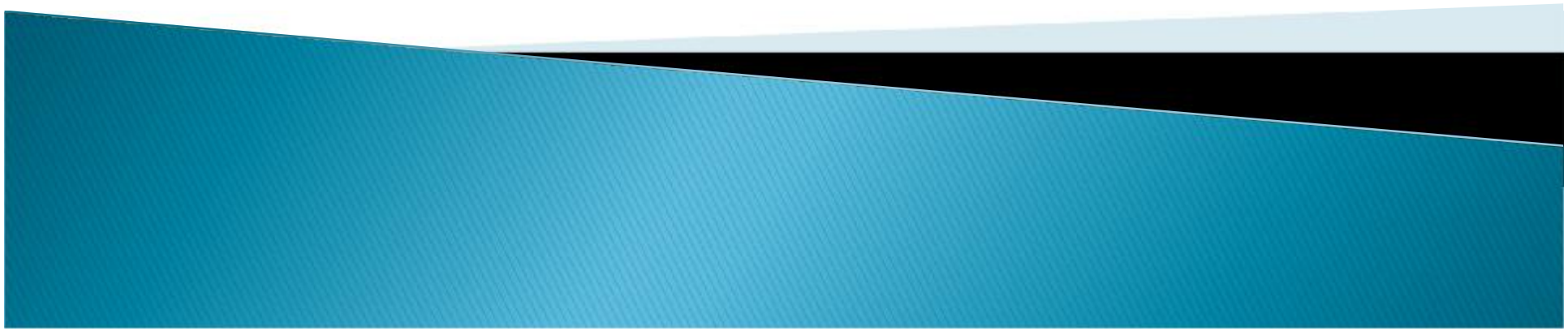
计算机系统 I

第四章

冯.诺伊曼模型



计算机系统的抽象层次

Problem Specification

compute the fibonacci sequence

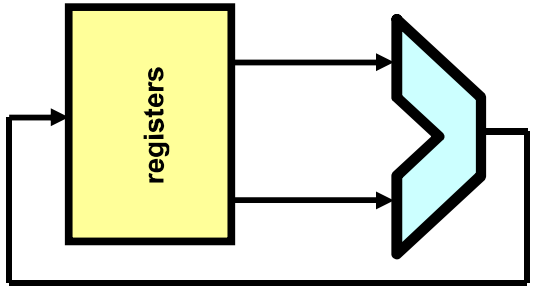
Algorithm Program

```
for(i=2; i<100; i++) {
  a[i] = a[i-1]+a[i-2];}
```

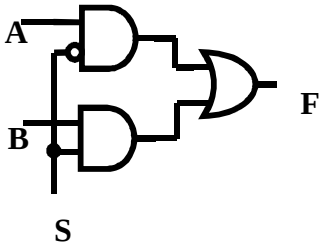
ISA (Instruction Set Architecture)

```
load r1, a[i];
add  r2, r2, r1;
```

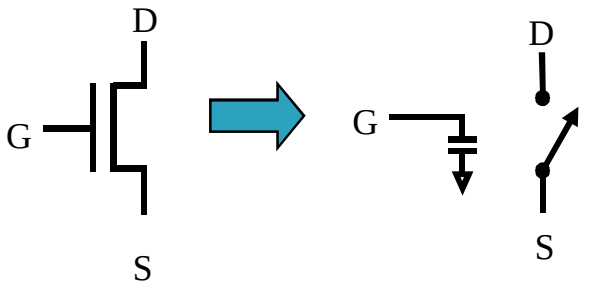
microArchitecture



Logic



Transistors



Physics/Chemistry

计算机系统微结构

- 计算机执行任务应该分成几个部件，每个部件如何协调工作。每个部件如何采用逻辑电路设计。
- 抽象层次：电路级--》部件级
- 冯.诺伊曼最早将二进制引入计算机应用，并定义了计算机的五大组成部件
- 冯.诺伊曼模型：计算机处理和程序执行的基础模型

冯·诺依曼机的特征

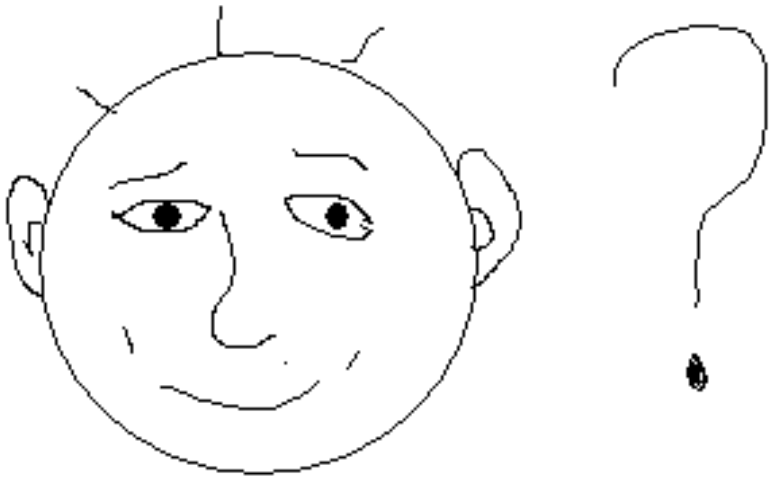
(1)计算机应由运算器、控制器、存储器、输入设备和输出设备五大部件组成；

(2)计算机中采用二进制来表示指令和数据；

(3)采用存储程序方式，计算机能自动逐条取出指令并执行程序。

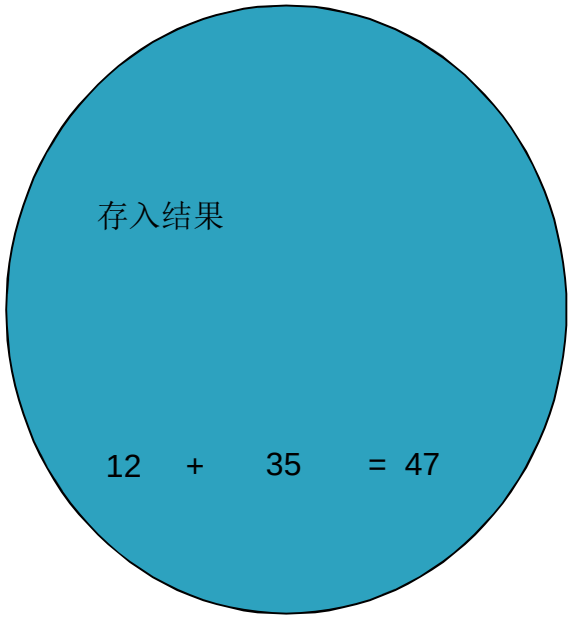
$(3 \times 4) + (5 \times 7)$?

- $3 \times 4 = 12$
- $5 \times 7 = 35$
- $12 + 35 = 47$



内 存

CPU



3 4 5 7

12 35 47

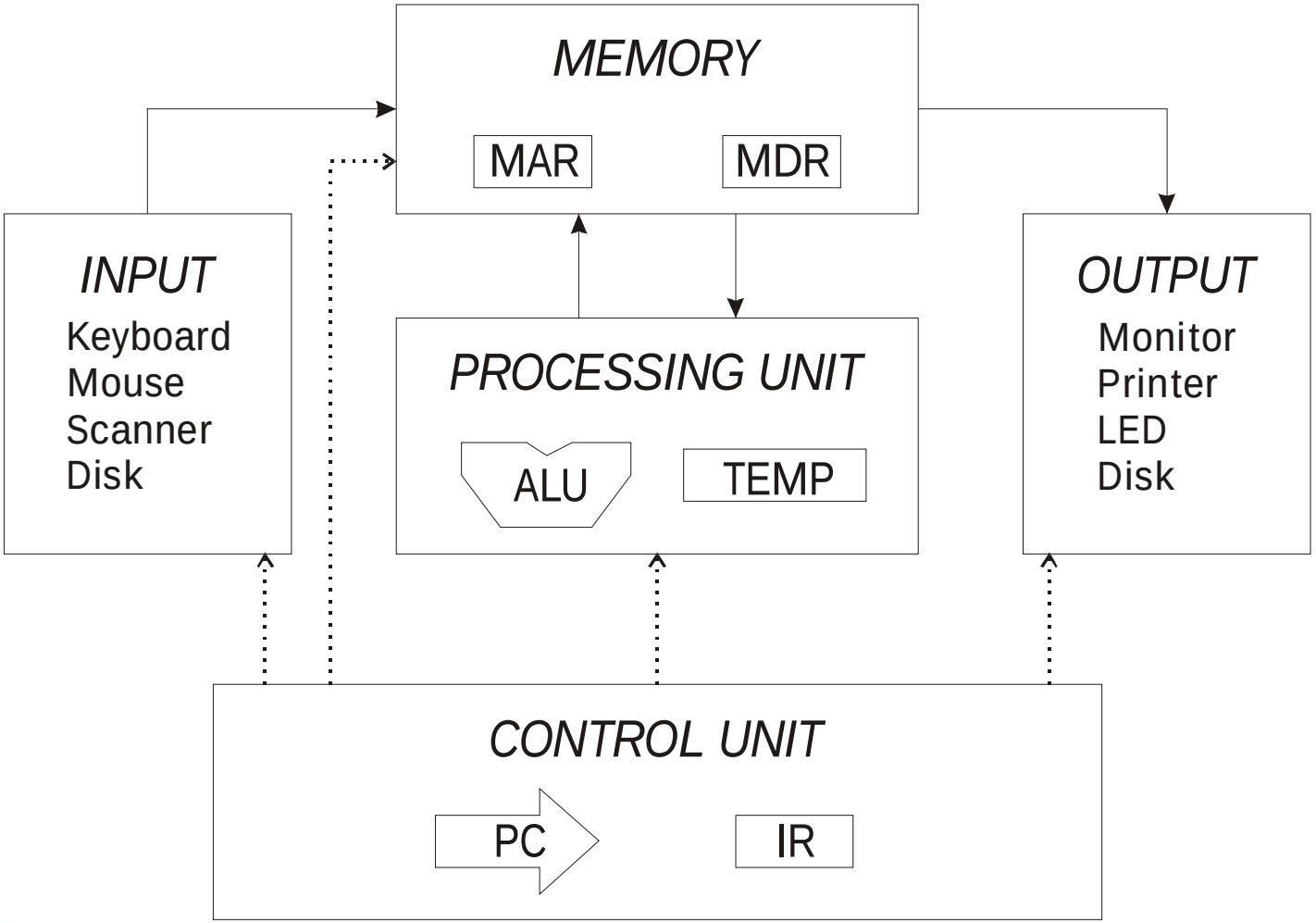
- 读取 3
- 读取 4
- 两数相乘
- 存入结果1
- 读取 5
- 读取 7
- 两数相乘
- 存入结果2
- 读取结果1
- 读取结果2
- 两数相加
- 存入结果

冯诺伊曼结构：数据和程序都在存储器中，CPU从内存中取指令和数据进行运算并把结果也放到内存中

思考

- 1　运算中间结果(12,35)能不能在运算器里面存储?
- 2 运算器能不能做成支持3个操作数同时运算?
- 3 内存掉电数据就丢失了？内存的运算数据怎么来的？

冯.诺伊曼模型


MEMORY
MAR
MDR
INPUT
Keyboard
Mouse
Scanner
Disk
PROCESSING UNIT
ALU
TEMP
OUTPUT
Monitor
Printer
LED
Disk
CONTROL UNIT
PC
IR

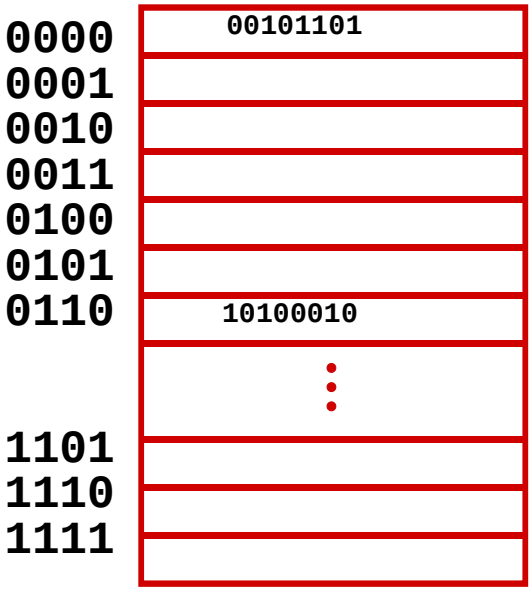
内存

- *基于门电路和锁存器2^k x m的位存储阵列。*
  寻址空间：2^k ，寻址能力：mbit
  1GB内存： 2^{30} x8bit

- 内存地址：访问需要的数据
  每个内存单元有唯一的地址（2^k个）
  每个单元存储m 位个数据

- 内存的基本操作
  LOAD
  - read a value from a memory location

  STORE
  - write a value to a memory location



内存访问接口

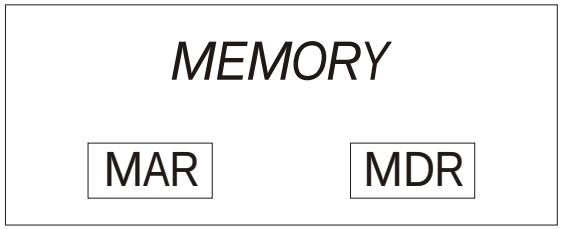
运算部件通过两个专用寄存器来访问（读写）内存的数据

内存地址寄存器MAR

MAR: Memory Address Register

内存数据寄存器

MDR: Memory Data Register



读内存某个单元（ LOAD 操作）

1）写内存单元地址(A)到MAR中.

2）发 “read”信号给内存. 内存将MAR中地址对应的数据送到MDR中准备好。

3）运算器从MDR中读取数据(X)。

写内存某个单元（STORE 操作）

1）将数据 (X) 写入 MDR.

2）将地址 (A) 写入 MAR.

3）发 “write”信号给内存

Advantages?

运算单元

功能单元(Functional Units)

ALU = Arithmetic and Logic Unit,算术逻辑运算单元

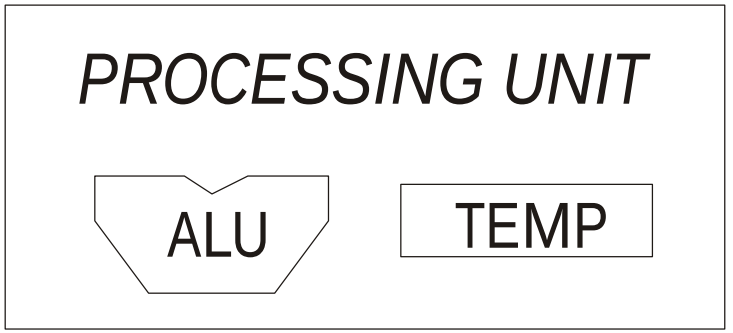
对操作数进行算术和逻辑运算。

(常规：+，-，and,or,not

或者特殊的multiply,Division,

square root, …)

LC-3 performs only ADD, AND, NOT



寄存器（临时存储单元）

用于存放操作数、运算中间结果的小容量的临时存储单元

LC-3 有8个16位(R0, …, R7)的寄存器

字长

运算器支持的操作数的最大宽度/寄存器的宽度

LC-3 是16位的

输入和输出设备

INPUT	*OUTPUT*
Keyboard Mouse Scanner Disk	Monitor Printer LED Disk

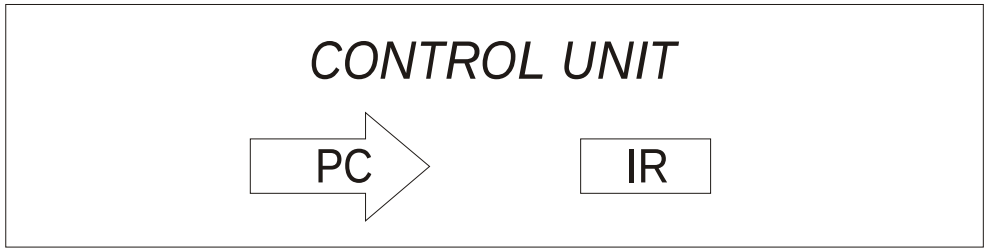
- 输入设备：提供数据给内存
  输出设备：负责从内存转换数据给用户

- 每个输入输出设备具有自己的访问接口，一般是一组专用寄存器。类似内存接口的MDR/MAR寄存器。

- LC-3 支持键盘输入：keyboard (input)和显示输出：monitor (output)
  - keyboard: 两个寄存器 data register (KBDR) / status register (KBSR)
  - monitor: 两个寄存器 data register (DDR) /status register (DSR)
- 设备可同时支持输出和输入功能
  - disk, network
- 控制设备访问的程序通常称为设备驱动程序（*driver*）.

控制单元

协同其他所有单元之间的工作。



两个重要的寄存器：

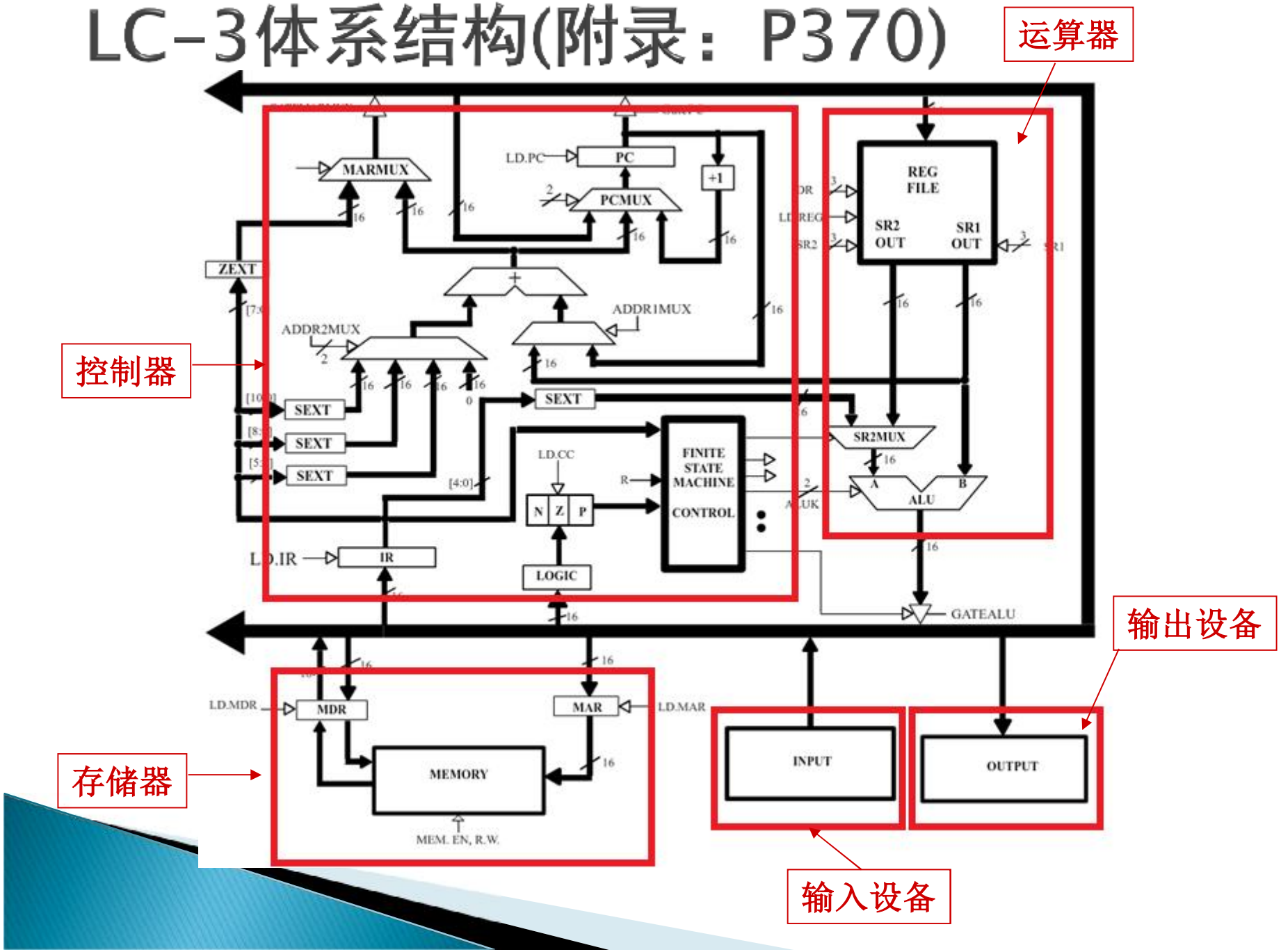
1 指令寄存器：Instruction Register (IR) 存放当前执行指令的内容。

2 程序寄存器：Program Counter (PC) 存放下一条要执行指令的地址.

- 自加： 当指令顺序执行时，由PC+1产生下一条指令的地址；
- 可改写:当遇到转移指令时，**转移地址** à PC作为下一条指令的地址。

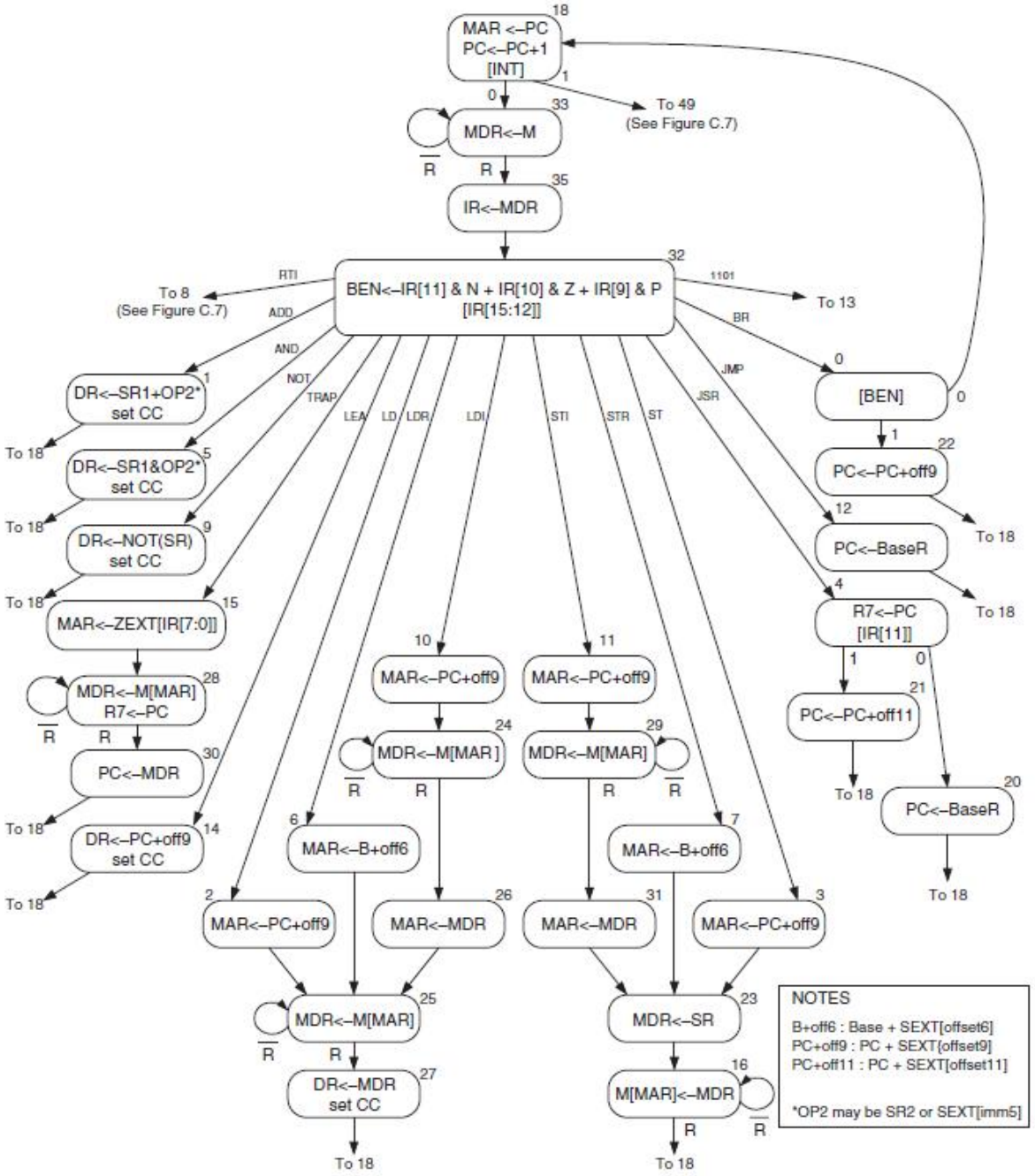
控制器的功能就是控制指令的执行过程

LC-3体系结构(附录：P370)



LC-3的5大部件

- 内存：包括存储单元，以及16位的MAR和16位的MDR寄存器。2^{16}x16bit
- 处理单元: 包括ALU和8个16位的寄存器(R0-R7)。
- 控制单元：PC,IR寄存器和控制逻辑有限状态机FSM
- 输入和输出单元：键盘和显示器。
- 部件的连接: 总线 ,同一时间只允许一个主设备使用


18
MAR <–PC
PC<–PC+1
[INT]
1
To 49
(See Figure C.7)
0
33
MDR<–M
R
R
35
IR<–MDR
32
BEN<–IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
RTI
To 8
(See Figure C.7)
1101
To 13
ADD
AND
NOT
TRAP
LEA
LD
LDR
LDI
STI
STR
ST
JSR
JMP
BR
0
[BEN]
0
1
22
PC<–PC+off9
To 18
12
PC<–BaseR
To 18
4
R7<–PC
[IR[11]]
1
0
21
PC<–PC+off11
To 18
20
PC<–BaseR
To 18
1
DR<–SR1+OP2*
set CC
To 18
5
DR<–SR1&OP2*
set CC
To 18
9
DR<–NOT(SR)
set CC
To 18
15
MAR<–ZEXT[IR[7:0]]
28
MDR<–M[MAR]
R7<–PC
R
R
30
PC<–MDR
To 18
14
DR<–PC+off9
set CC
To 18
10
MAR<–PC+off9
24
MDR<–M[MAR]
R
R
11
MAR<–PC+off9
29
MDR<–M[MAR]
R
R
6
MAR<–B+off6
7
MAR<–B+off6
2
MAR<–PC+off9
26
MAR<–MDR
31
MAR<–MDR
3
MAR<–PC+off9
25
MDR<–M[MAR]
R
R
27
DR<–MDR
set CC
To 18
23
MDR<–SR
16
M[MAR]<–MDR
R
R
To 18
NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
PC+off11 : PC + SEXT[offset11]
*OP2 may be SR2 or SEXT[imm5]

指令处理

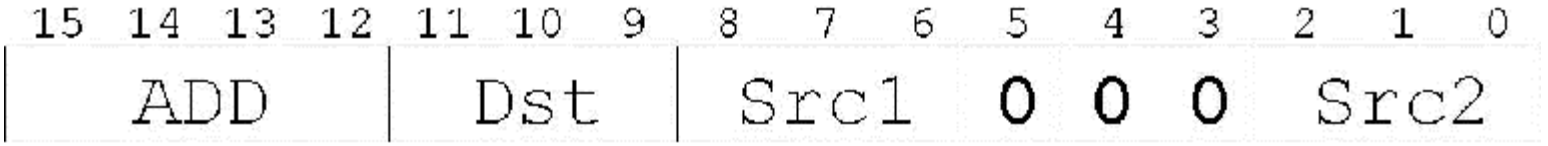
- 冯.诺伊曼模型的核心思想：程序和数据都是以bit流的方式存放在计算机内存中，程序在控制单元的控制下，依次完成指令的读取和执行。

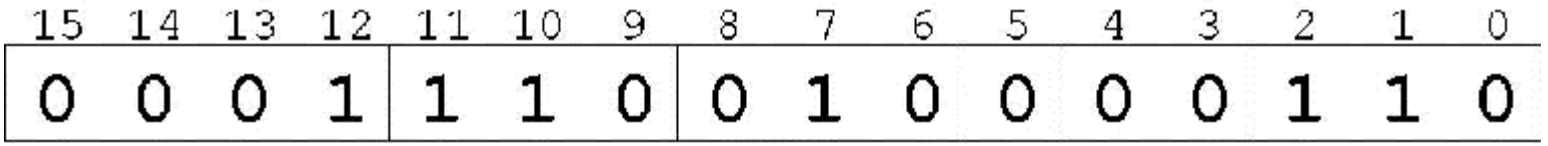
指令：

- 指令是由硬件电路直接完成的计算机执行的最小单位。
- 指令由两部分组成:
  - *操作码 opcode*:　指示指令具体做什么操作
  - *操作数 operands*: 指示操作的对象，一般存放操作数的地址（寄存器和内存地址），也可以在指令中直接存放较小的操作数。
  - 指令实质是一串二进制代码
  - 指令的助记形式: ADD R1,R2,R3　　ADD R1,R2,1
- 和数据信息一样，指令信息也必需用二进制编码。
  - 简单的计算机系统里面指令一般具有固定的二进制编码长度比如说 16 或者 32 位（指令字长）.
  - 控制单元负责具体解释每条指令并产生控制信号协调其它部件来完成指令执行。
  - 指令的执行具有原子性
- 一个计算机系统的所有指令和格式称为指令集。*Instruction Set Architecture (ISA)*.

指令编码 例: LC-3的ADD指令

- LC-3指令字长 ： 16bit.
  - 高四位bits [15:12]编码指令的操作码，最多16条指令。
- LC-3 运算单元具有8个16位的寄存器 (R0-R7) 用来暂存操作数和计算中间结果.可用三位二进制数编码每个寄存器。
  - ADD指令编码：目的操作数和源操作数都在寄存器中。

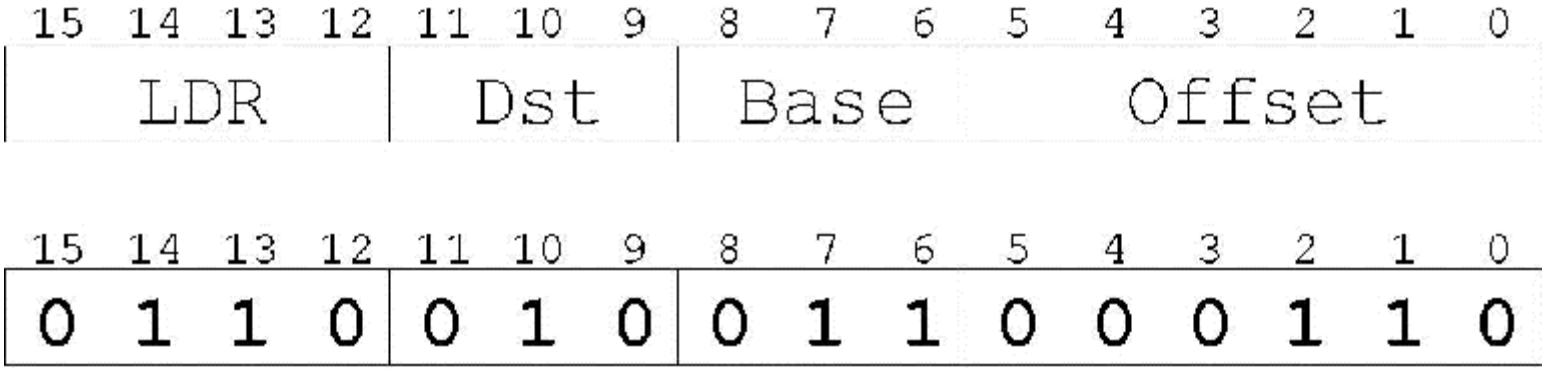




"Add the contents of R2 to the contents of R6, and store the result in R6."

指令编码 例: LC-3的LDR指令

- Load 指令– 从内存读数据到寄存器中
- 指令中访问16位地址的方法：使用基址+偏移的模式:
  - 16位基地址预先存放在指定的基址寄存器中
  - 通过基址+偏移—---得到访问的内存地址
  - load ： 从计算得到的内存地址读取内容到目标地址



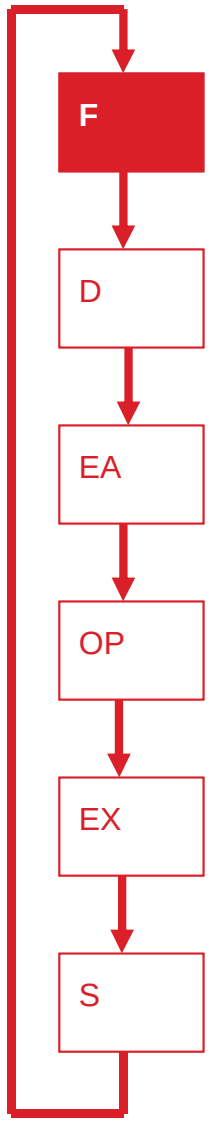
"Add the value 6 to the contents of R3 to form a memory address. Load the contents of that memory location to R2."

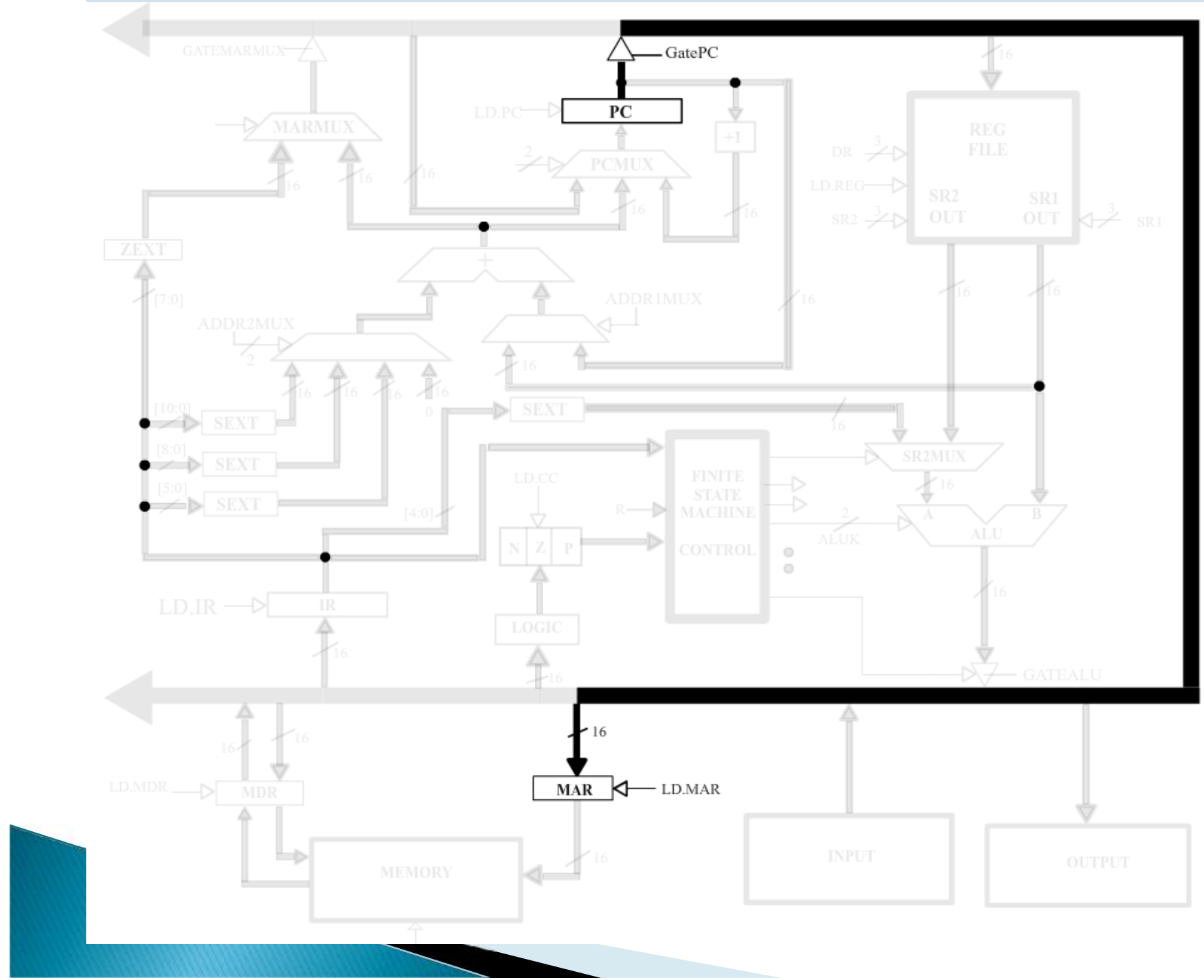
指令周期的六个步骤

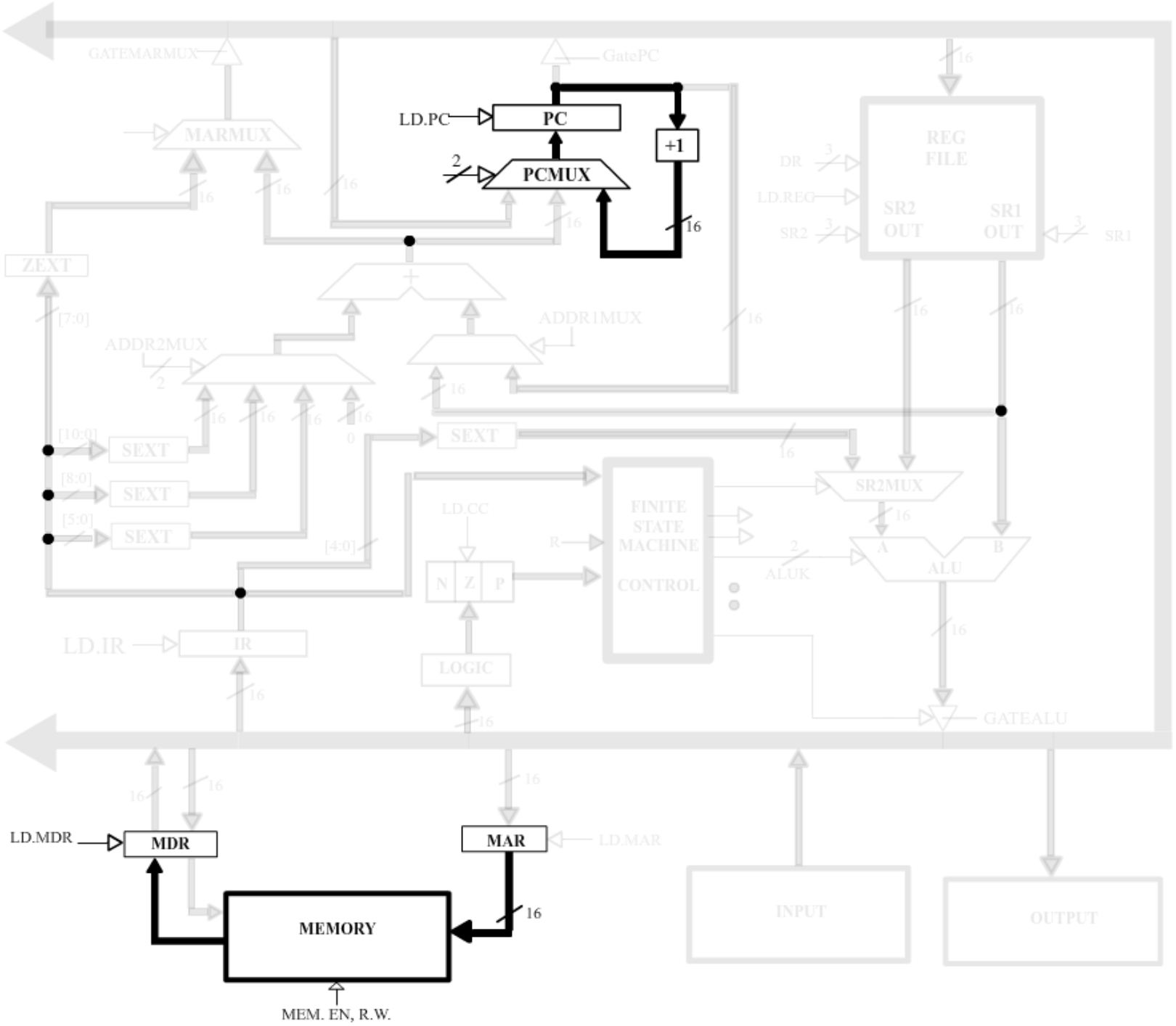

Fetch ：取指令
Decode：译码
Evaluate address：地址计算
Fetch operands ：取操作数
Execute：执行
Store result：存放结果

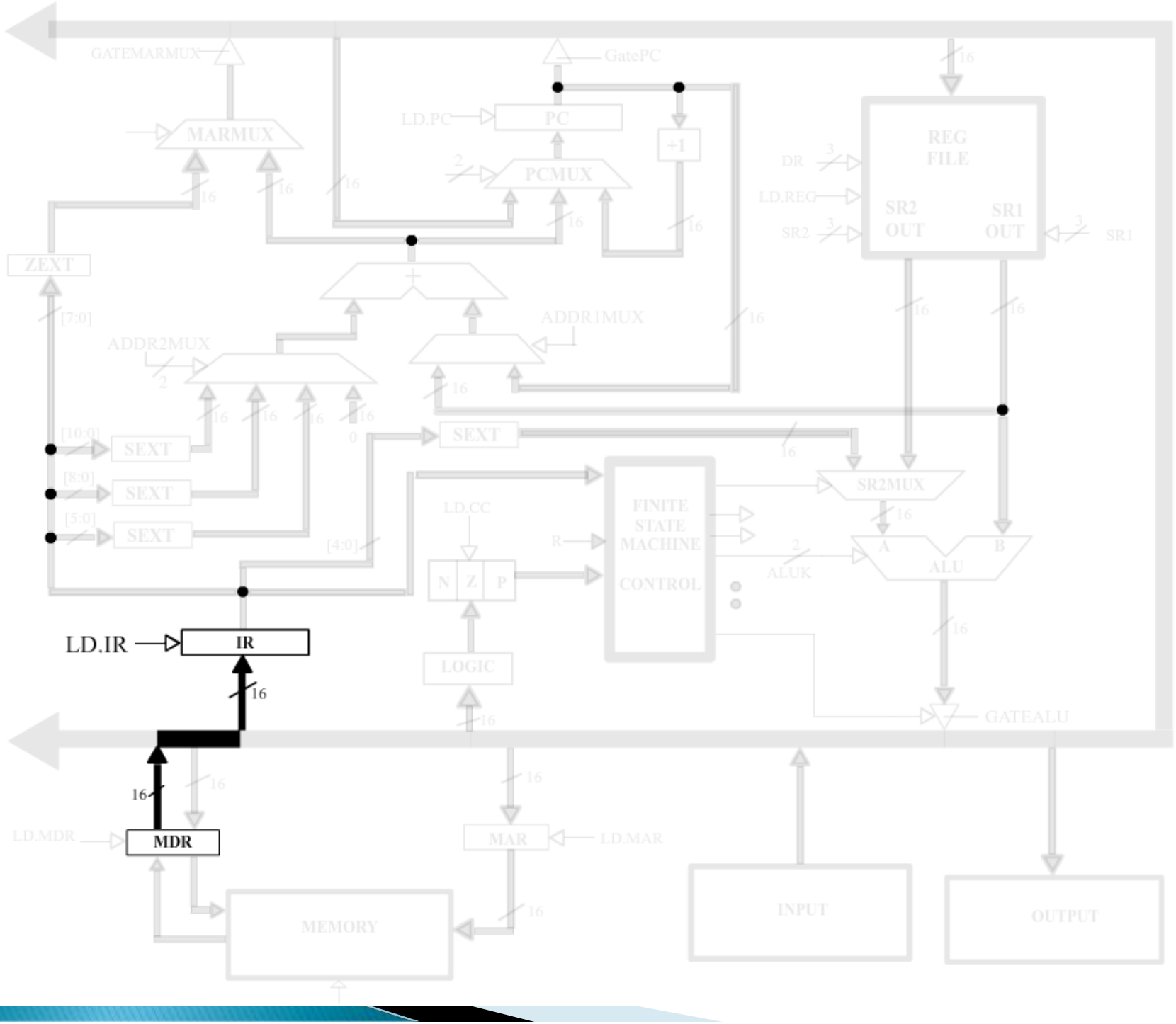
指令周期: FETCH

- 从内存中取下一条指令(地址存放在PC寄存器) 到指令寄存器 (IR)。
  - 把PC中存放的内容拷贝到 内存的MAR寄存器中.
  - 给内存发读信号。
  - 拷贝MDR的内容到IR寄存器中。
  - PC=PC+1,PC指向下一条待执行的指令




GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
REG
FILE
DR
LD.REG
SR2
SR2
OUT
SR1
OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
SEXT
SR2MUX
LD.CC
FINITE
STATE
MACHINE
CONTROL
N
Z
P
A
B
ALU
ALUK
[4:0]
LD.IR
IR
LOGIC
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT

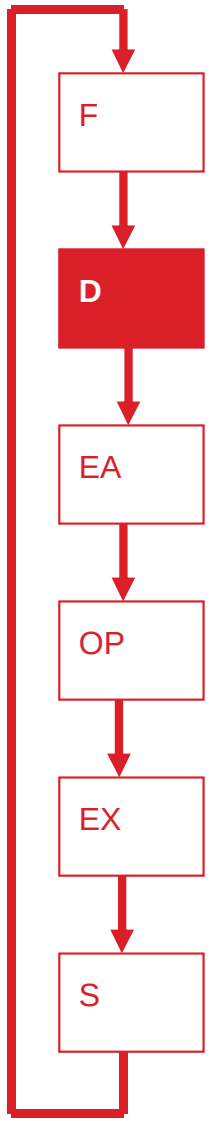

GATEMARMUX
GatePC
LD.PC
PC
+1
2
PCMUX
16
MARMUX
16
16
16
16
REG
FILE
DR
3
LD.REG
SR2
OUT
SR1
OUT
SR2
3
3
SR1
ZEXT
[7:0]
ADDR2MUX
2
ADDR1MUX
16
16
16
16
16
16
16
16
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
0
SEXT
16
SR2MUX
16
LD.CC
R
FINITE
STATE
MACHINE
CONTROL
[4:0]
N
Z
P
2
ALUK
A
B
ALU
16
LD.IR
IR
LOGIC
16
16
GATEALU
16
16
LD.MDR
MDR
MAR
LD.MAR
MEMORY
16
MEM. EN, R.W.
INPUT
OUTPUT


GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
REG FILE
DR
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
SR2MUX
LD.CC
FINITE STATE MACHINE
CONTROL
R
N
Z
P
A
B
ALU
ALUK
LOGIC
LD.IR
IR
16
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT

指令周期: DECODE

- 首先从指令中识别操作码（opcode）
  - LC-3：指令的高4位总是操作码。利用一个 4-to-16 译码器对操作码译码，有效对应的控制线.
- 根据操作码，确定对应的操作数。操作码不同，需要的操作数和定义不同。

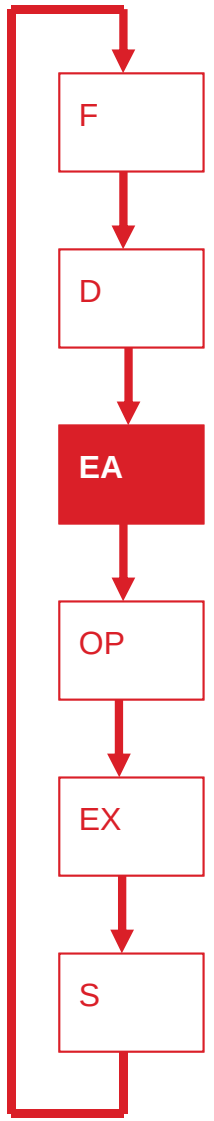
  例:
  - 指令LDR, 最后六位是地址偏移 （offset)
  - 指令ADD, 最后三位定义了参与运算的第二个源操作数



指令周期：EVALUATE ADDRESS

- 在指令中通常无法直接给出16位的有效地址，需要根据指令实际定义的寻址方式进行计算。

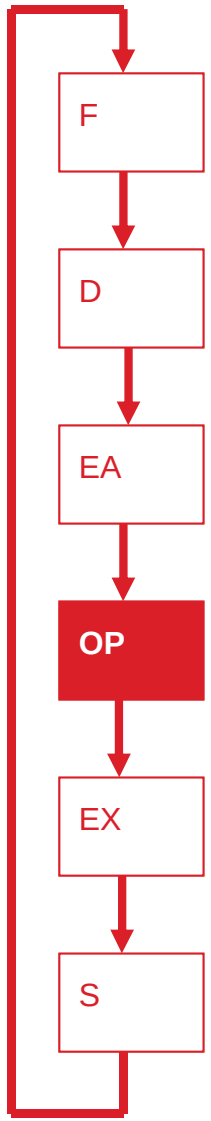
- Examples:
  - 间接寻址：基址存放在寄存器中，通过基址+偏移来计算数据的实际地址(as in LDR)
  - 相对寻址：在指令地址的基础上加偏移。通过PC+偏移来计算数据的实际地址
  - 立即地址：偏移+0



指令周期: FETCH OPERANDS

- 数据存放地点：寄存器，内存
- 根据寄存器地址或计算得到的内存地址读取操作数。

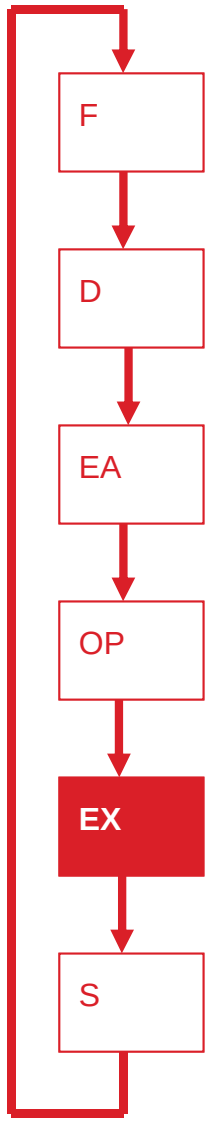
- Examples:
  - load data from memory (LDR)
  - read data from register file (ADD)



指令周期: 执行

- 基于得到的操作数完成操作码制定的操作。

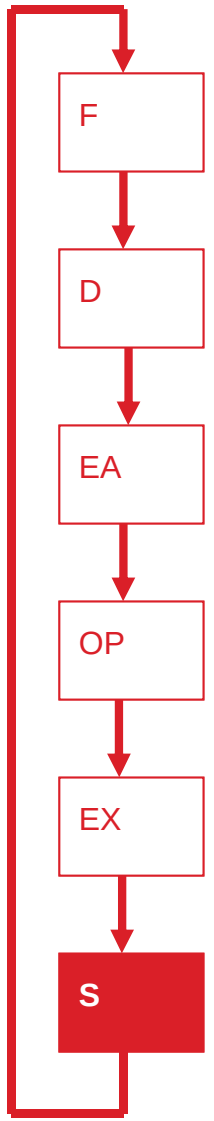
- Examples:
  - 把操作数送到 ALU，发加操作信号（ADD指令）
  - 什么也不做：数据转移类指令(e.g., for loads and stores)

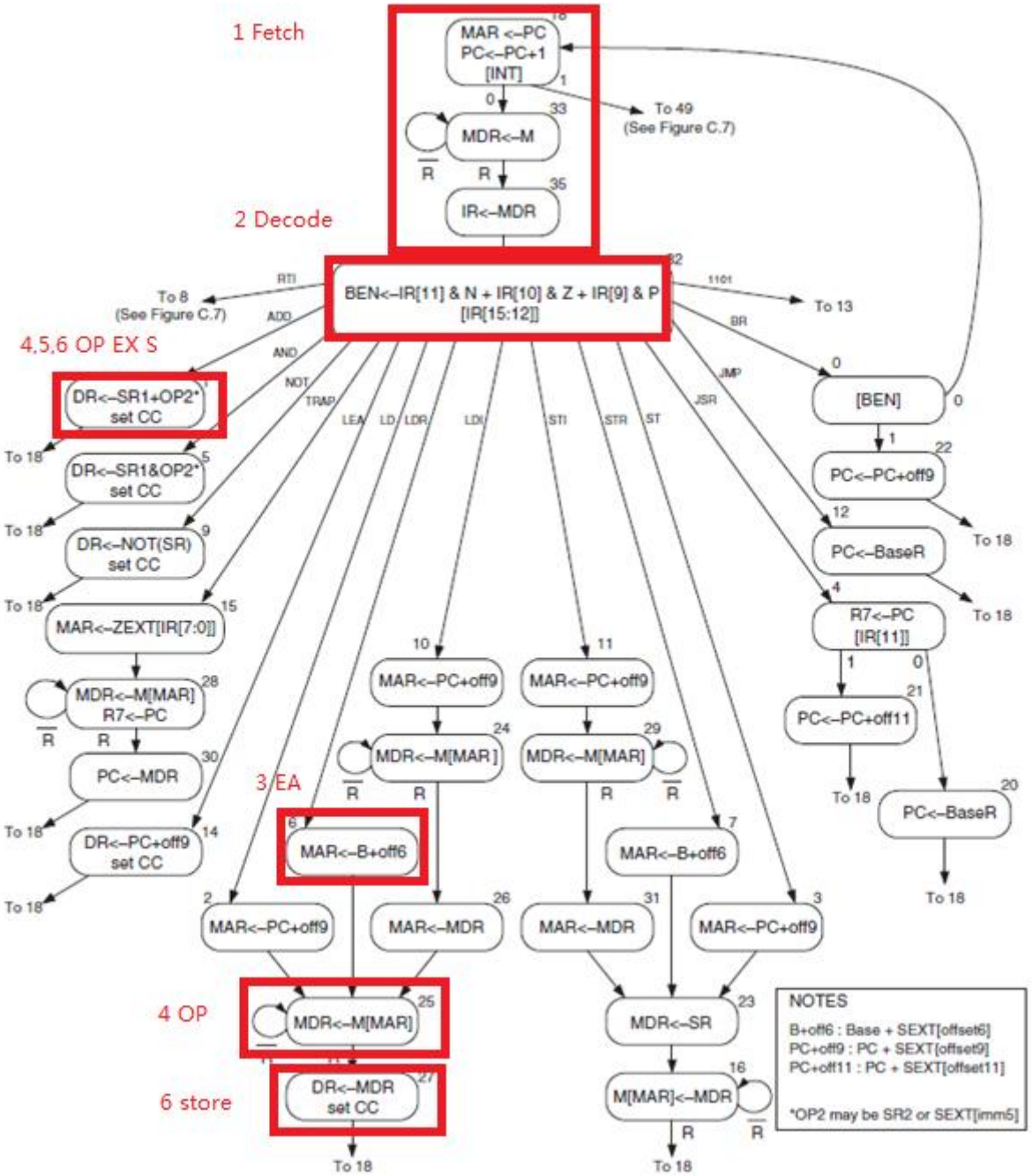


指令周期: STORE RESULT

- 把运算结果存放到目标地点
  (寄存器 or 存储器)

- Examples:
  - ADD指令的执行结果存放在目标寄存器中
  - LOAD类指令的执行结果存放在目标寄存器中
  - STORE类指令的执行结果是存放在存储器中
    - 写地址到 MAR寄存器中, 要写的数据送到 MDR
    - 发写信号给内存



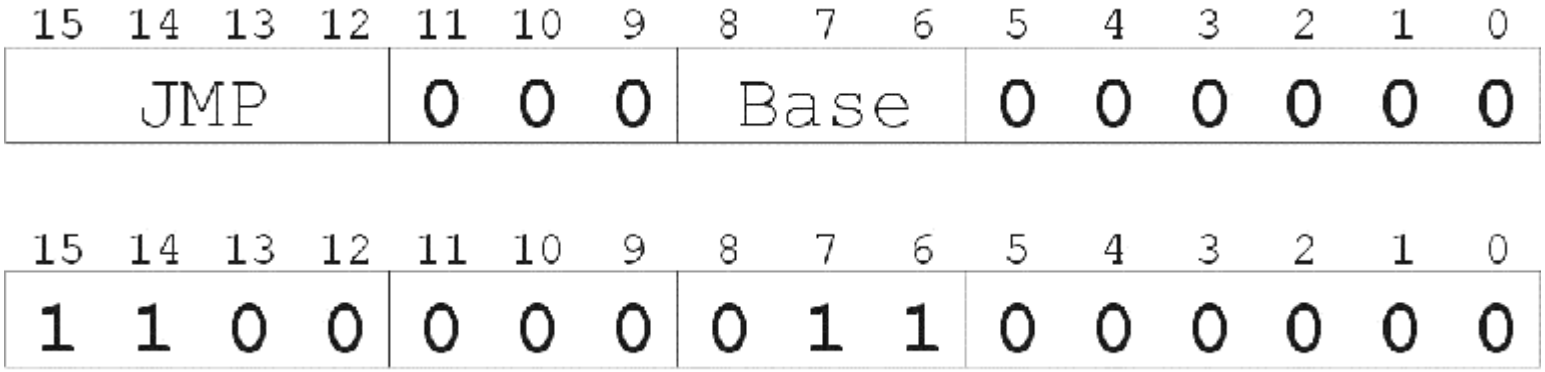

1 Fetch
MAR <–PC
PC<–PC+1
[INT]
To 49
(See Figure C.7)
MDR<–M
IR<–MDR
2 Decode
BEN<–IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
RTI
To 8
(See Figure C.7)
ADD
AND
NOT
TRAP
LEA
LD
LDR
LDI
STI
STR
ST
JSR
JMP
BR
1101
To 13
4,5,6 OP EX S
DR<–SR1+OP2*
set CC
DR<–SR1&OP2*
set CC
DR<–NOT(SR)
set CC
MAR<–ZEXT[IR[7:0]]
MDR<–M[MAR]
R7<–PC
PC<–MDR
DR<–PC+off9
set CC
[BEN]
PC<–PC+off9
PC<–BaseR
R7<–PC
[IR[11]]
PC<–PC+off11
PC<–BaseR
To 18
3 EA
MAR<–PC+off9
MDR<–M[MAR]
MAR<–B+off6
MAR<–MDR
4 OP
MDR<–M[MAR]
6 store
DR<–MDR
set CC
MDR<–SR
M[MAR]<–MDR
NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
PC+off11 : PC + SEXT[offset11]
*OP2 may be SR2 or SEXT[imm5]

改变指令执行顺序–控制指令

- 指令的顺序执行：在FETCH 阶段的最后, PC＝PC＋1
- 但实际程序中的分支指令，循环指令，子程序调用等会改变程序的执行顺序。
- 控制指令：可以改写PC寄存器的内容。
  - 无条件跳转 – 改变PC值到目标地址。
  - 条件跳转：分支指令等 – 改变PC值取决于跳转条件是否成立

例: LC-3 JMP 指令

- 把 PC 值改写为某个寄存器的内容。改变下一条要执行的指令地址为寄存器里面存放的地址值



"Load the contents of R3 into the PC."

小结

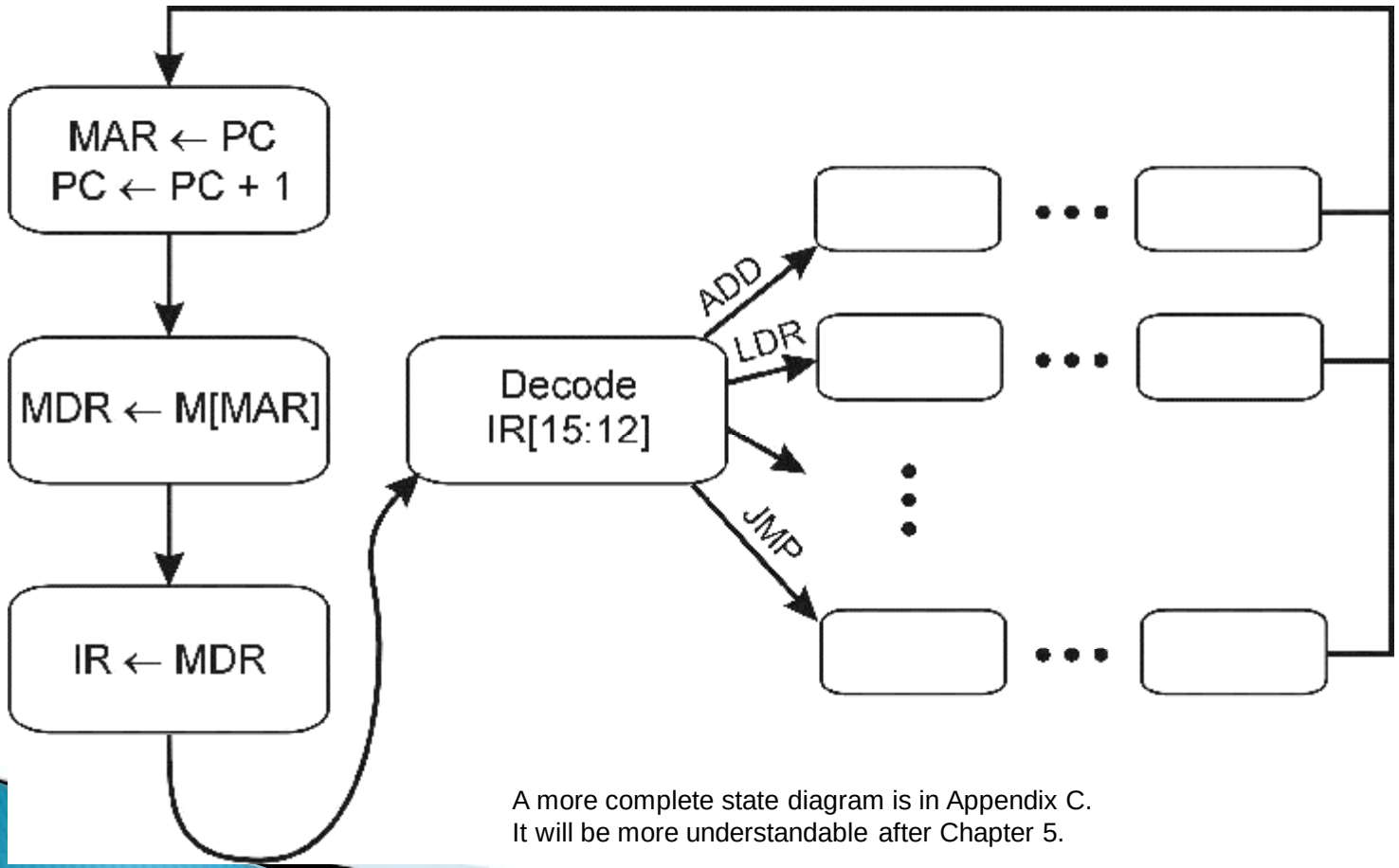
- 指令和数据一样，要按固定的格式进行二进制编码
- 三种类型的指令
  - 计算型指令(ADD, AND, …)，操作数一般都在寄存器中。
  - 数据移动指令(LD, ST, …)，负责从寄存器和内存之间的数据转移。
  - 控制指令(JMP, BRnz, …)，根据条件改变指令执行顺序。
- 指令执行的六个基本步骤

  $F \rightarrow D \rightarrow EA \rightarrow OP \rightarrow EX \rightarrow S$

  - 并不是所有的指令执行都需要六个步骤完成。每一个步骤的执行时间也可能不同。

控制单元的控制状态机

- 控制单元是一个时序电路状态机. 以下是LC-3状态机的示意图。

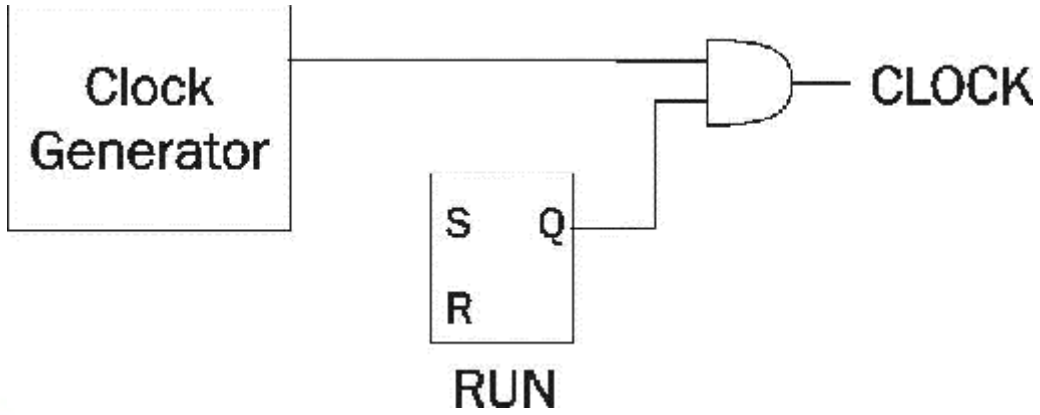


A more complete state diagram is in Appendix C.
It will be more understandable after Chapter 5.

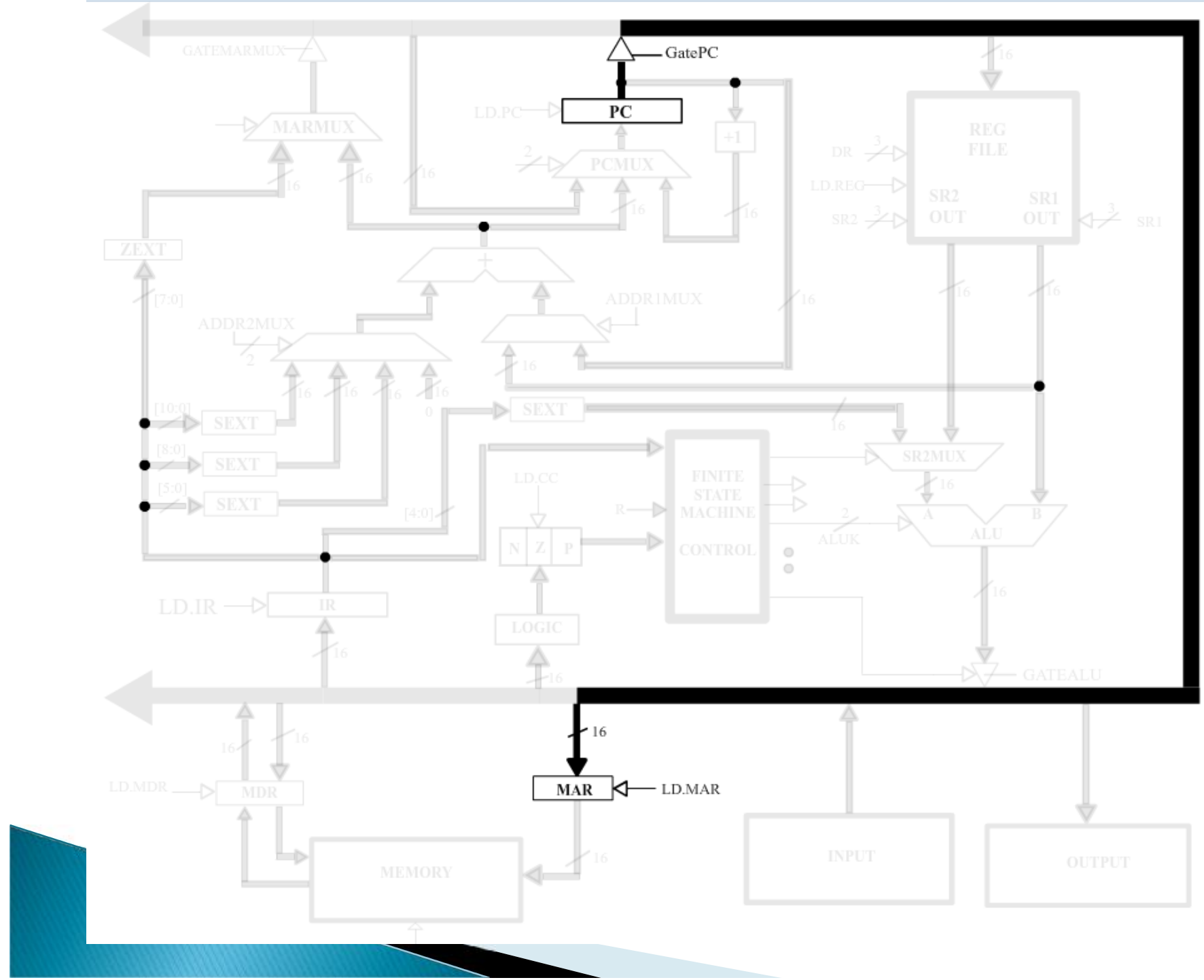
停机操作

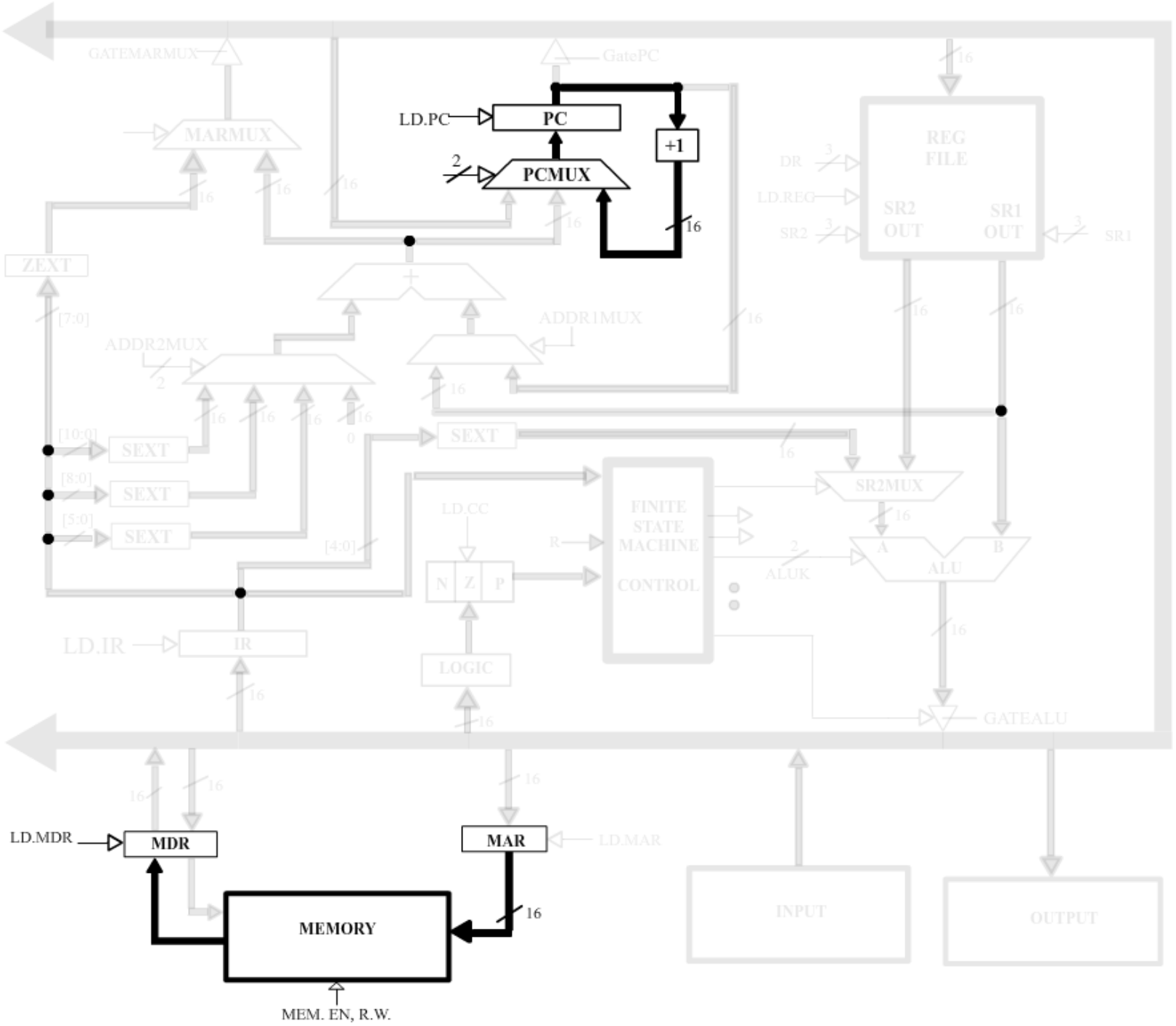
- 系统时钟如果不停止，控制单元会不断重复地进行指令处理。
  - 不是执行程序的指令，就是执行操作系统的指令。

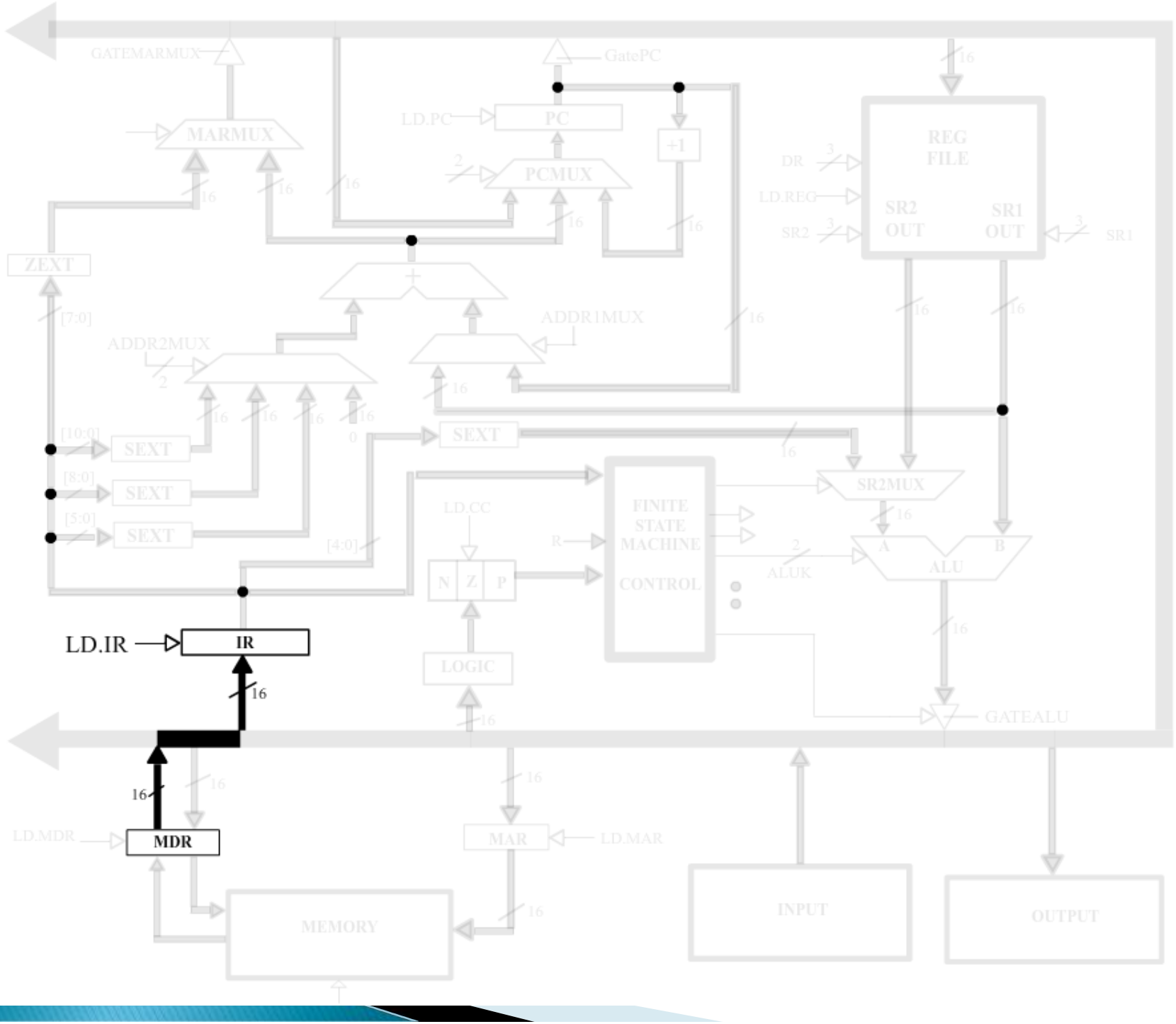
- 系统停机的方法：停止时钟（S=1,R=0）。控制单元状态机是在时钟变化控制下进行状态切换，时钟停止，状态机停止工作，终止的指令的执行。

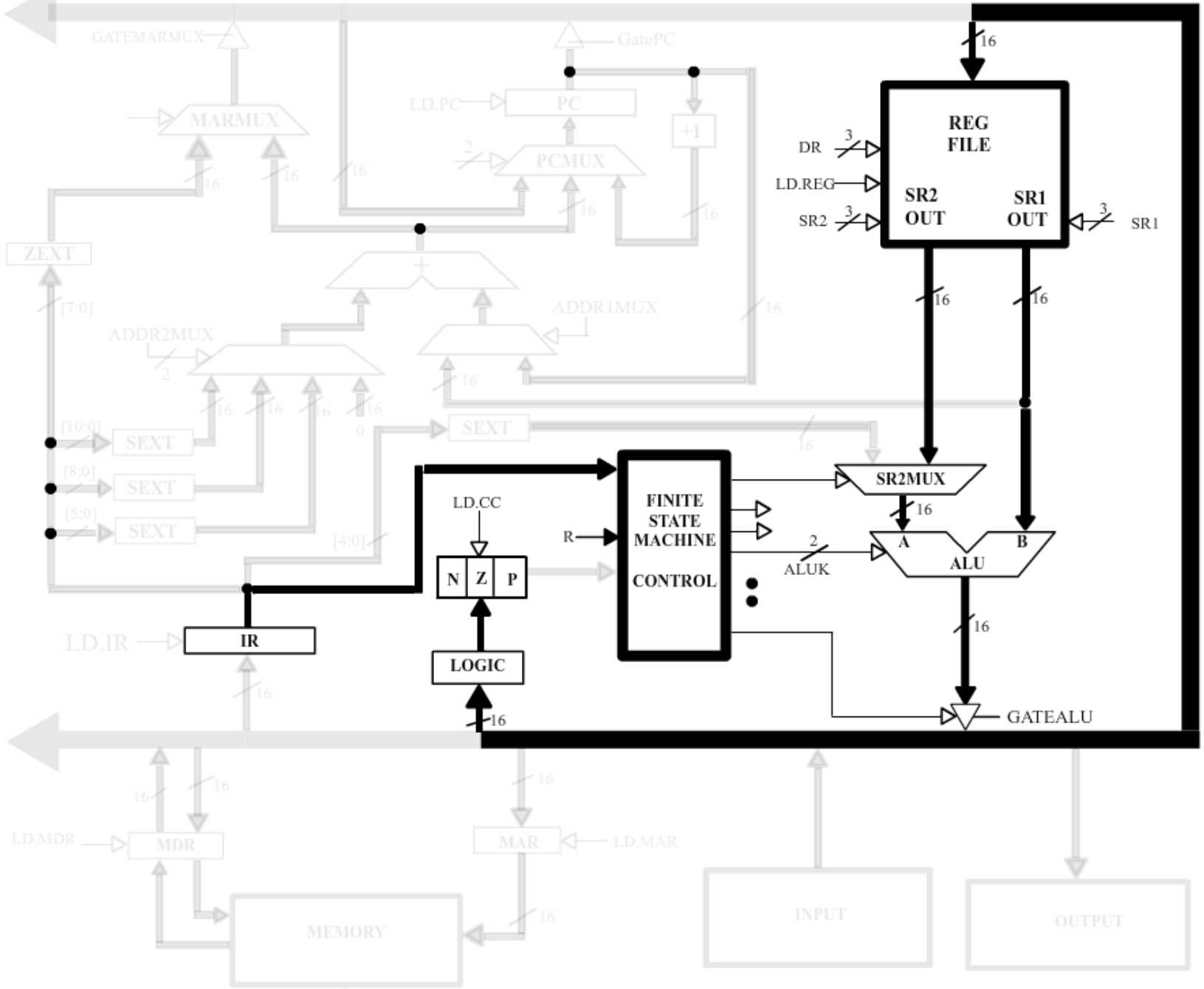


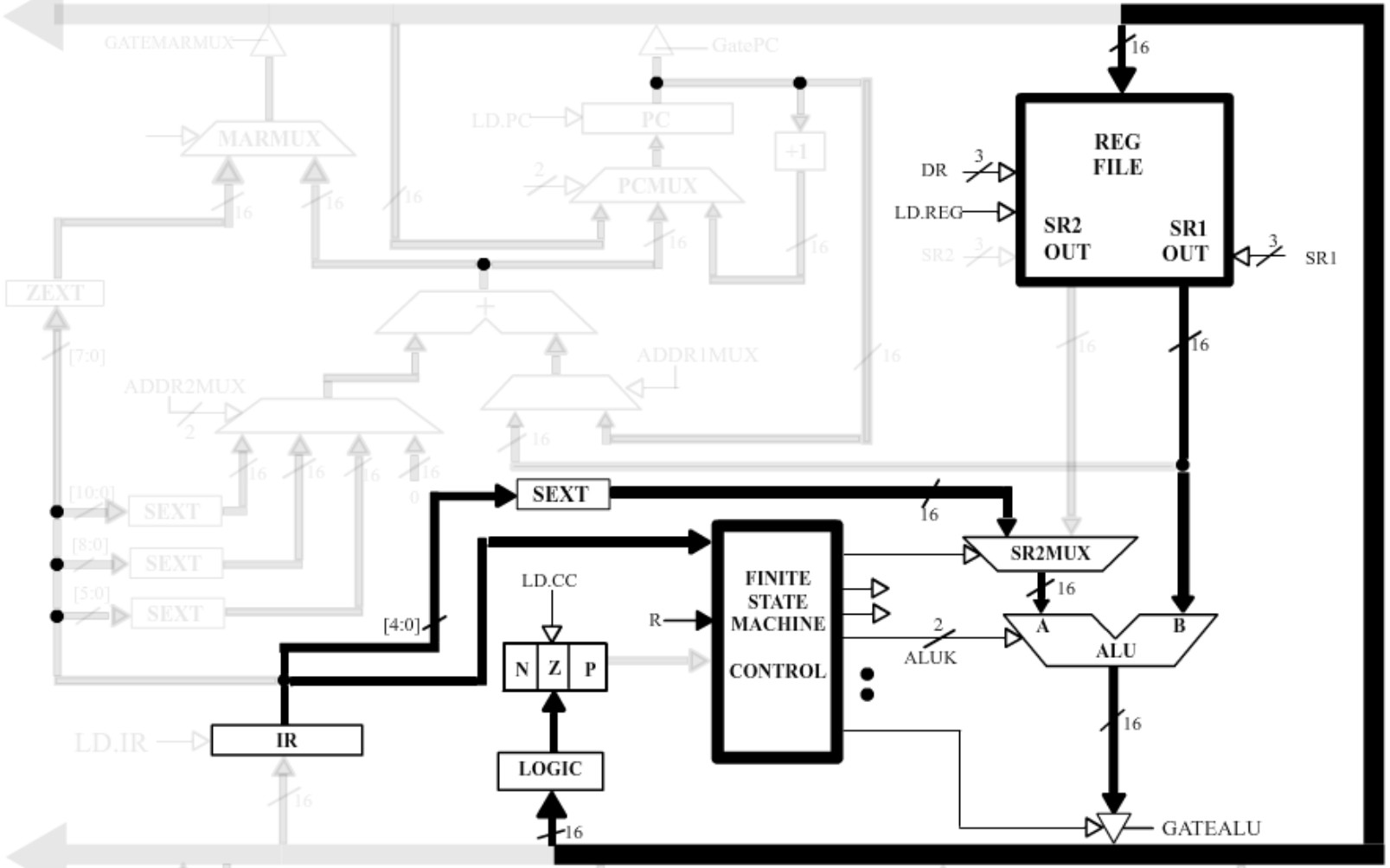
ADD指令执行过程


GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
REG
FILE
DR
LD.REG
SR2
SR2
OUT
SR1
OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
SEXT
SR2MUX
LD.CC
FINITE
STATE
MACHINE
CONTROL
N
Z
P
A
B
ALU
ALUK
[4:0]
LD.IR
IR
LOGIC
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT

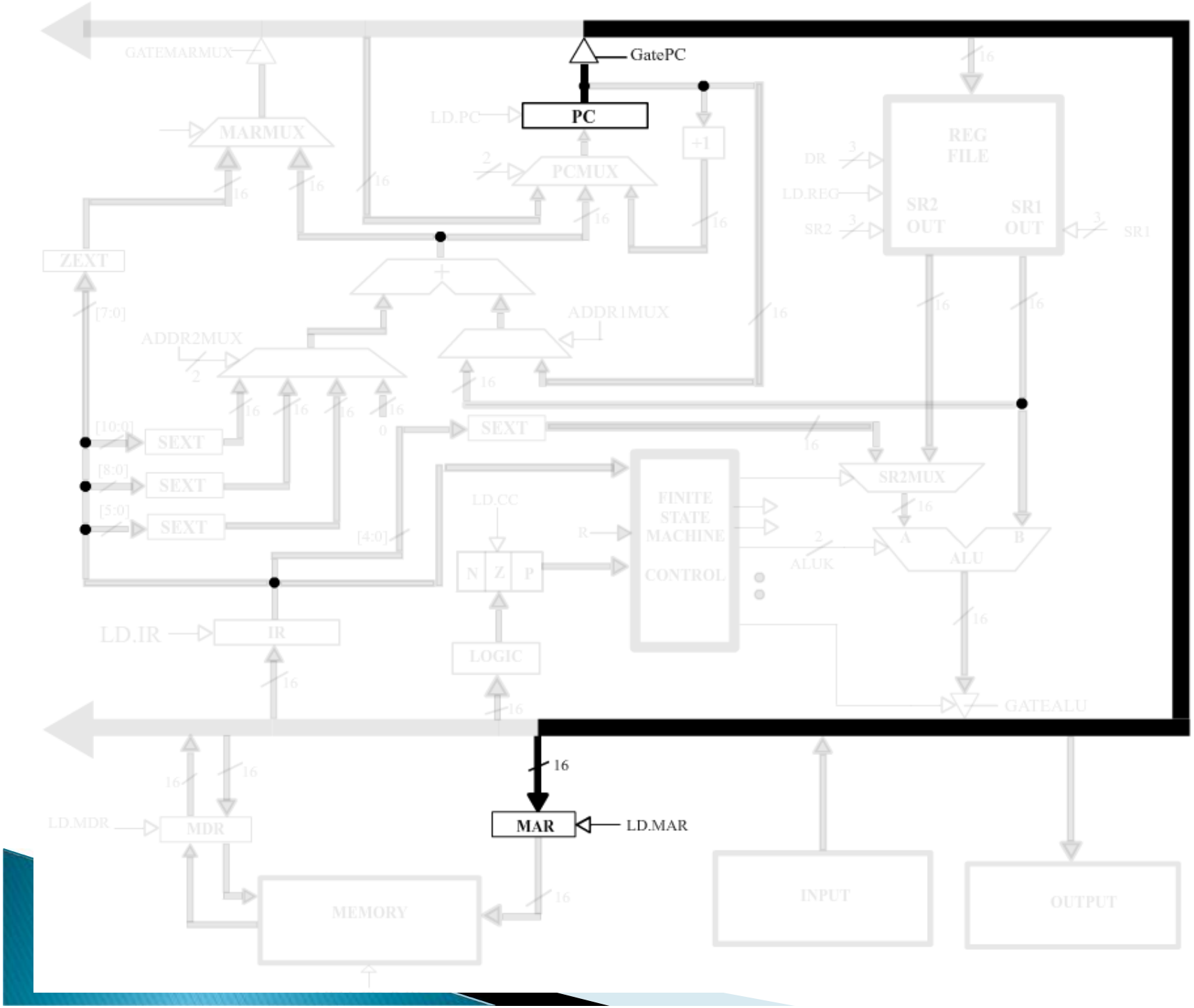

GATEMARMUX
GatePC
LD.PC
PC
+1
2
PCMUX
16
MARMUX
16
16
16
16
REG
FILE
DR
3
LD.REG
SR2
OUT
SR1
OUT
SR2
3
3
SR1
ZEXT
[7:0]
ADDR2MUX
2
ADDR1MUX
16
16
16
16
16
16
16
16
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
0
SEXT
16
SR2MUX
16
LD.CC
R
FINITE
STATE
MACHINE
CONTROL
[4:0]
N
Z
P
2
ALUK
A
B
ALU
16
LD.IR
IR
LOGIC
16
16
GATEALU
16
16
LD.MDR
MDR
MAR
LD.MAR
MEMORY
16
MEM. EN, R.W.
INPUT
OUTPUT

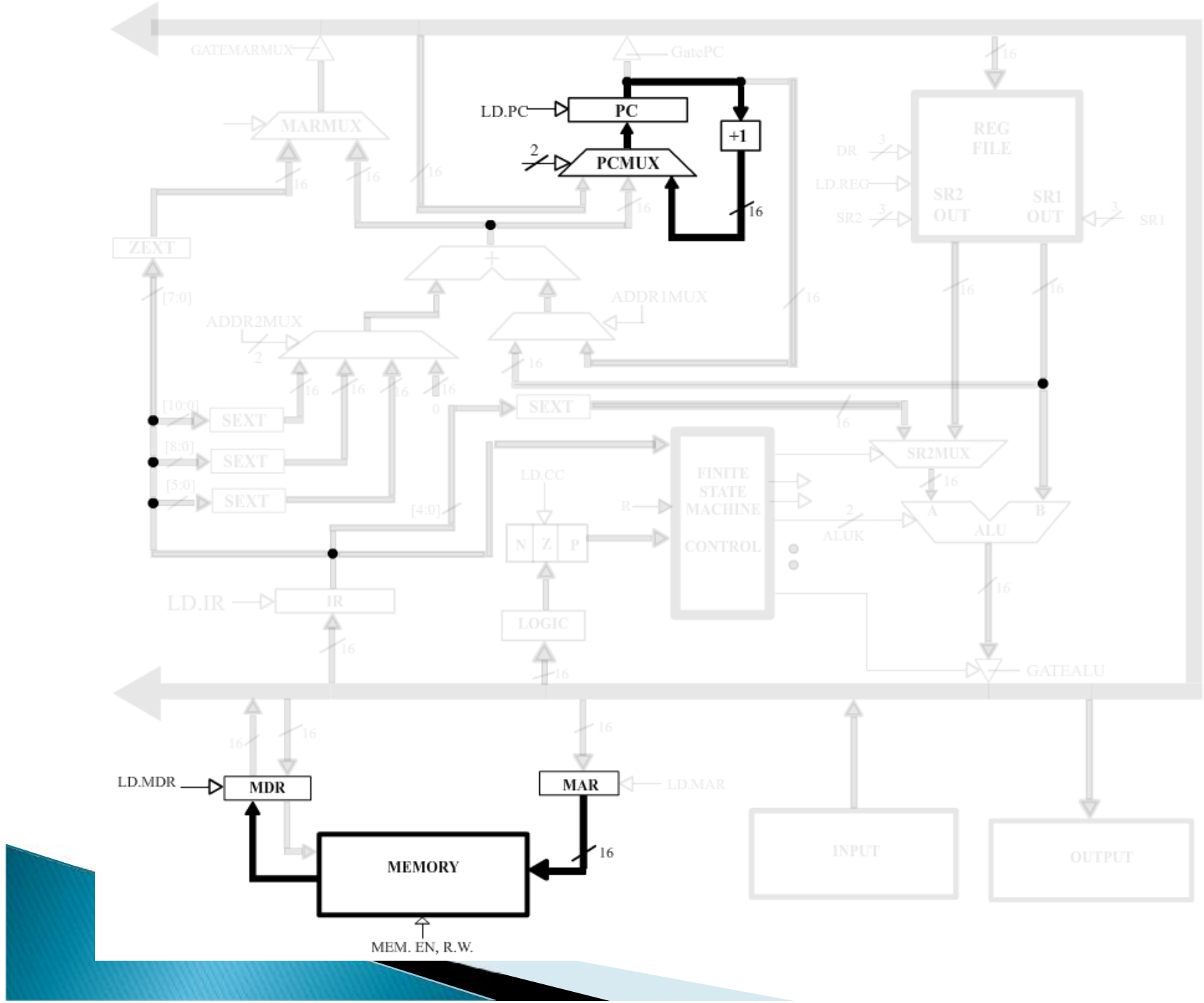

GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
REG FILE
DR
LD.REG
SR2
SR2 OUT
SR1 OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
SR2MUX
LD.CC
FINITE STATE MACHINE
CONTROL
R
N
Z
P
A
B
ALU
ALUK
LOGIC
LD.IR
IR
16
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT

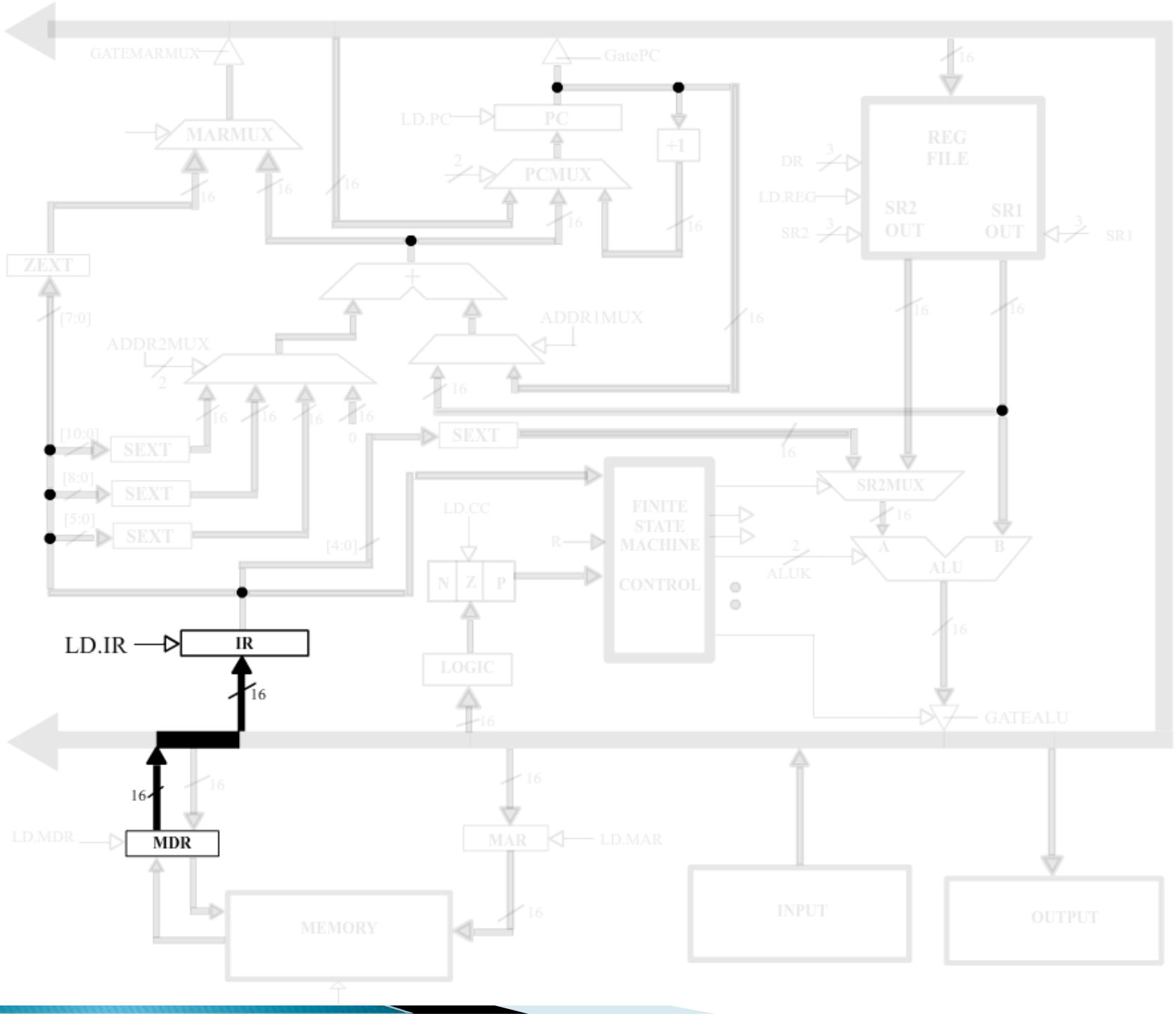

GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
2
16
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
+
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
0
REG
FILE
DR
3
LD.REG
SR2
SR2
OUT
SR1
OUT
SR1
SR2MUX
A
B
ALU
ALUK
LD.CC
N
Z
P
R
FINITE
STATE
MACHINE
CONTROL
LD.IR
IR
LOGIC
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT

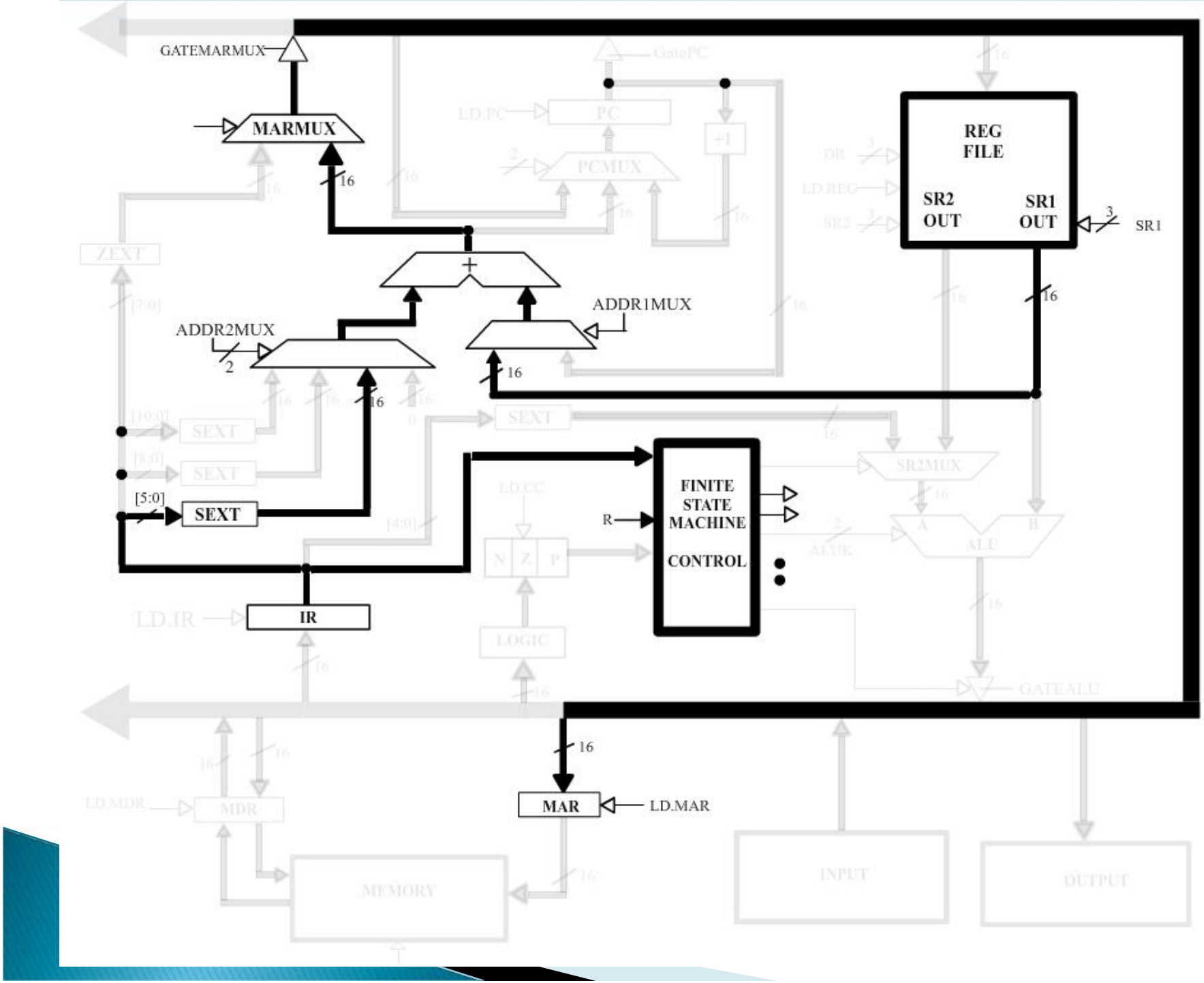

REG
FILE
DR
LD.REG
SR2
OUT
SR1
OUT
SR1
16
3
SEXT
SR2MUX
FINITE
STATE
MACHINE
CONTROL
LD.CC
N
Z
P
R
[4:0]
2
ALUK
A
B
ALU
IR
LD.IR
LOGIC
GATEALU
GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
ZEXT
ADDR2MUX
ADDR1MUX
[10:0]
[8:0]
[5:0]
[7:0]
LD.MDR
MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT

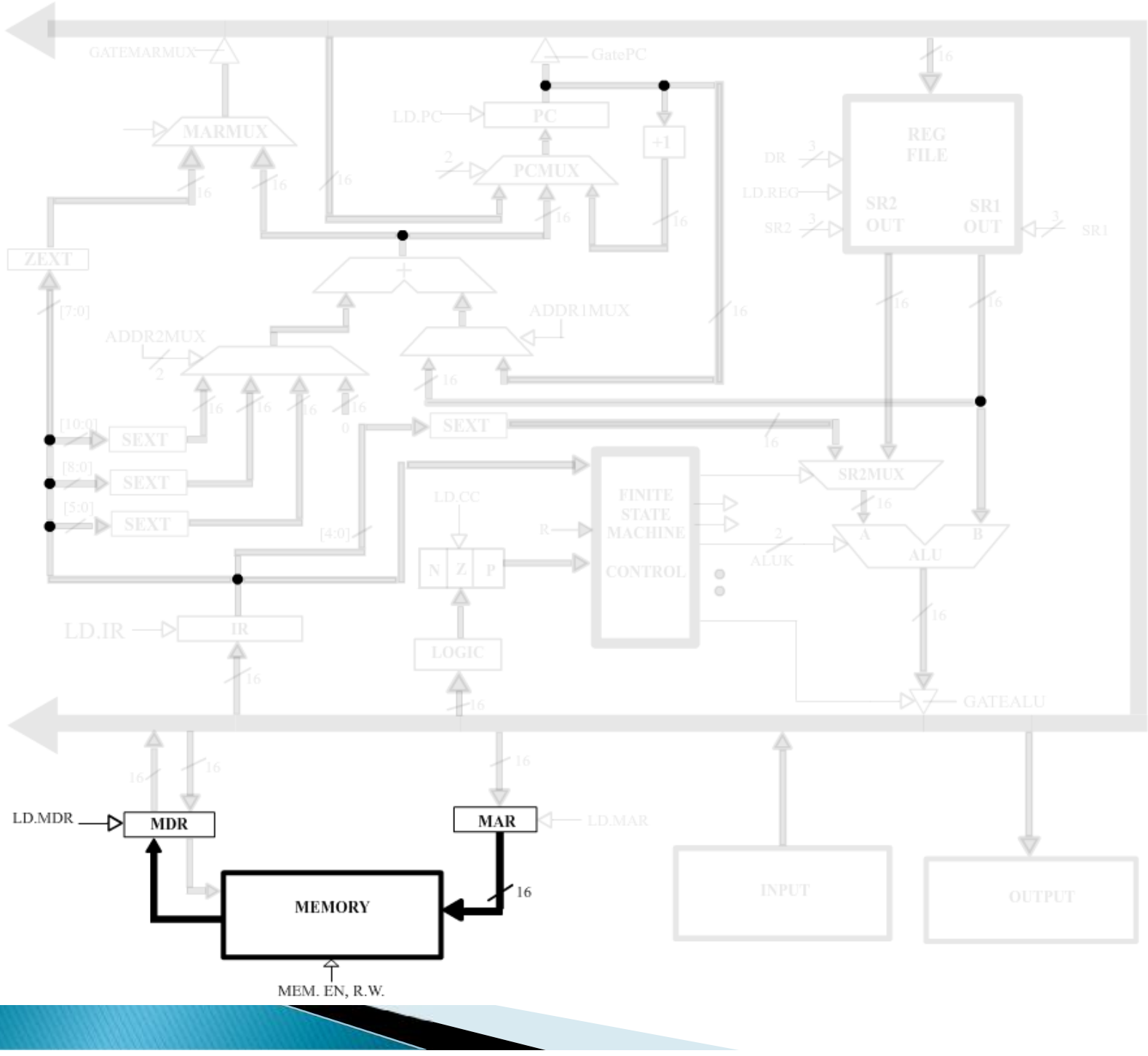
LDR指令执行过程

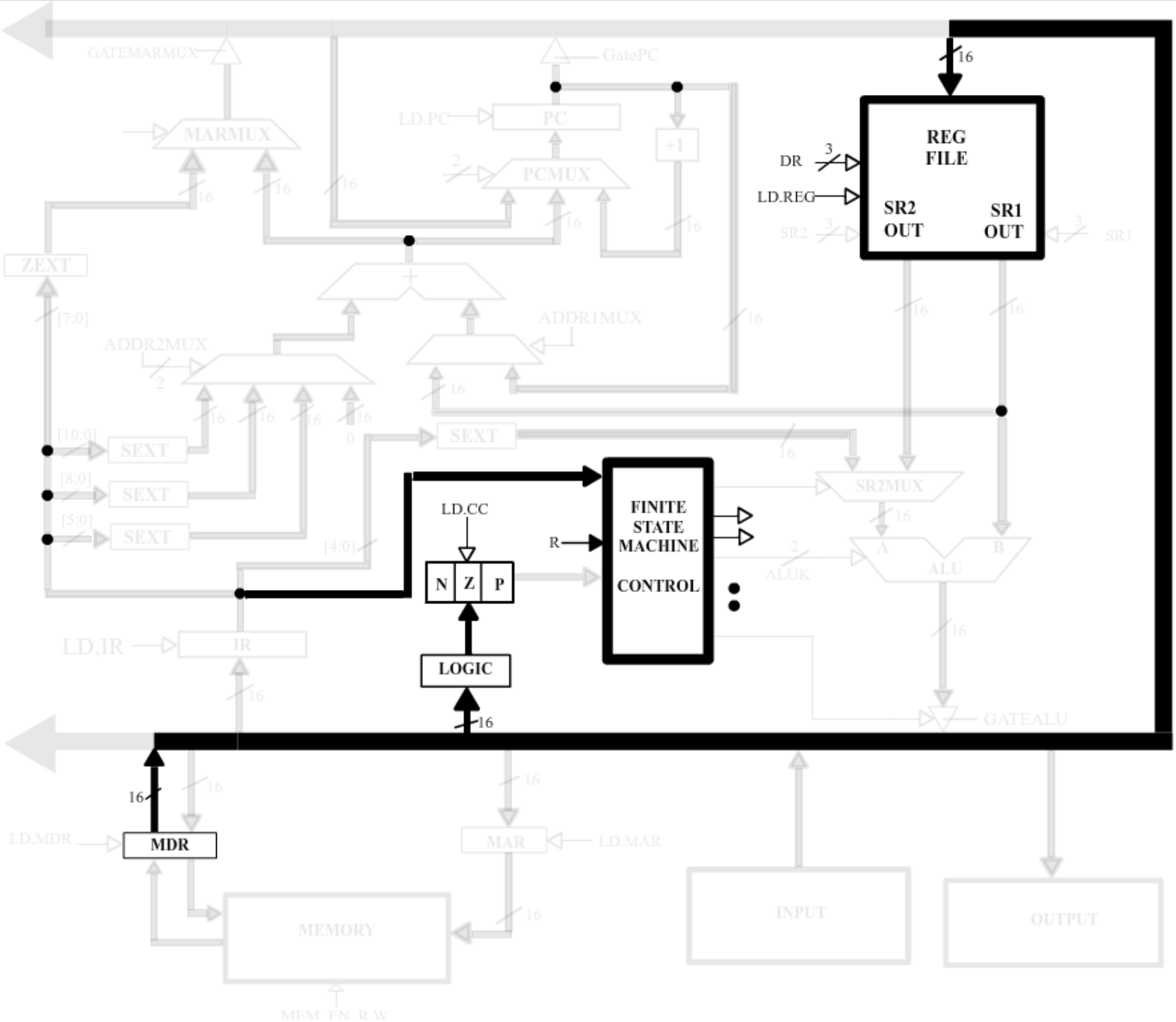

GATEMARMUX
GatePC
LD.PC
PC
MARMUX
PCMUX
+1
REG
FILE
DR
LD.REG
SR2
SR2
OUT
SR1
OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
SEXT
SEXT
SEXT
SR2MUX
LD.CC
FINITE
STATE
MACHINE
CONTROL
N
Z
P
A
B
ALU
ALUK
LD.IR
IR
LOGIC
GATEALU
MAR
LD.MAR
LD.MDR
MDR
MEMORY
INPUT
OUTPUT


GATEMARMUX
GatePC
LD.PC
PC
+1
2
PCMUX
16
MARMUX
REG
FILE
DR
LD.REG
SR2
SR2
OUT
SR1
OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
LD.CC
N
Z
P
FINITE
STATE
MACHINE
CONTROL
SR2MUX
A
B
ALU
ALUK
LD.IR
IR
LOGIC
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
MEM. EN, R.W.
INPUT
OUTPUT


GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
2
16
REG
FILE
DR
3
LD.REG
SR2
SR2
OUT
SR1
OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
[4:0]
0
SR2MUX
LD.CC
FINITE
STATE
MACHINE
CONTROL
R
N
Z
P
A
B
ALU
ALUK
LD.IR
IR
LOGIC
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT


GATEMARMUX
MARMUX
16
REG
FILE
SR2
OUT
SR1
OUT
3
SR1
ADDR1MUX
ADDR2MUX
2
16
16
[5:0]
SEXT
FINITE
STATE
MACHINE
CONTROL
R
N
Z
P
IR
LD.IR
MAR
LD.MAR
16


GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
REG
FILE
DR
LD.REG
SR2
SR2
OUT
SR1
OUT
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
SEXT
[10:0]
[8:0]
[5:0]
SEXT
SEXT
SEXT
LD.CC
[4:0]
N
Z
P
FINITE
STATE
MACHINE
CONTROL
SR2MUX
A
B
ALU
ALUK
LD.IR
IR
LOGIC
GATEALU
LD.MDR
MDR
MAR
LD.MAR
MEMORY
MEM. EN, R.W.
INPUT
OUTPUT


GATEMARMUX
GatePC
LD.PC
PC
+1
MARMUX
PCMUX
2
16
REG
FILE
DR
3
LD.REG
SR2
OUT
SR1
OUT
SR2
SR1
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
+
[10:0]
[8:0]
[5:0]
SEXT
SEXT
SEXT
0
[4:0]
LD.CC
R
FINITE
STATE
MACHINE
CONTROL
N
Z
P
SR2MUX
A
B
ALU
ALUK
LD.IR
IR
LOGIC
GATEALU
MDR
LD.MDR
MAR
LD.MAR
MEMORY
INPUT
OUTPUT
MEM_EN, R.W

作业

- Ex 4.5 (B3不做)
- Ex 4.7