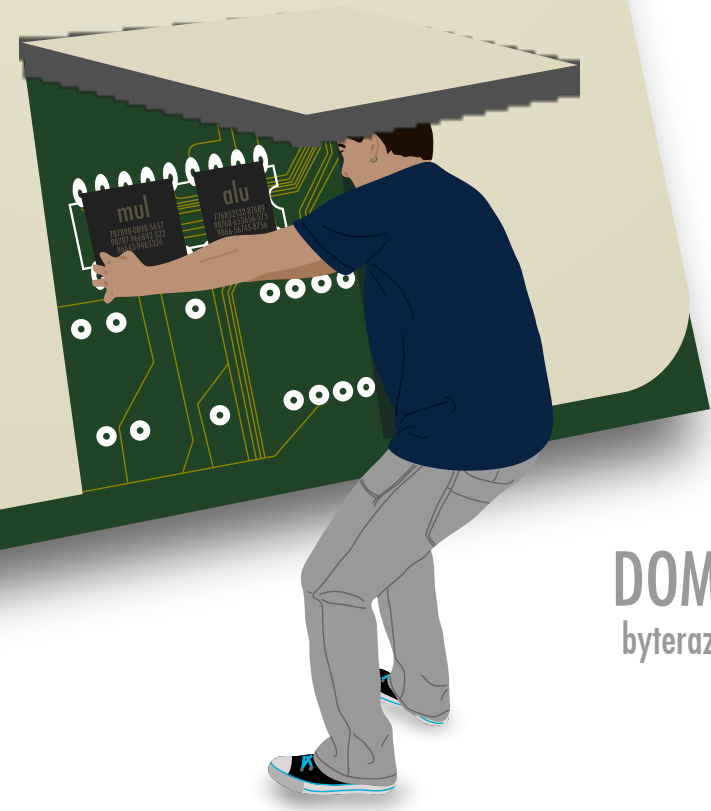
PROCESSOR DESIGN
how to build your own



DOMINIK MEYER
byterazor@federationhq.de
30C3

Who am I ?

profession

- research assistant at Helmut Schmidt University Hamburg
- research field: runtime reconfigurable systems

hobbies

- Field Programmable Gate Arrays (FPGAs)
- Hacking (VHDL, Perl, Linux, . . .)
- Rowing

contact

- web : `http://byterazor.federationhq.de`
- eMail : byterazor@federationhq.de
- twitter: @byterazor

Handout

http://byterazor.federationhq.de/download/handout_30C3.pdf

Overview

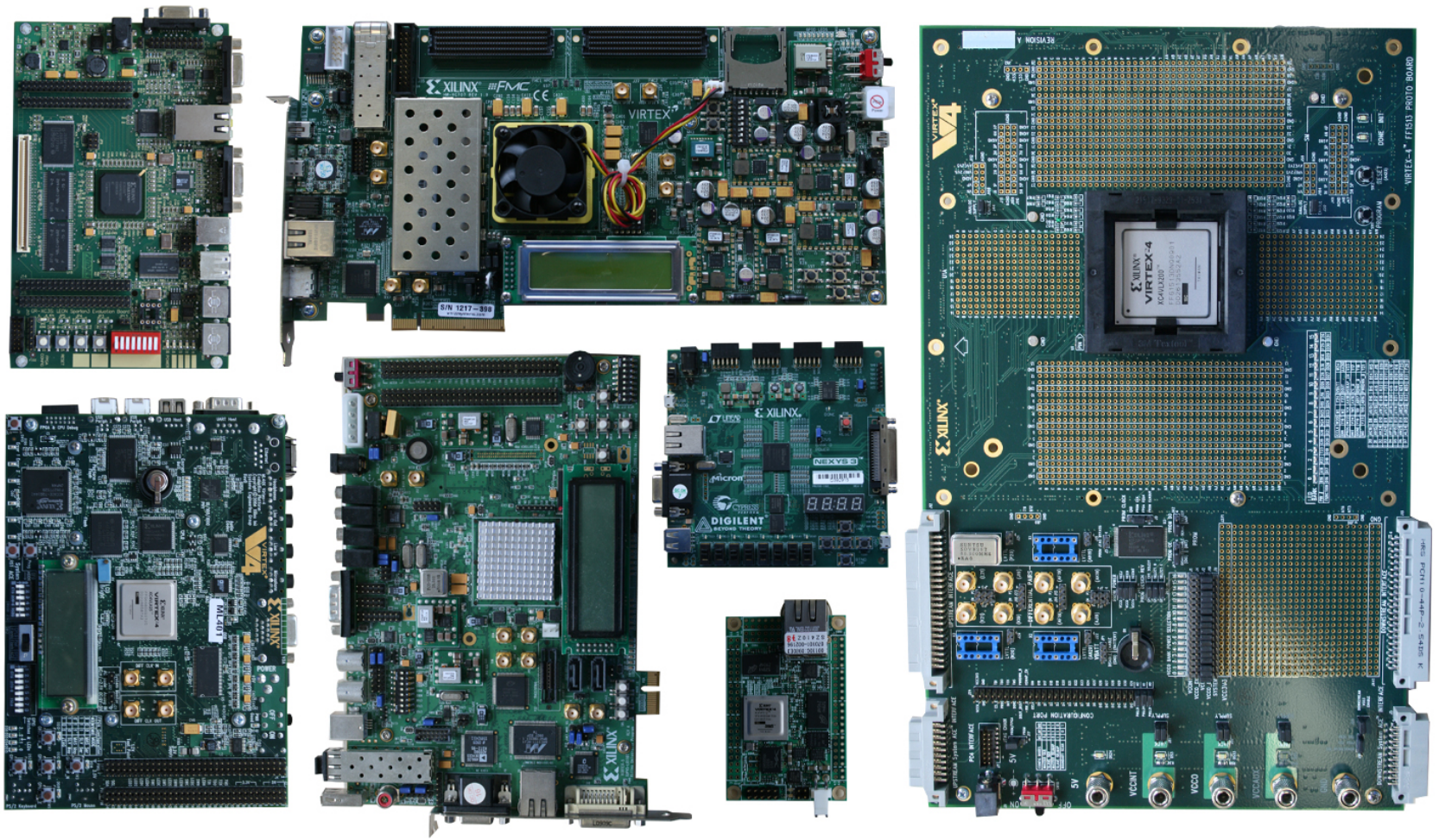
Disclaimer

- introduction to processor design
- nothing about processor performance
- just about a simple unpipelined processor core

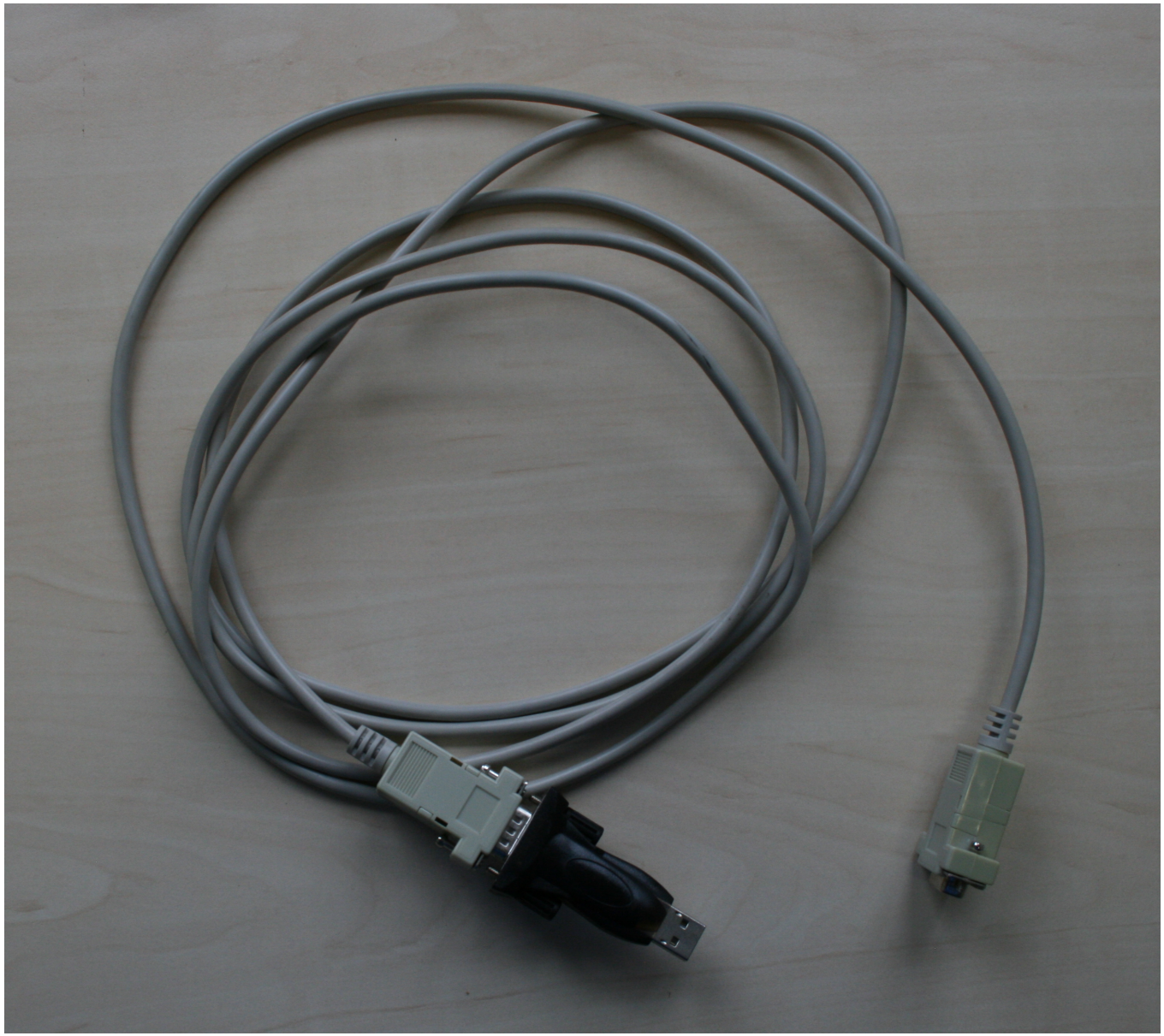
What do i need to build a processor core ?

Requirements ?

FPGA Board



Serial Connection to PC



HDL Knowledge
for example VHDL

```
entity c3b4bencoding is
  port (
   iRD    :  in std_logic;
   iK     :  in std_logic;
   iWord  :  in std_logic_vector(2 downto 0);
   oWord  : out std_logic_vector(3 downto 0)
  );
end c3b4bencoding;
```

english http://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf

german http://tams-www.informatik.uni-hamburg.de/vhdl/doc/ajmMaterial/vhdl.pdf

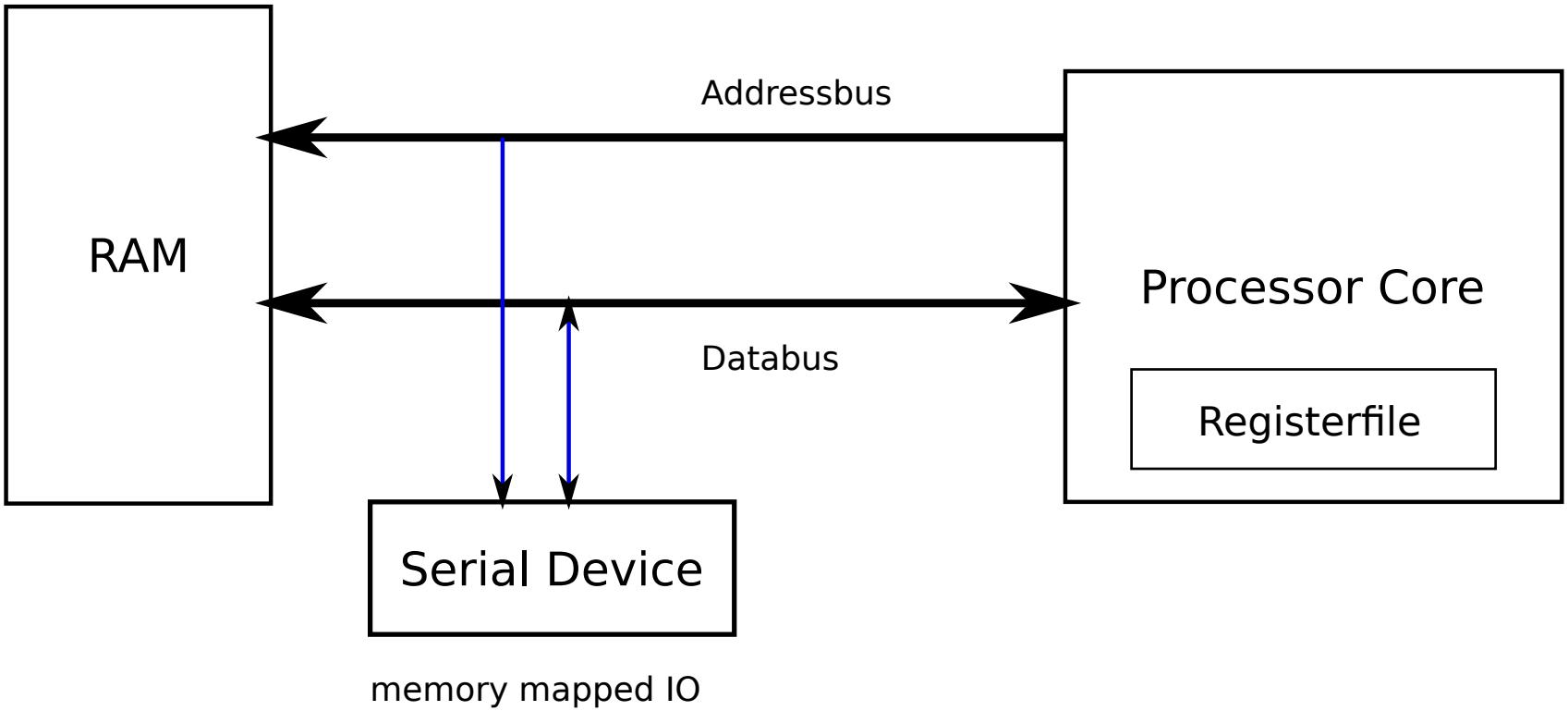
Starter knowledge of processor design

Topic of this talk !

How does a processor work ?

How ?

Not working alone !
System on Chip (SOC)



Remember: memory access is very slow, compared to intra processor memory

Instruction Set Architecture

Mnemonic	Parameter	Description
li	dst,<immidiate>	$dst = immediate$
loa	dst,<address>	$dst = RAM(address)$
sto	op1,<address>	$RAM(address) = op1$
shl	dst,op1	$dst = shl(op1)$
shr	dst,op1	$dst = shr(op1)$
add	dst,op1,op2	$dst = op1 + op2$
sub	dst,op1,op2	$dst = op1 - op2$
addc	dst,op1,op2	$dst = op1 + op2 + C$
subc	dst,op1,op2	$dst = op1 - op2 - C$
or	dst,op1,op2	$dst = op1 or op2$
and	dst,op1,op2	$dst = op1 and op2$
xor	dst,op1,op2	$dst = op1 xor op2$
not	dst,op1	$dst = not op1$
jpz	<address>	jump if zero to <address>
jpc	<address>	jump if carry to <address>
jmp	<address>	jump to address

Small C Programm

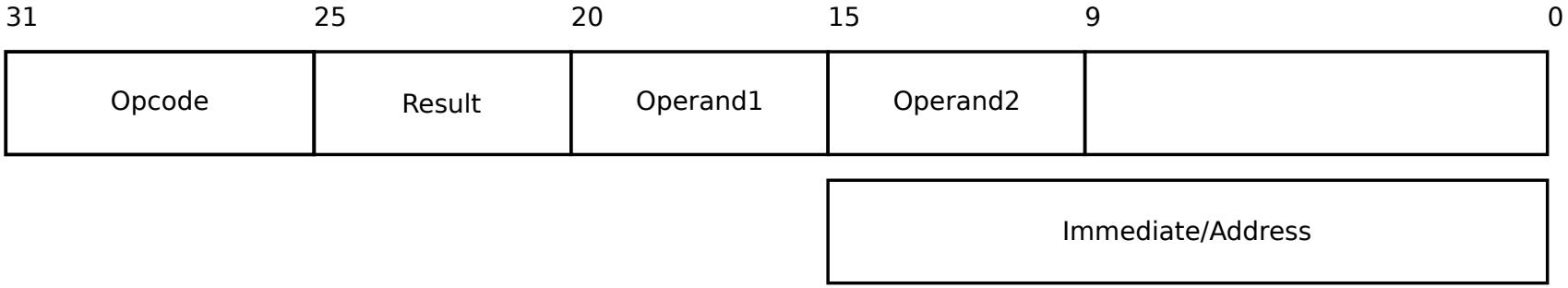
```c
// very simple multiplication
int main() {
    int a=5;
    int b=6;
    int result=0;

    int i;
    for (i=0; i<b; i++) {
        result += a;
    }


    return 0;
}
```

Play Compiler

```
// very simple multiplication
int main() {
    int a=5;
    int b=6;
    int result=0;

    int i;
    for (i=0; i<b; i++) {
        result += a;
    }


    return 0;
}
```

```
main:
  li $1, 5          # operand1
  li $2, 6          # operand2
  li $3, 0          # res reg

  li $4, 1          # for loop
  add $5, $0, $2    # cpy 1.op
loop:
  add $3,$3,$1      # add res
  sub $5,$5,$4      # decrement
  jpz end           # if zero
                    # jump end
  jmp loop
end:
```

Instruction Encoding

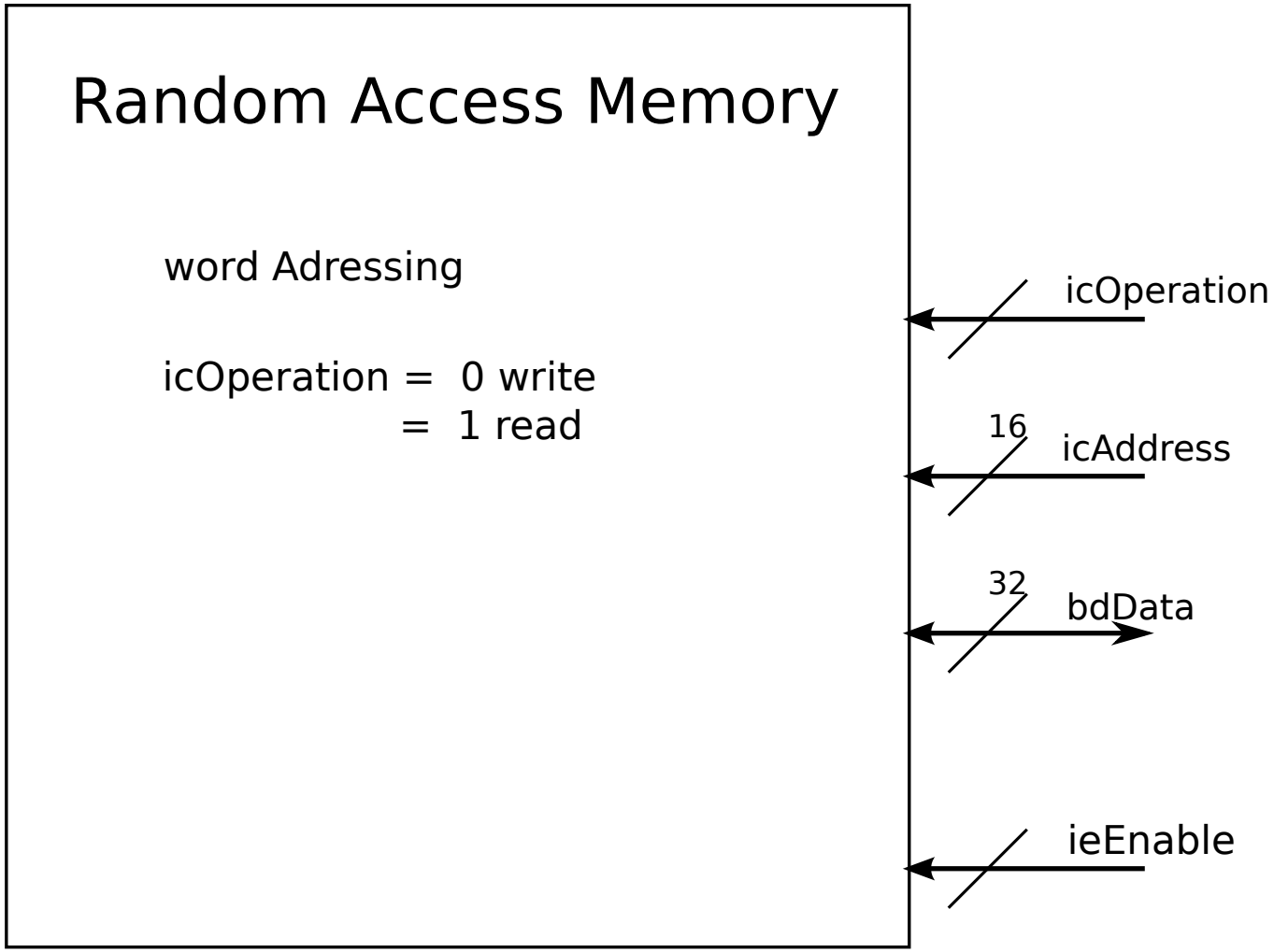


Instruction	Opcode
li	001111
add	000100
sub	000101
jpz	001100
jmp	001110
hlt	111111

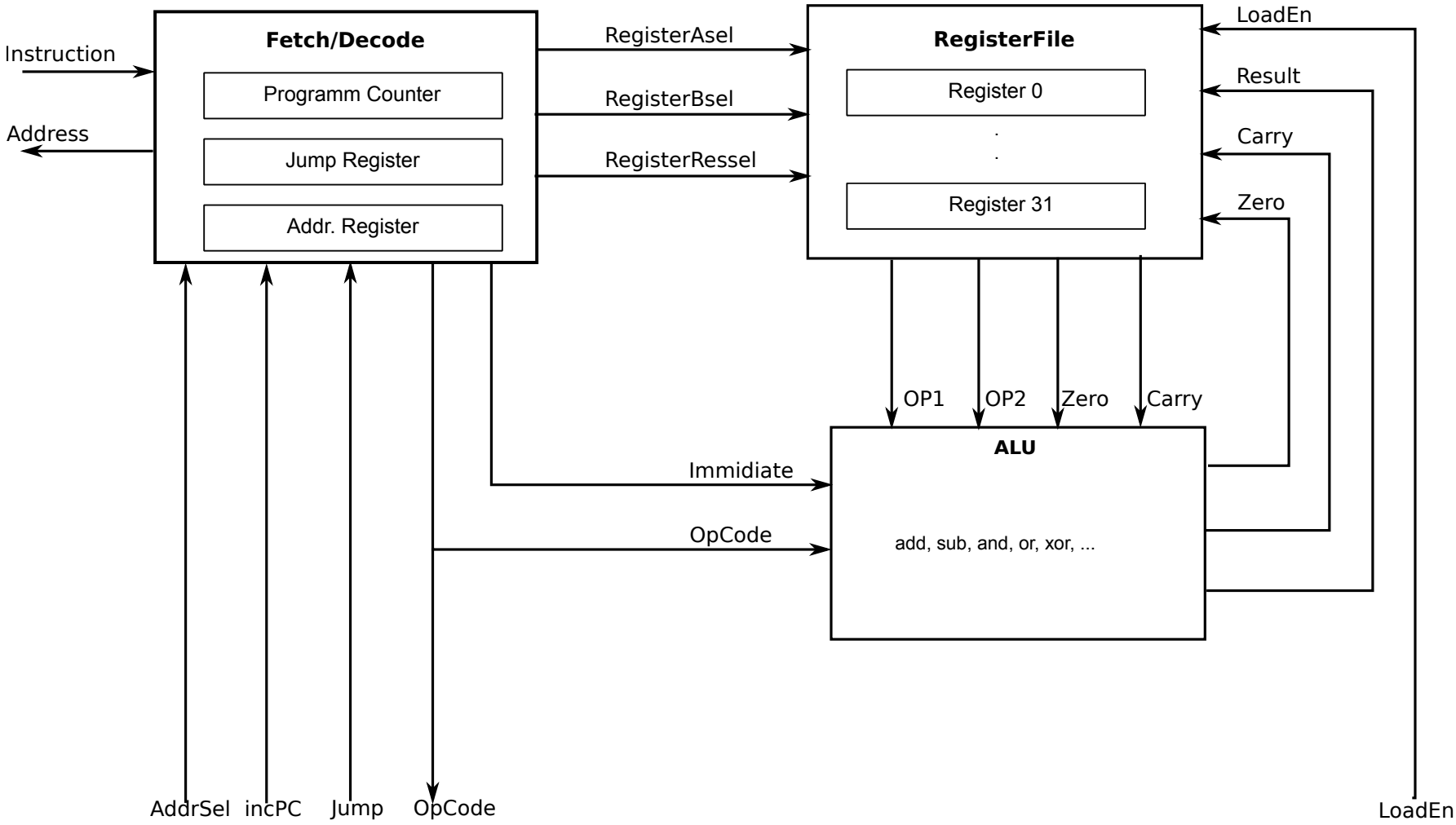
Assembling + RAM Image

Address	Data	Comment
0	001111_00001_00000_00000_00000000101	li $1, 5
1	001111_00010_00000_00000_00000000110	li $2, 6
2	001111_00011_00000_00000_00000000000	li $3, 0
3	001111_00100_00000_00000_00000000001	li $4, 1
4	000100_00101_00000_00010_00000000000	add $5, $0, $2
5	000100_00011_00011_00001_00000000000	add $3, $3, $1
6	000101_00101_00101_00100_00000000000	sub $5, $5, $4
7	001100_00000_00000_00000_00000001001	jpz end
8	001110_00000_00000_00000_00000000101	jmp loop
9	111111_00000_00000_00000_00000000000	hlt

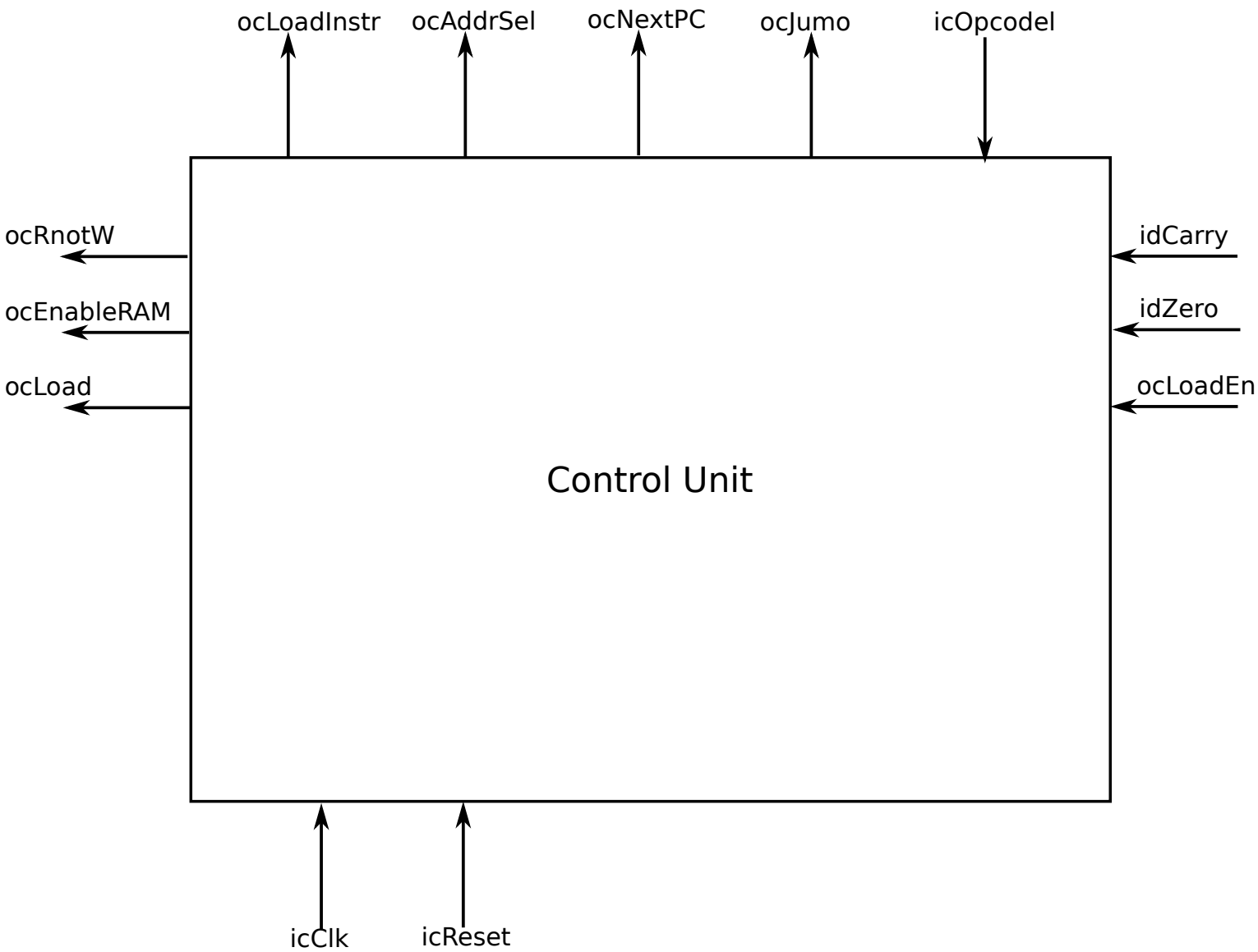
Random Access Memory (RAM)



Datapath


Instruction
Address
Fetch/Decode
Programm Counter
Jump Register
Addr. Register
RegisterAsel
RegisterBsel
RegisterRessel
RegisterFile
Register 0
Register 31
LoadEn
Result
Carry
Zero
OP1
OP2
Zero
Carry
ALU
add, sub, and, or, xor, ...
Immidiate
OpCode
AddrSel
incPC
Jump
OpCode
LoadEn

Control Unit



$$FSM_{Moore} = (S, I, O, f_t, f_o, z_0)$$

S: set of states

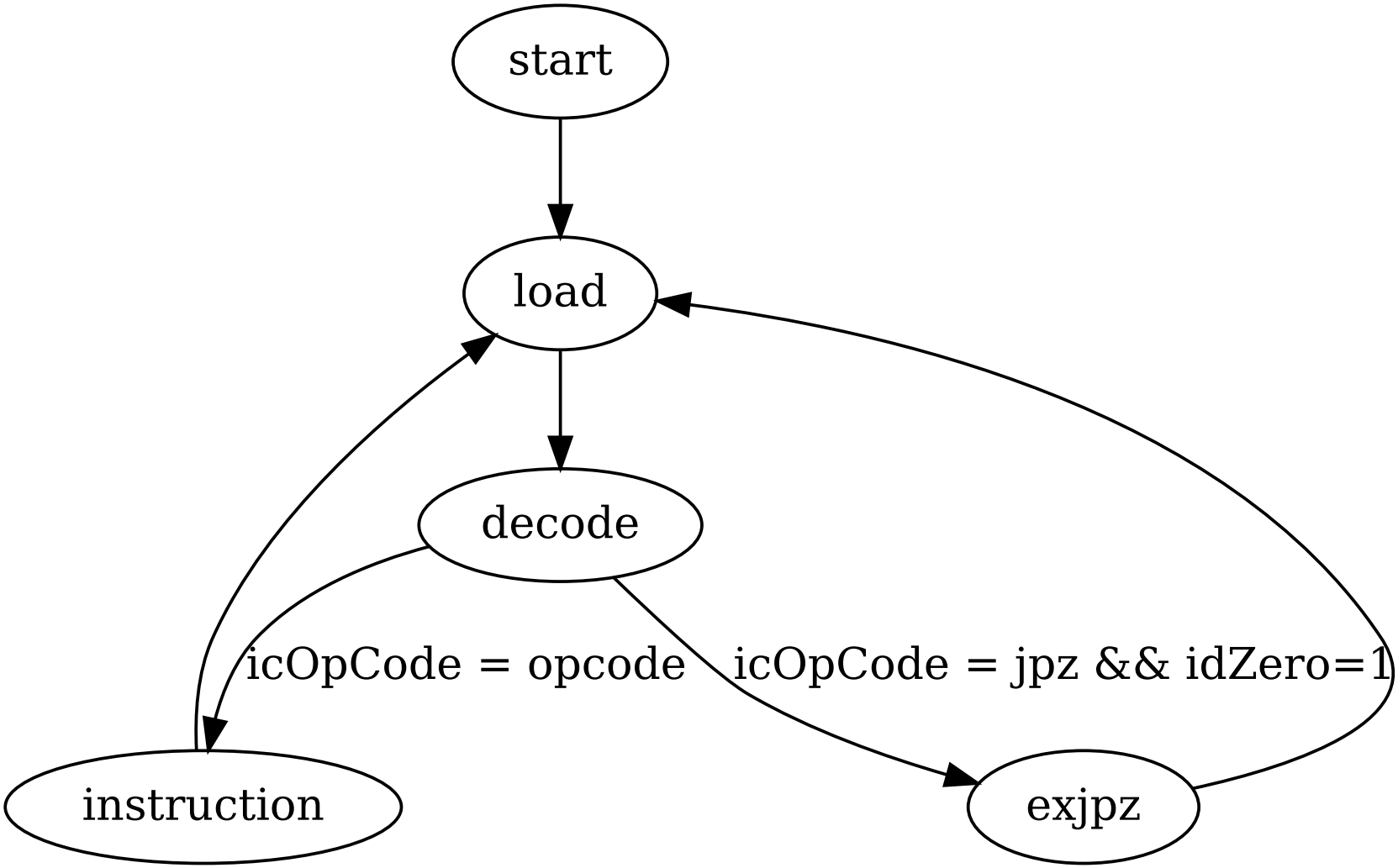
I: set of input signals

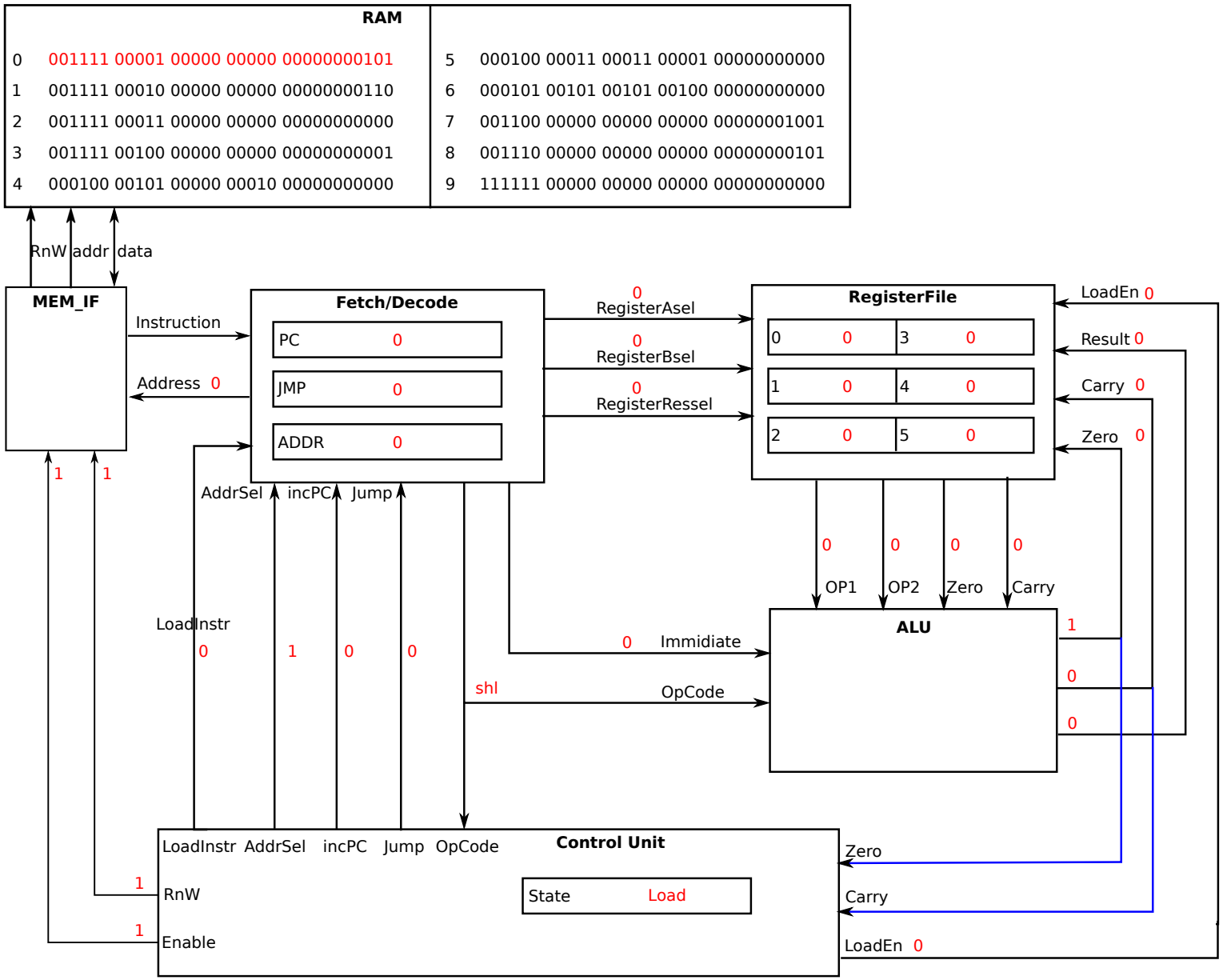
O: set of output signals

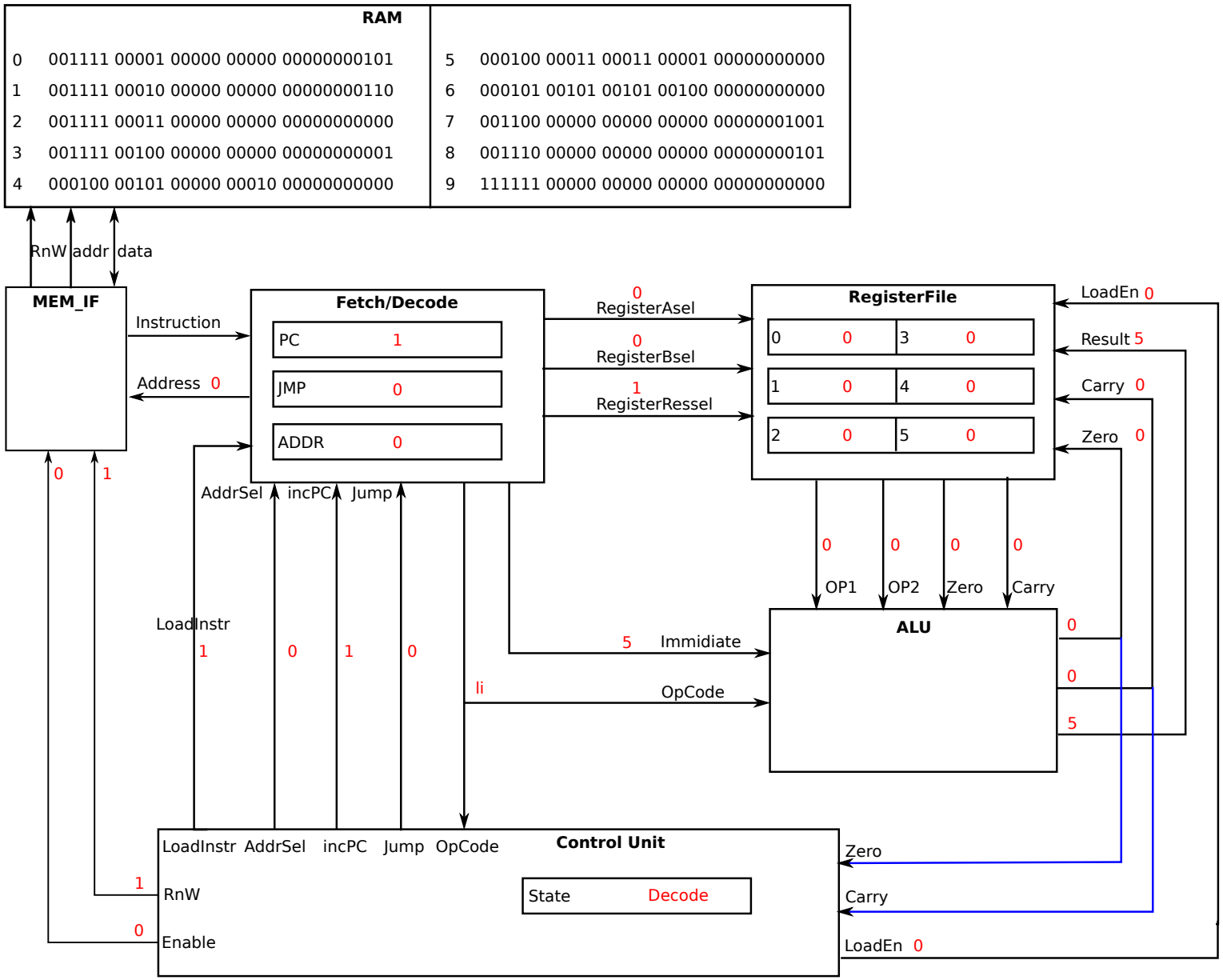
f_t : $S \times I \Rightarrow Z$

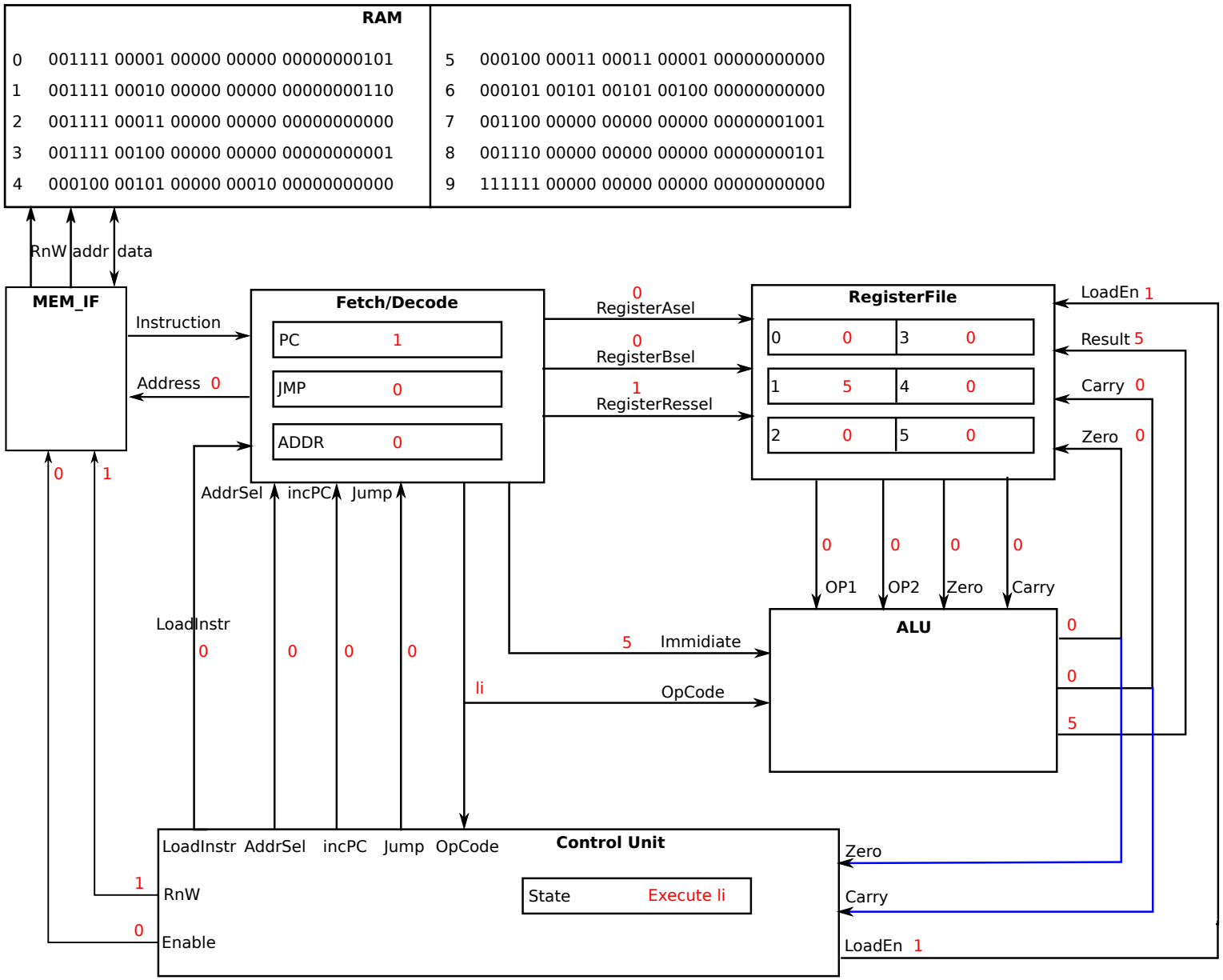
f_o : $S \Rightarrow O$

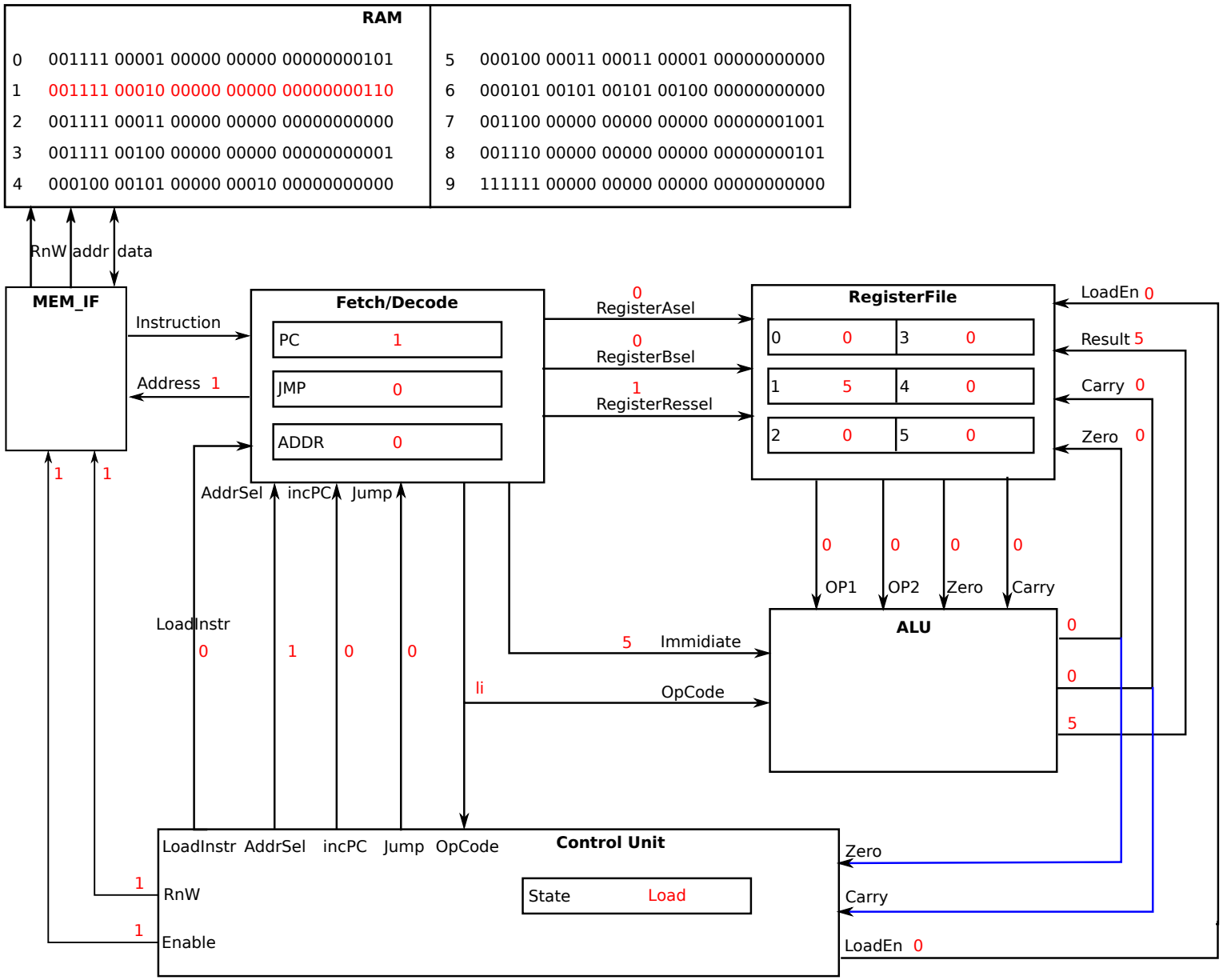
s_0 : the start state of the state machine

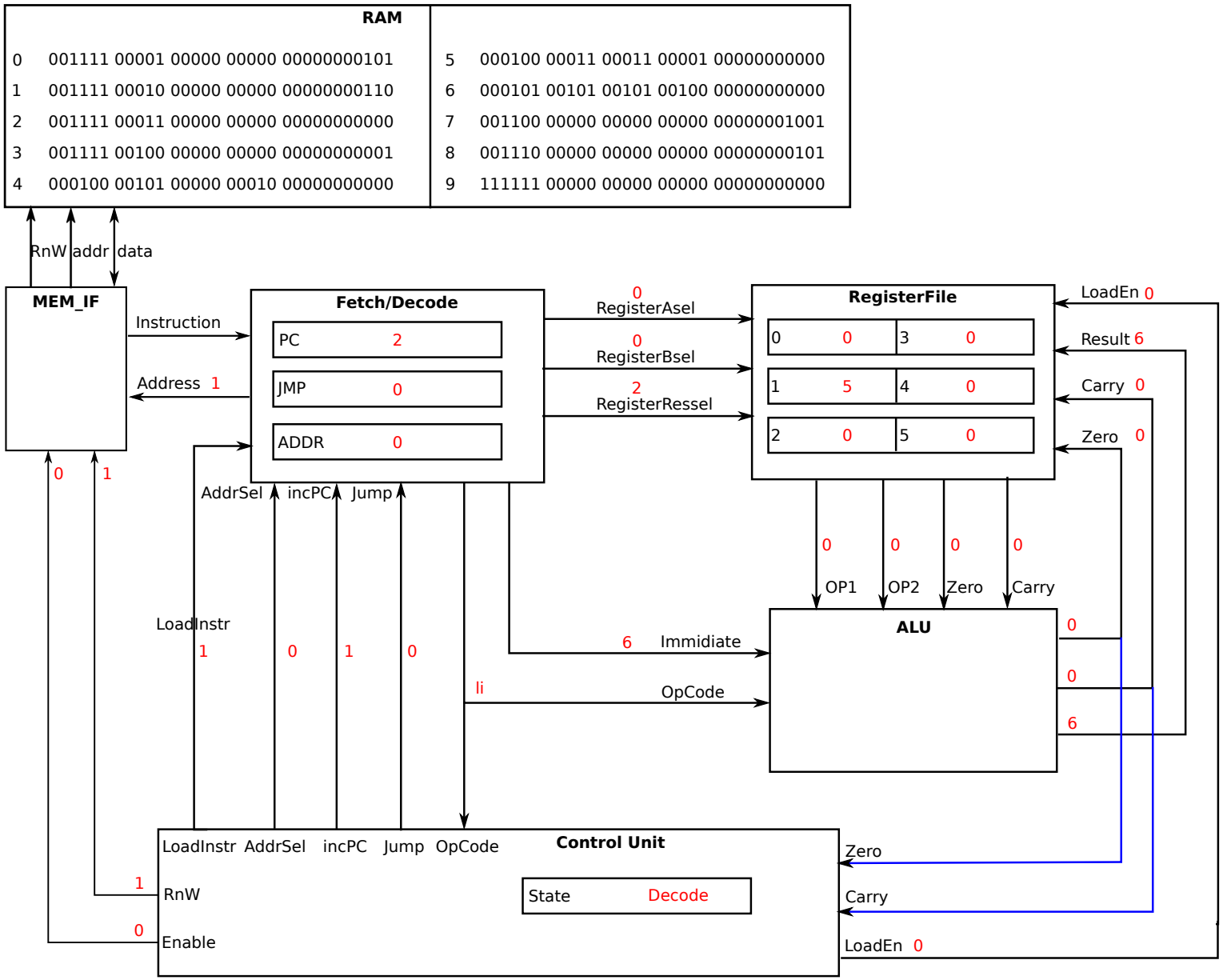

start
load
decode
icOpCode = opcode
icOpCode = jpz && idZero=1
instruction
exjpz

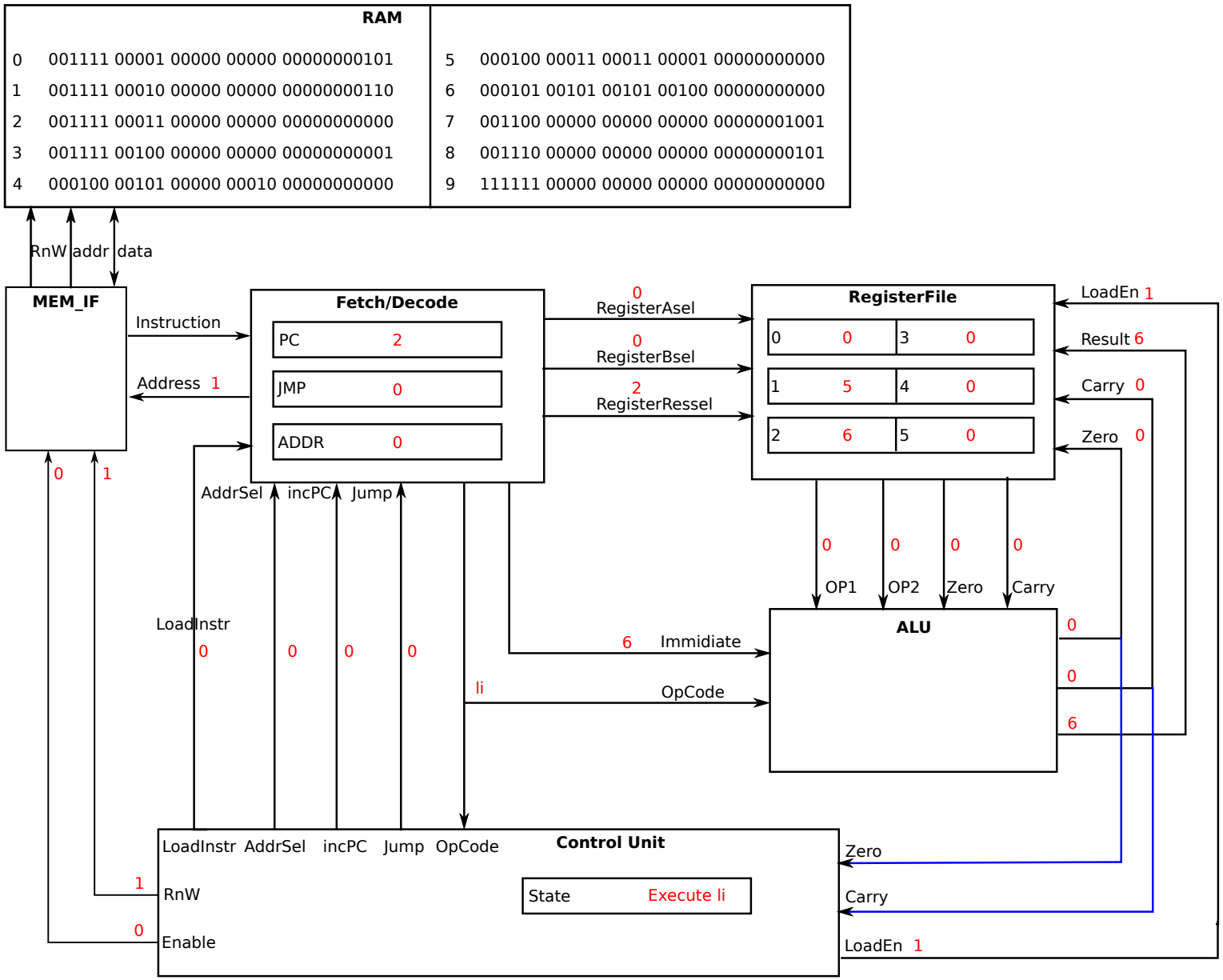

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 0
Fetch/Decode
PC 0
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
0 RegisterRessel
RegisterFile
0 0 3 0
1 0 4 0
2 0 5 0
LoadEn 0
Result 0
Carry 0
Zero 0
1 1
AddrSel incPC Jump
0 0 0 0
OP1 OP2 Zero Carry
ALU
LoadInstr
0 1 0 0
0 Immidiate
shl OpCode
1
0
0
Control Unit
LoadInstr AddrSel incPC Jump OpCode
State Load
1 RnW
1 Enable
Zero
Carry
LoadEn 0

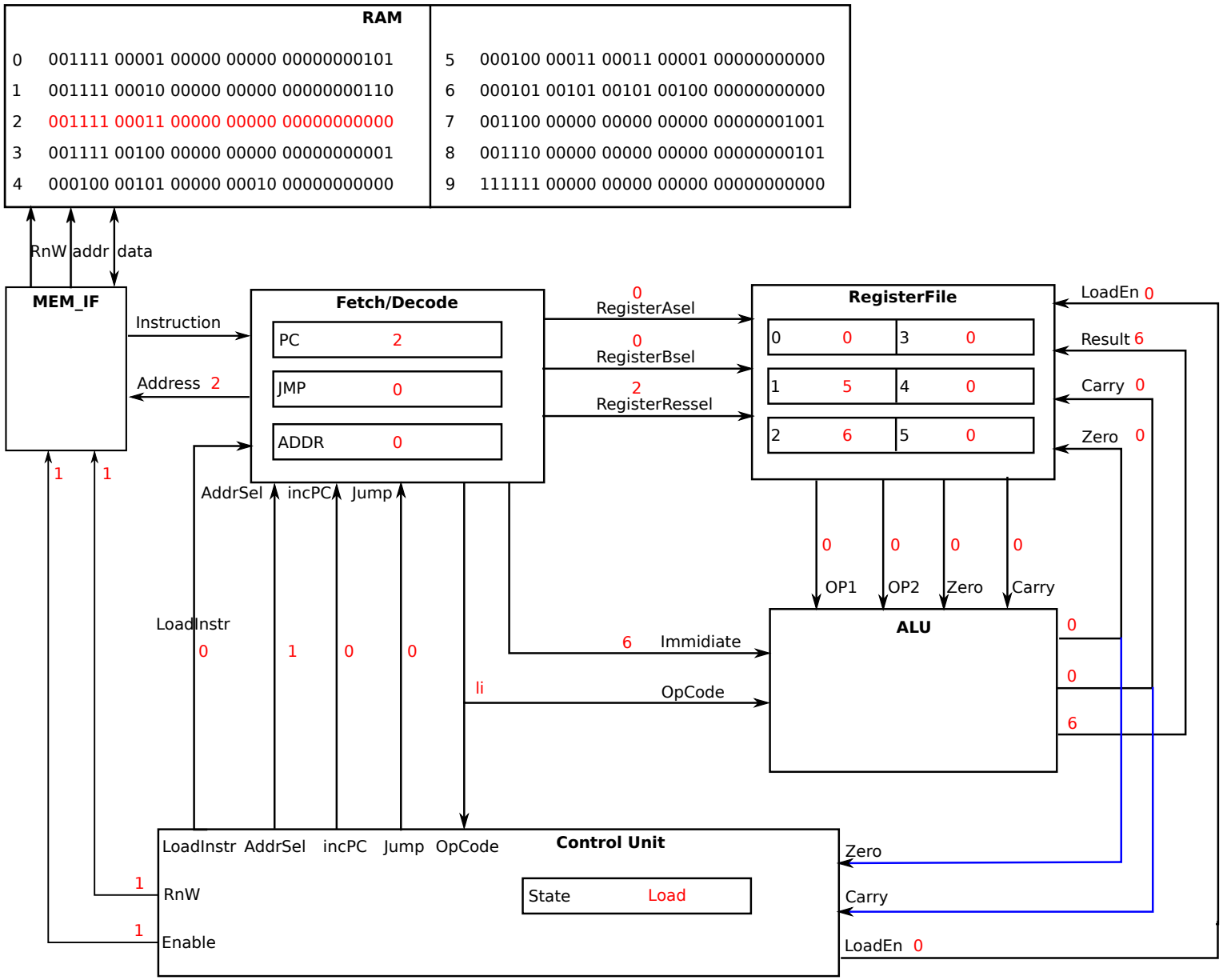

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 0
Fetch/Decode
PC 1
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
1 RegisterRessel
RegisterFile
0 0 3 0
1 0 4 0
2 0 5 0
LoadEn 0
Result 5
Carry 0
Zero 0
0 1
AddrSel incPC Jump
LoadInstr
1 0 1 0
0 0 0 0
OP1 OP2 Zero Carry
ALU
5 Immidiate
li OpCode
0
0
5
LoadInstr AddrSel incPC Jump OpCode
Control Unit
State Decode
1 RnW
0 Enable
Zero
Carry
LoadEn 0

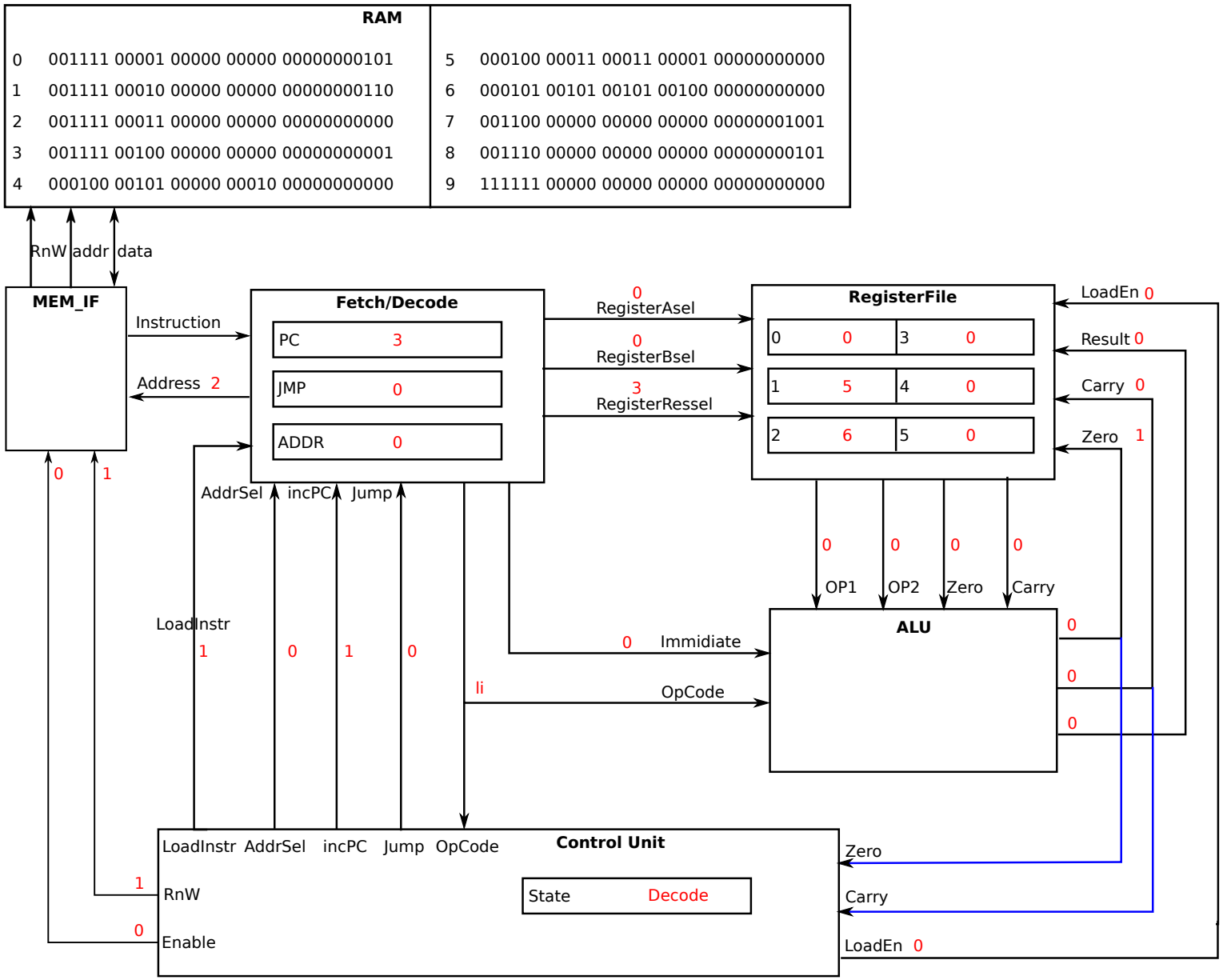

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 0
Fetch/Decode
PC 1
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
1 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 0 5 0
LoadEn 1
Result 5
Carry 0
Zero 0
0 1
AddrSel incPC Jump
LoadInstr
0 0 0 0
0 0 0 0
OP1 OP2 Zero Carry
ALU
5 Immidiate
li OpCode
0
0
5
LoadInstr AddrSel incPC Jump OpCode
Control Unit
State Execute li
1 RnW
0 Enable
Zero
Carry
LoadEn 1

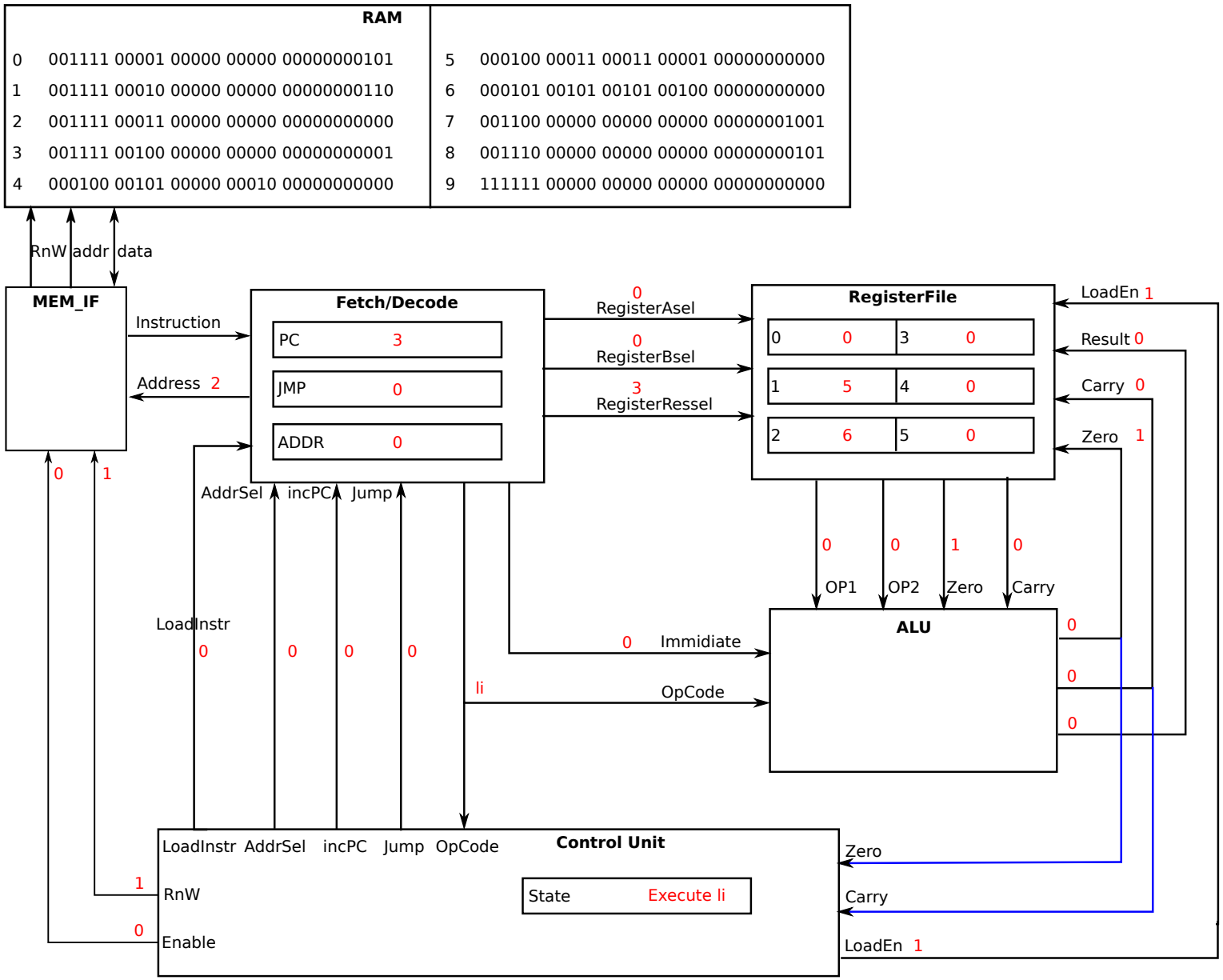

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 1
Fetch/Decode
PC 1
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
1 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 0 5 0
LoadEn 0
Result 5
Carry 0
Zero 0
1 1
AddrSel incPC Jump
0 0 0 0
OP1 OP2 Zero Carry
ALU
LoadInstr
0 1 0 0
5 Immidiate
Ii
OpCode
0
0
5
LoadInstr AddrSel incPC Jump OpCode
Control Unit
State Load
1 RnW
1 Enable
Zero
Carry
LoadEn 0

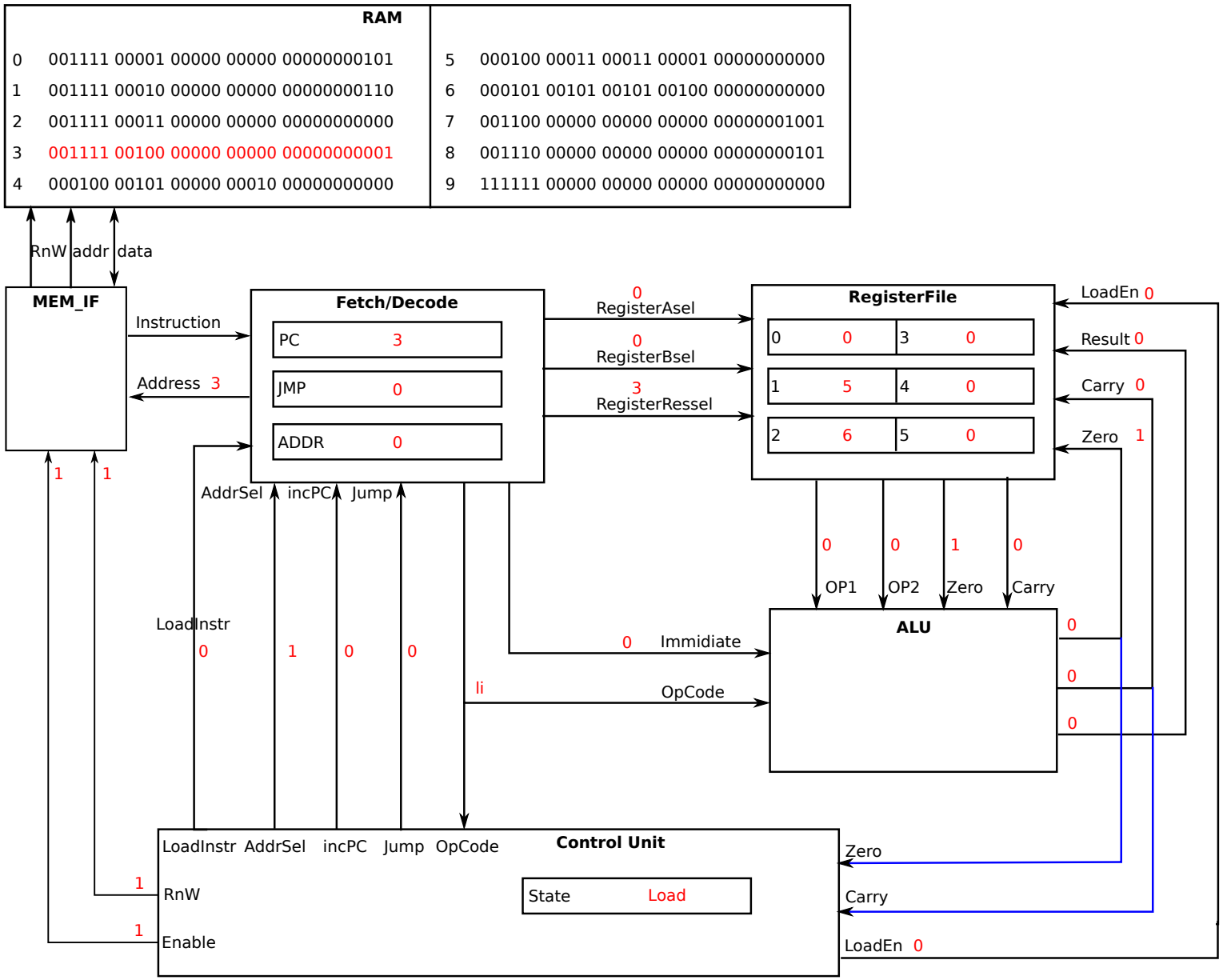

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 1
Fetch/Decode
PC 2
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
2 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 0 5 0
LoadEn 0
Result 6
Carry 0
Zero 0
0 1
AddrSel incPC Jump
LoadInstr
1 0 1 0
0 0 0 0
OP1 OP2 Zero Carry
ALU
6 Immidiate
li OpCode
0
0
6
LoadInstr AddrSel incPC Jump OpCode
Control Unit
State Decode
1 RnW
0 Enable
Zero
Carry
LoadEn 0

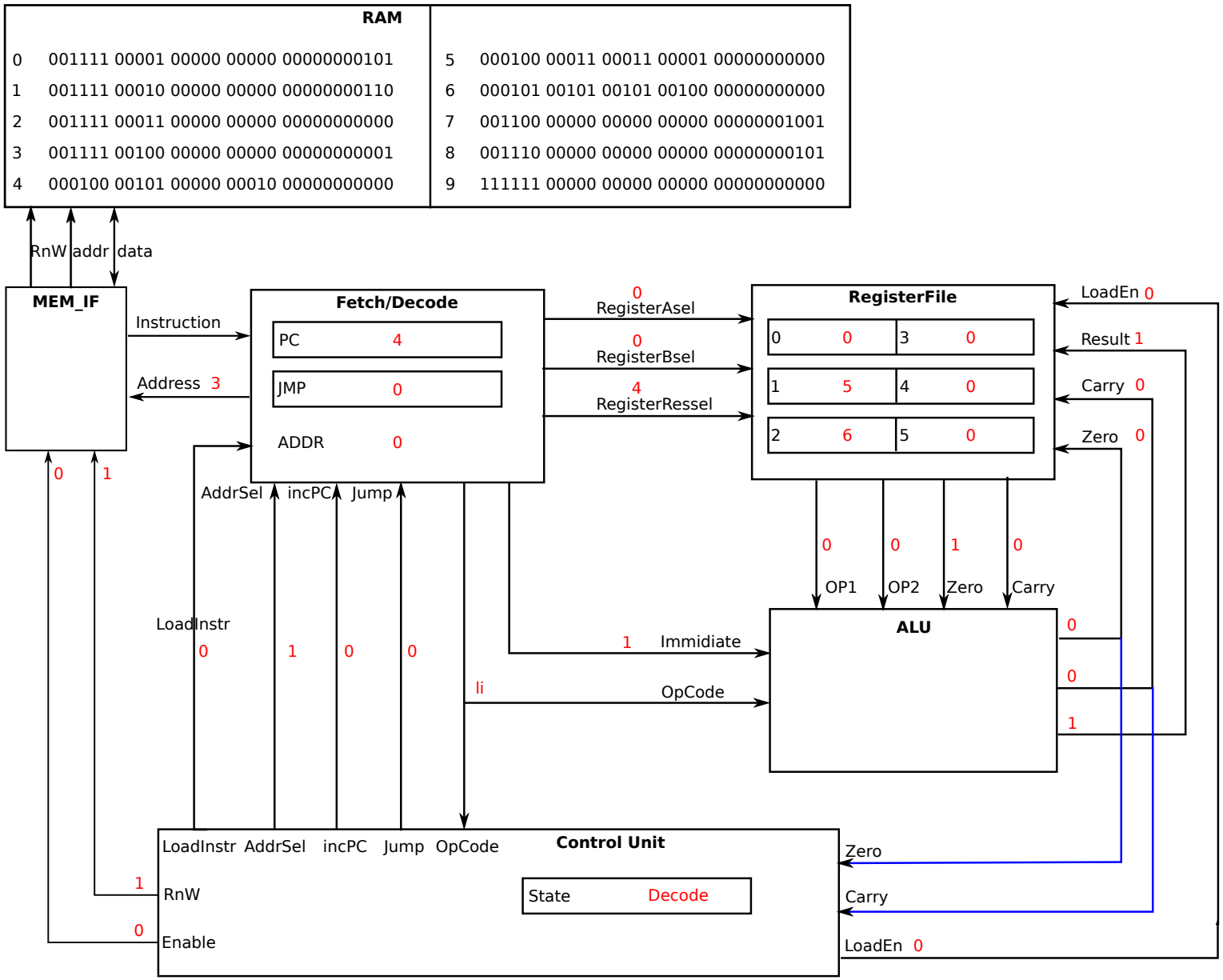

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 1
Fetch/Decode
PC 2
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
2 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 6 5 0
LoadEn 1
Result 6
Carry 0
Zero 0
0 1
AddrSel incPC Jump
0 0 0 0
LoadInstr
0 0 0 0
OP1 OP2 Zero Carry
ALU
6 Immidiate
li OpCode
0
0
6
LoadInstr AddrSel incPC Jump OpCode
Control Unit
State Execute li
1 RnW
0 Enable
Zero
Carry
LoadEn 1

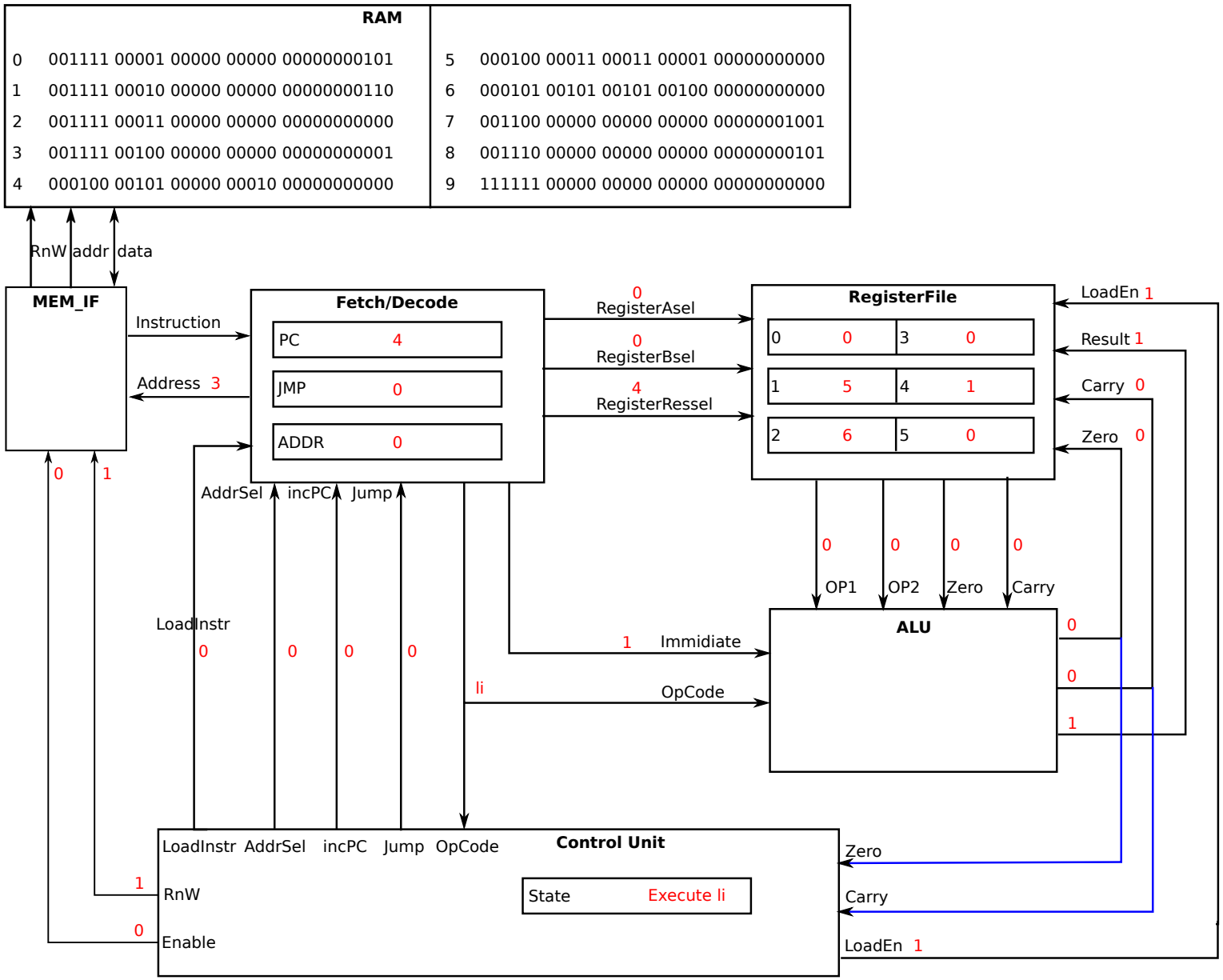

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 2
Fetch/Decode
PC 2
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
2 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 6 5 0
LoadEn 0
Result 6
Carry 0
Zero 0
1 1
AddrSel incPC Jump
0 0 0 0
OP1 OP2 Zero Carry
ALU
LoadInstr
0 1 0 0
6 Immidiate
li OpCode
0
0
6
Control Unit
LoadInstr AddrSel incPC Jump OpCode
State Load
1 RnW
1 Enable
Zero
Carry
LoadEn 0

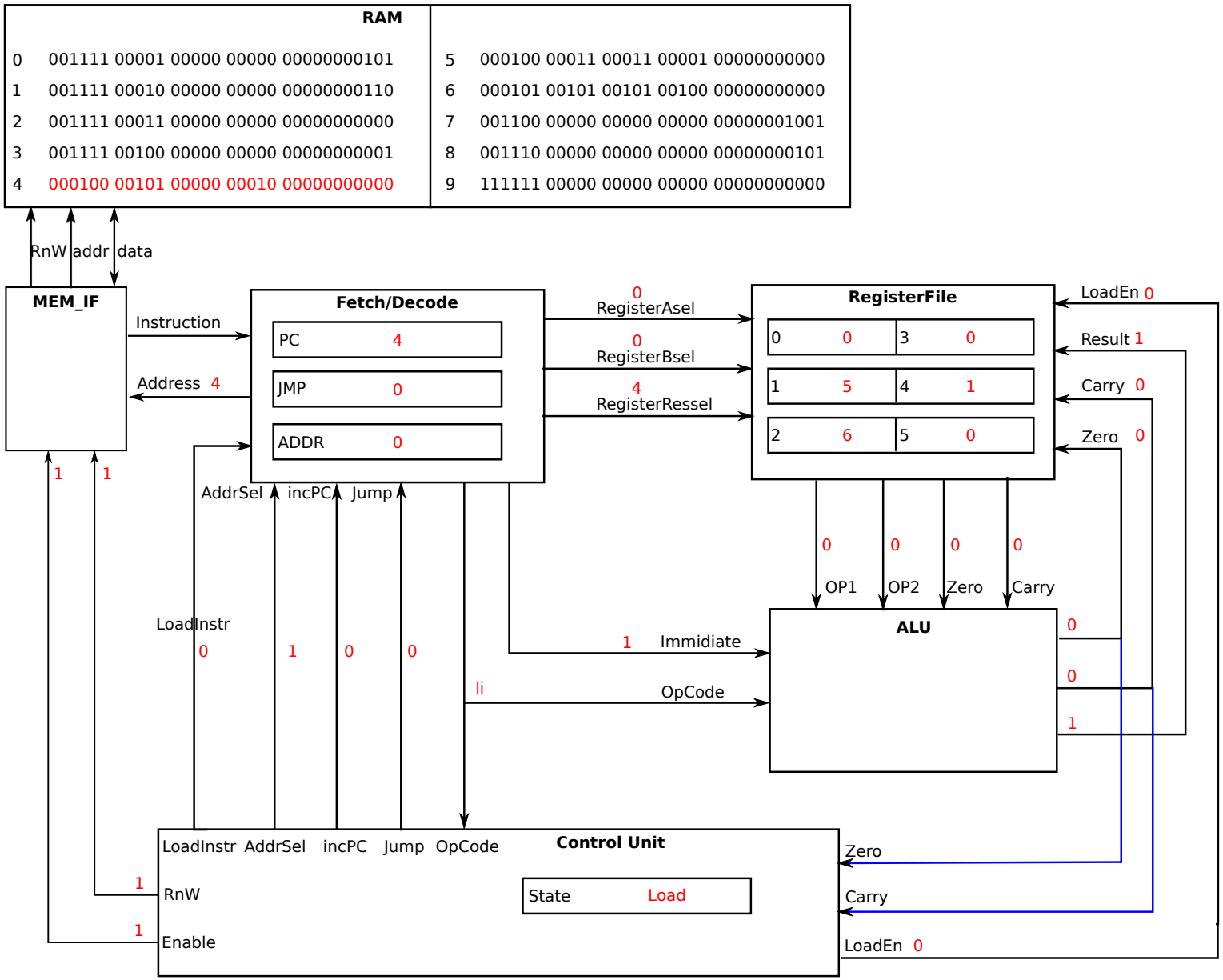

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 2
Fetch/Decode
PC 3
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
3 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 6 5 0
LoadEn 0
Result 0
Carry 0
Zero 1
0 1
AddrSel incPC Jump
LoadInstr
1 0 1 0
0 Immidiate
Ii
OpCode
0 0 0 0
OP1 OP2 Zero Carry
ALU
0 0 0
Control Unit
LoadInstr AddrSel incPC Jump OpCode
State Decode
1 RnW
0 Enable
Zero
Carry
LoadEn 0

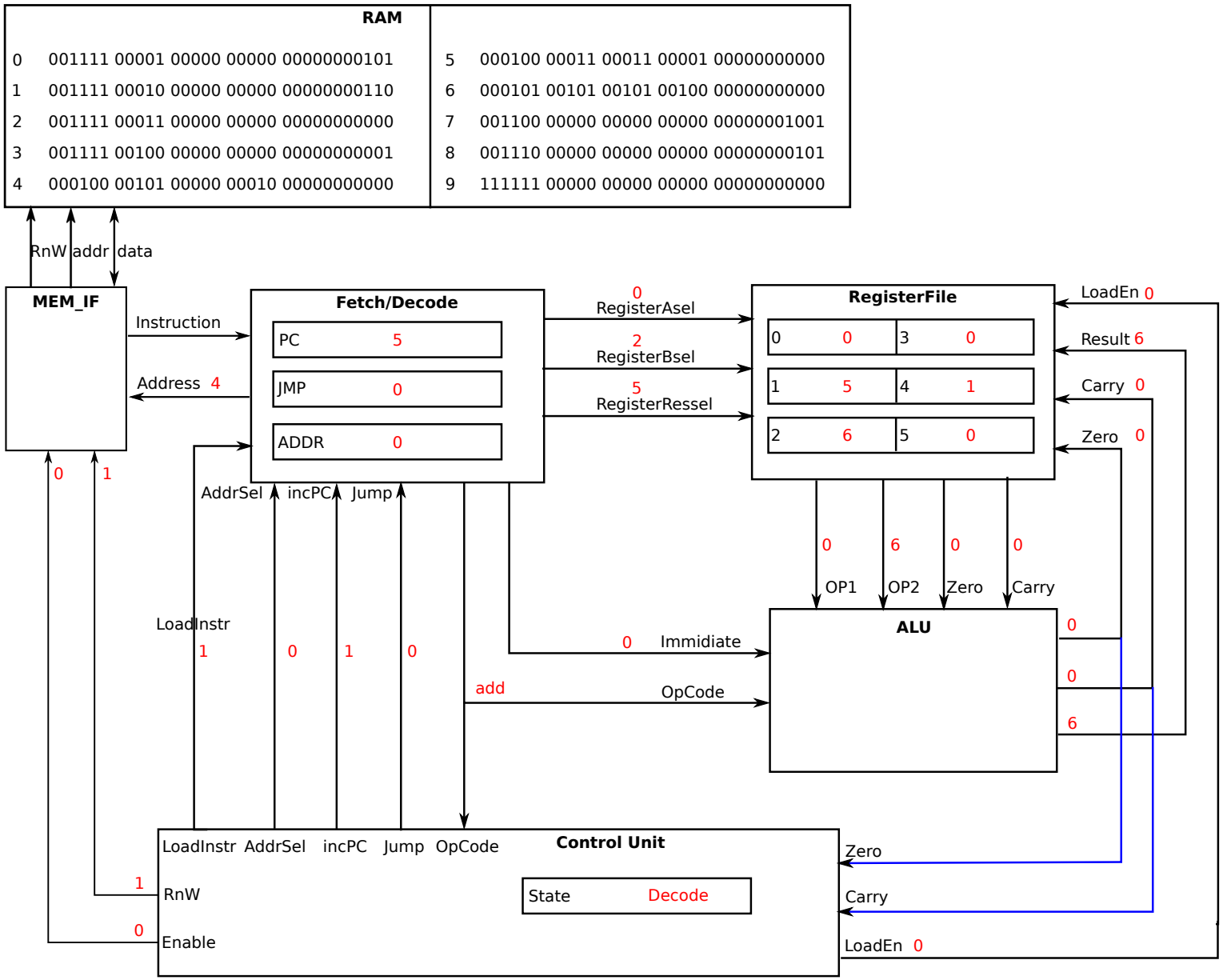

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 2
Fetch/Decode
PC 3
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
3 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 6 5 0
LoadEn 1
Result 0
Carry 0
Zero 1
0 1
AddrSel incPC Jump
0 0 1 0
OP1 OP2 Zero Carry
ALU
LoadInstr
0 0 0 0
0 Immidiate
li
OpCode
0
0
0
Control Unit
LoadInstr AddrSel incPC Jump OpCode
State Execute li
1 RnW
0 Enable
Zero
Carry
LoadEn 1

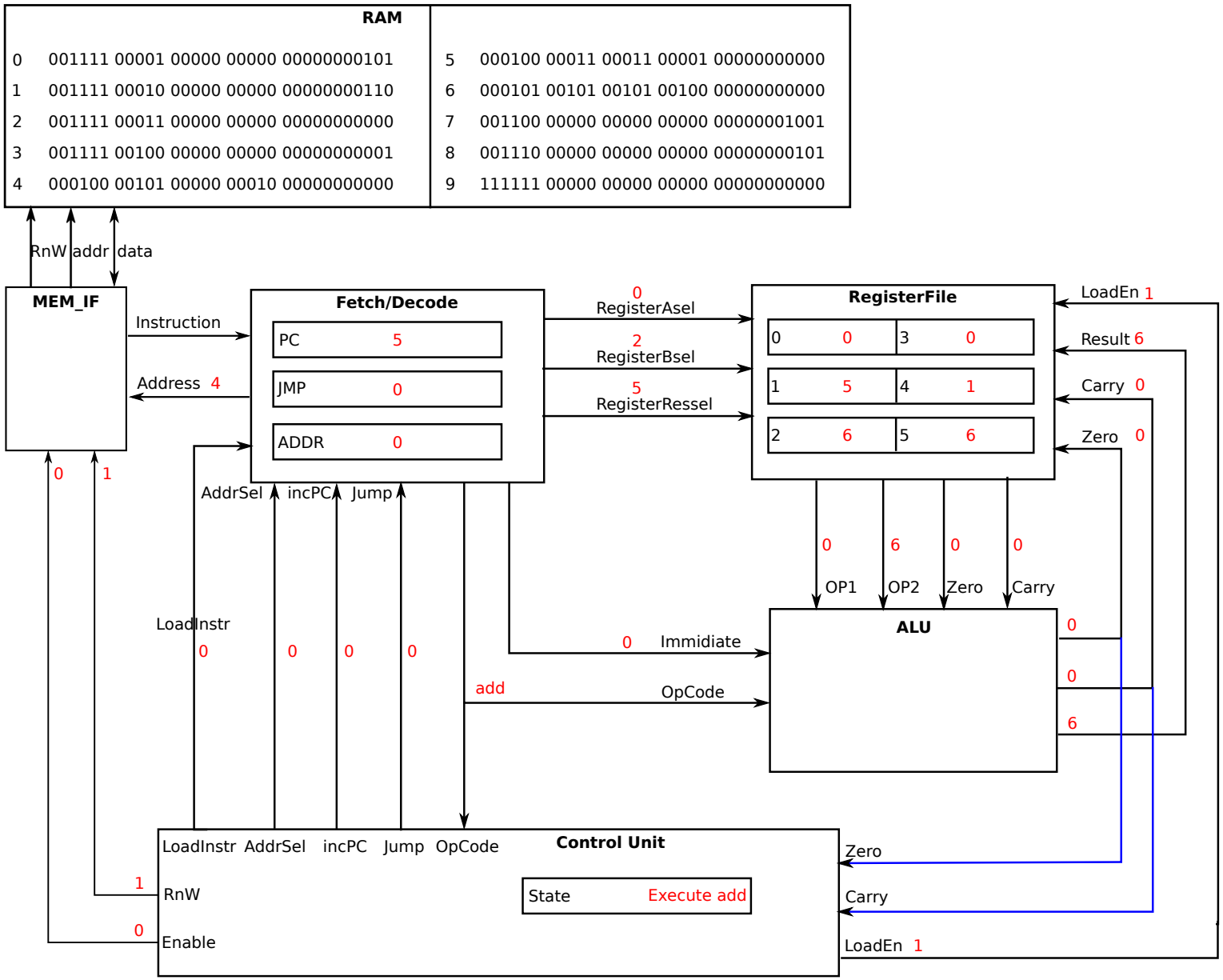

RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 3
Fetch/Decode
PC 3
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
3 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 6 5 0
LoadEn 0
Result 0
Carry 0
Zero 1
1 1
AddrSel incPC Jump
0 1 0 0
LoadInstr
0 Immidiate
Ii
OpCode
0 0 1 0
OP1 OP2 Zero Carry
ALU
0
0
0
Control Unit
LoadInstr AddrSel incPC Jump OpCode
State Load
1 RnW
1 Enable
Zero
Carry
LoadEn 0


RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 3
Fetch/Decode
PC 4
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
4 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 0
2 6 5 0
LoadEn 0
Result 1
Carry 0
Zero 0
0 1
AddrSel incPC Jump
0 0 1 0
OP1 OP2 Zero Carry
LoadInstr
1 Immidiate
li OpCode
ALU
0
0
1
Control Unit
LoadInstr AddrSel incPC Jump OpCode
State Decode
1 RnW
0 Enable
Zero
Carry
LoadEn 0


RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 3
Fetch/Decode
PC 4
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
4 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 1
2 6 5 0
LoadEn 1
Result 1
Carry 0
Zero 0
0 1
AddrSel incPC Jump
0 0 0 0
OP1 OP2 Zero Carry
LoadInstr
1 Immidiate
li OpCode
ALU
0
0
1
Control Unit
LoadInstr AddrSel incPC Jump OpCode
State Execute li
1 RnW
0 Enable
Zero
Carry
LoadEn 1


RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 4
Fetch/Decode
PC 4
JMP 0
ADDR 0
0 RegisterAsel
0 RegisterBsel
4 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 1
2 6 5 0
LoadEn 0
Result 1
Carry 0
Zero 0
1 1
AddrSel incPC Jump
0 0 0 0
OP1 OP2 Zero Carry
ALU
LoadInstr
0 1 0 0
1 Immidiate
Ii
OpCode
0
0
1
LoadInstr AddrSel incPC Jump OpCode
Control Unit
State Load
1 RnW
1 Enable
Zero
Carry
LoadEn 0


RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 4
Fetch/Decode
PC 5
JMP 0
ADDR 0
0 RegisterAsel
2 RegisterBsel
5 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 1
2 6 5 0
LoadEn 0
Result 6
Carry 0
Zero 0
0 1
AddrSel incPC Jump
0 6 0 0
OP1 OP2 Zero Carry
ALU
LoadInstr
1 0 1 0
0 Immidiate
add OpCode
0
0
6
LoadInstr AddrSel incPC Jump OpCode
Control Unit
State Decode
1 RnW
0 Enable
Zero
Carry
LoadEn 0


RAM
0 001111 00001 00000 00000 00000000101
1 001111 00010 00000 00000 00000000110
2 001111 00011 00000 00000 00000000000
3 001111 00100 00000 00000 00000000001
4 000100 00101 00000 00010 00000000000
5 000100 00011 00011 00001 00000000000
6 000101 00101 00101 00100 00000000000
7 001100 00000 00000 00000 00000001001
8 001110 00000 00000 00000 00000000101
9 111111 00000 00000 00000 00000000000
RnW addr data
MEM_IF
Instruction
Address 4
Fetch/Decode
PC 5
JMP 0
ADDR 0
0 RegisterAsel
2 RegisterBsel
5 RegisterRessel
RegisterFile
0 0 3 0
1 5 4 1
2 6 5 6
LoadEn 1
Result 6
Carry 0
Zero 0
0 1
AddrSel incPC Jump
0 6 0 0
OP1 OP2 Zero Carry
ALU
LoadInstr
0 0 0 0
0 Immidiate
add OpCode
0
0
6
LoadInstr AddrSel incPC Jump OpCode
Control Unit
State Execute add
1 RnW
0 Enable
Zero
Carry
LoadEn 1

Implementation

Example Implementations of Components

Control Unit

```
case sState is
when load       =>
    sState_next     <= decode;
when decode     =>
    case icOpCode is
            when shl        => sState_next <= exshl;
            when shr        => sState_next <= exshr;
            when sto        => sState_next <= exsto;
            when loa        => sState_next <= exloa;
            when li         => sState_next <= exli;
            when add        => sState_next <= exadd;
            when sub        => sState_next <= exsub;
            when addc       => sState_next <= exaddc;
            when subc       => sState_next <= exsubc;
            when opand      => sState_next <= exopand;
            when opor       => sState_next <= exopor;
            when opxor      => sState_next <= exopxor;
            when opnot      => sState_next <= exopnot;
```

```
case sState is

    when load       =>
                ocRnotWRam      <= '1'; -- read from RAM
                ocLoadEn        <= '0'; -- do not save result
                ocEnableRAM     <= '1'; -- enable RAM
                ocLoadInstr     <= '0'; -- load instruction
                ocNextPC        <= '0'; -- do not increment pc
                ocAddrSel       <= '1'; -- pc on addressbus
                ocJump          <= '0'; -- no ocJump

    when decode     =>
                ocRnotWRam      <= '1'; -- read from RAM
                ocLoadEn        <= '0'; -- do not save result
                ocEnableRAM     <= '0'; -- disable RAM
                ocLoadInstr     <= '1'; -- load instruction
                ocNextPC        <= '1'; -- increment pc
                ocAddrSel       <= '0'; -- pc on addressbus
                ocJump          <= '0'; -- no ocJump
```

Register File

```
process(iClk, iReset)
begin
    if (iReset = '1') then
        for i in 31 downto 0 loop
            registerFile(i) <= (others=>'0');
        end loop;

        sdCarry <= '0';
        sdZero  <= '0';
    elsif (rising_edge(iClk)) then
        if (icLoadEn = '1') then
         registerFile(to_integer(unsigned(icRegINsel))) <= idDataIn;
         sdCarry <= idCarryIn;
         sdZero  <= idZeroIn;
        end if;
    end if;
end process;

odRegA      <= registerFile(to_integer(unsigned(icRegAsel)));
odRegB      <= registerFile(to_integer(unsigned(icRegBsel)));
odCarryOut  <= sdCarry;
odZeroOut   <= sdZero;
```

- This Processor Core
- 30Mhz on Virtex5 FPGA
- self written Assembler (Perl)

- Partial Reconfigurable Heterogenous System
- Arm810 (Armv4, no Thumb)
- 100 MHz (120Mhz max)
- 40 BogoMips
- 120 MB/s read from RAM
- 95 MB/s write to RAM

- can contain backdoors
- can contain hidden instructions
- not detectable, because only absence of a feature can be detected

Why should i build my own processor core ?

WHY ?

Why should i build my own processor core ?

- you know the code
- you can find backdoors
- you can make it open source
- you can add crypto functions
- you can add everything you want

Conclusion

- everyone can write his/her own processor core
- hard cores could be compromised
- start writing open source processor cores

Thanks for your attention!
Questions ?

If anyone is interested contact me !