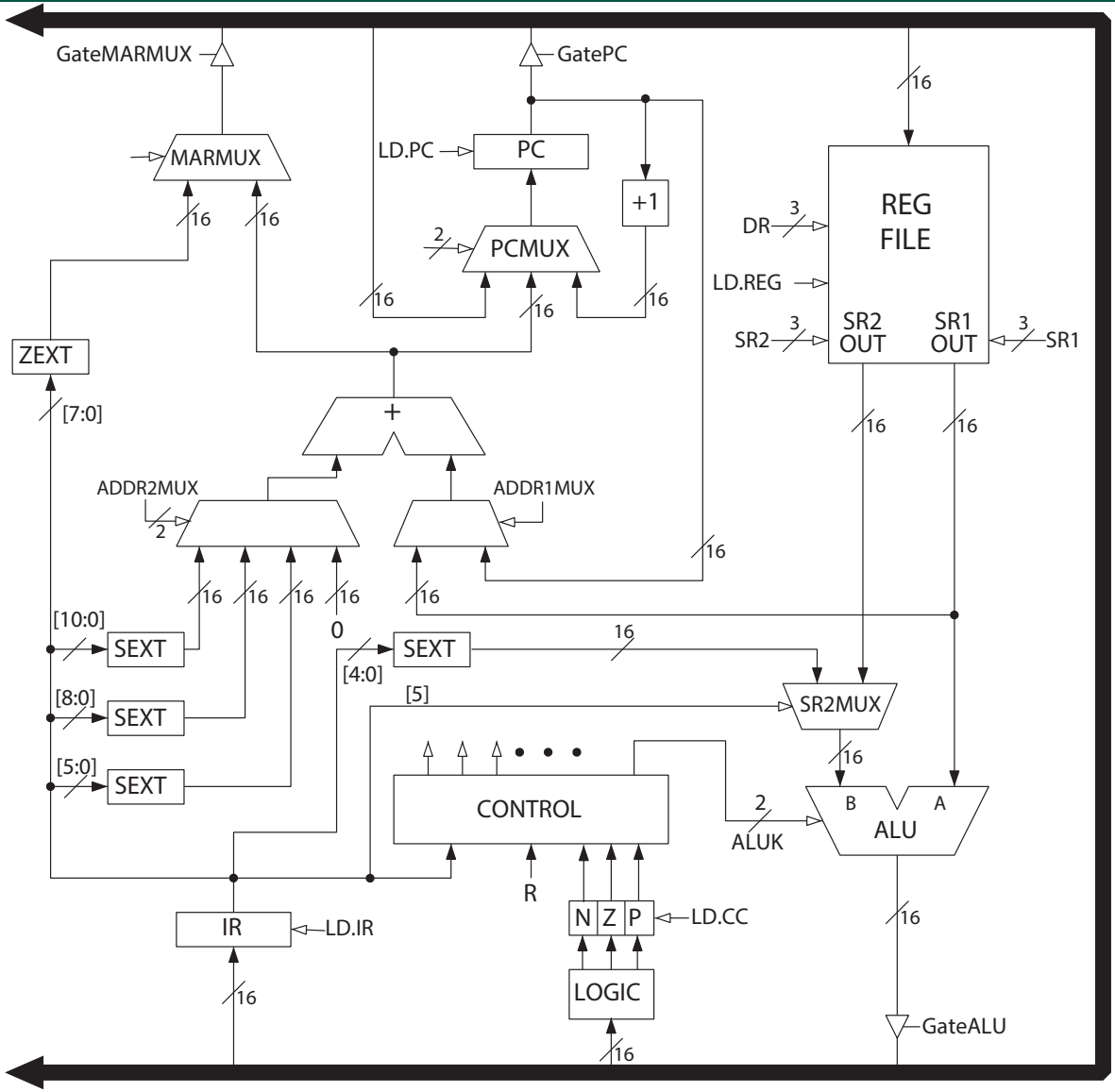
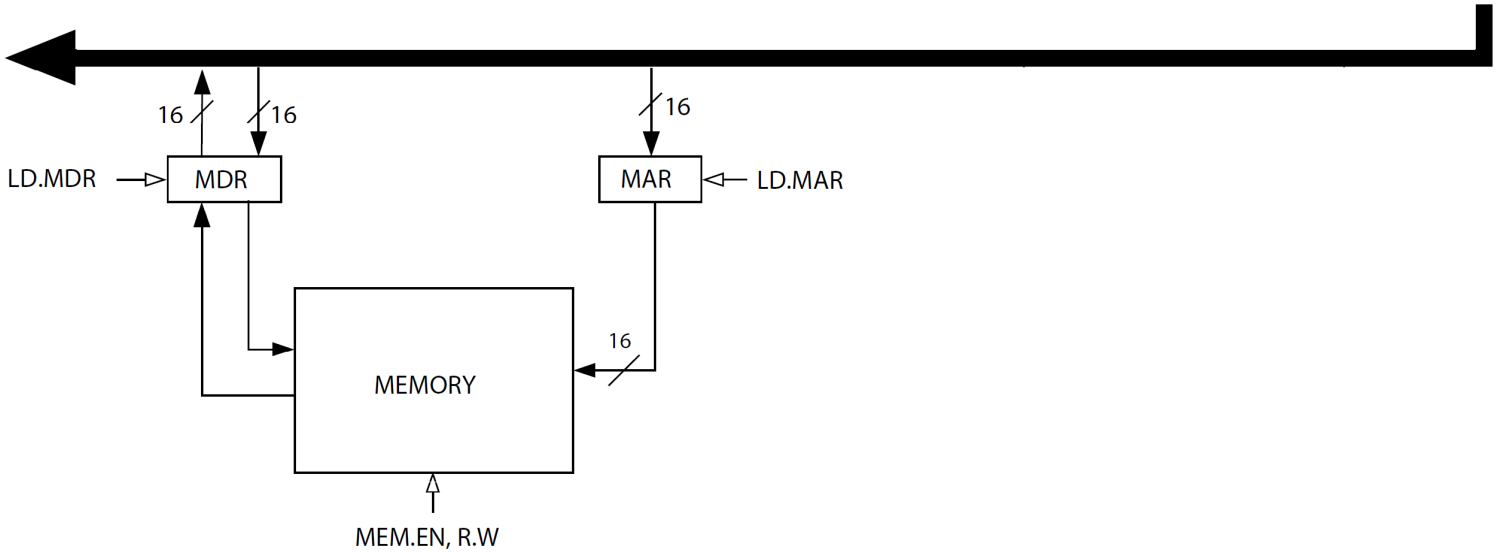
The LC-3 ISA

The LC-3


GateMARMUX
GatePC
16
MARMUX
LD.PC
PC
+1
REG
FILE
DR
3
16
16
2
PCMUX
LD.REG
16
16
16
SR2
OUT
SR1
OUT
SR2
3
SR1
3
ZEXT
[7:0]
+
16
16
ADDR2MUX
2
ADDR1MUX
16
16
16
16
16
16
[10:0]
SEXT
0
SEXT
16
[4:0]
[8:0]
SEXT
[5]
SR2MUX
16
[5:0]
SEXT
CONTROL
2
B
A
ALU
ALUK
R
IR
LD.IR
N
Z
P
LD.CC
16
16
LOGIC
16
GateALU

LC-3 Memory



Definition

The **address space** of a computer is the number of uniquely identifiable memory locations.

- **Addressability** is the number of bits at each address.
- The LC-3 has 16-bit addresses (an address space of 2^{16}), and 16 bits of addressability.

ADD (Addition, Register)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0001 (ADD)				Dest			Src1			0	0	0	Src2		

1. Adds the value of `Src1` to `Src2`
2. Places the result in `DestR`.

Example

Suppose `R1` contains `0x000A`. Then executing:

```
0001 010 001 000 001
```

. . . results in `R2` containing `0x0014`.

ADD (Addition, Immediate)

15 14 13 12	11 10 9	8 7 6	5	4 3 2 1 0
0001 (ADD)	Dest	Src1	1	Imm5

1. Sign-extends Imm5 to 16 bits, adds it to the value of Src1
2. Places the result in Dest.

Definition

Sign-extension increases the number of bits in a two's complement binary number by prepending it with copies of its MSB.

Example

Suppose R7 contains 0x000D. Then executing:

```
0001 101 111 1 11110
```

. . . results in R5 containing 0x000B.

AND (Bitwise "And", Register)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0101 (AND)				Dest			Src1			0	0	0	Src2		

1. Performs a bitwise "and" of the values in `Src1` and `Src2`
2. Places the result in `Dest`.

Definition

The **bitwise "and"** of two values contains a '1' in bit n if both values have a '1' in bit n, and a '0' otherwise.

Example

Suppose `R1` contains `0x000A` and `R2`, `0x0009`. Then executing:

```
0101 011 001 000 010
```

. . . results in `R3` containing `0x0008`.

AND (Bitwise "And", Immediate)

15 14 13 12	11 10 9	8 7 6	5	4 3 2 1 0
0101 (AND)	Dest	Src1	1	Imm5

1. Sign-extends Imm5 to 16 bits, "ands" it with the value of Src1
2. Places the result in Dest.

Example

Suppose R6 contains 0x000F. Then executing:

0001 100 110 1 00110

. . . results in R4 containing 0x0006.

NOT (Bitwise "Not")

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1001 (NOT)				Dest			Src1			1	1	1	1	1	1

1. Performs a bitwise "not" of the value in `Src1`
2. Places the result in `Dest`.

Definition

The **bitwise "not"** of a value contains a '1' in bit n if the value has a '0' in bit n, and a '0' otherwise.

Example

Suppose `R0` contains `0x0000`. Then executing:

```
1001 000 000 111111
```

. . . results in `R0` containing `0xFFFF`.

Computational Instructions

How do we compute `R2 = R0 - R1`? (3 instructions)

How do we compute `R1 = R0 % 2`? (1 instruction)

Definition

A **bit shift** moves each bit of a value to the left or to the right.

How do we left shift `R0`?

Computational Instructions

How do we multiply `R0` by 3? (2 instructions)

How do we multiply `R0` by 25 efficiently? (6 instructions)

How do we right shift `R0`? (JK, don't try this at home, many istructions)

Computational Instructions

Definition

The **bitwise "or"** of two values contains a '1' in bit n if either operand has a '1' in bit n, and a '0' otherwise.

Theorem

De Morgan's laws (for propositions) state:

- `NOT(p AND q) == (NOT p) OR (NOT q)`
- `NOT(p OR q)  == (NOT p) AND (NOT q)`

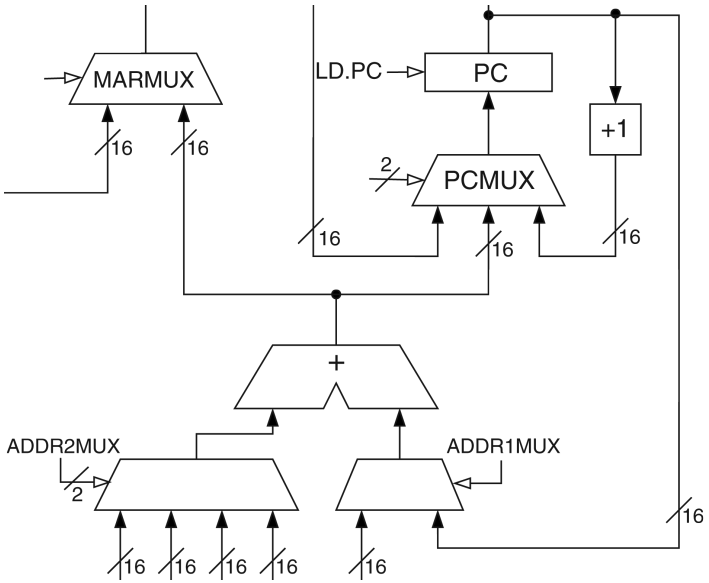
Compute `R2 = R0 OR R1`.

Load Instructions

4 different Load instructions. Why so many???

- Problem:
  - Need 16 bits to specify an address in memory.
  - Instructions are only 16 bits long.
  - Need 4 bits for the opcode, and 3 for destination register.
- 4 different solutions to the issue.
- Modern machines also encounter this issue.

LC-3 Addressing



- **PC-relative** uses an offset of the PC as an address.
- **Base+Offset** uses an offset of a register value as an address.
- **Indirect** uses another value in memory as an address.

LD (Load PC-Relative)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0010 (LD)				DestR			PCOffset9								

1. Sign-extends `PCOffset9` to 16 bits, adds it to the PC
2. Loads the value at that address into `DestR`.

Definition

PC-relative addressing uses an offset from the PC.

Example

Suppose address x3000 contains: `0010 010 111111101`

Executing this instructions results in `R2` containing the value at location `0x2FFE`.

LDI (Load Indirect)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1010 (LDI)				DestR			PCOffset9								

1. Sign-extends `PCOffset9` to 16 bits, adds it to the PC
2. Loads the value at that address.
3. Uses the value as another address, loads the value stored there, and places it in `DestR`.

Definition

Indirect addressing uses another value in memory.

Example

Suppose address x3009 contains: `1010 111 000000110` and address `0x3010` contains `0x2FFF`

Executing this instructions results in `R7` containing the value at address `0x2FFF`.

LDR (Load Base+Offset)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0110 (LDR)				DestR			BaseR			Offset6					

1. Sign-extends `Offset6` to 16 bits, adds it to `BaseR`
2. Loads the value at that address into `DestR`.

Definition

Base+Offset addressing uses an offset from a register's value.

Example

Suppose `R0` contains `0x3100`. Then executing:

```
0110 001 000 000010
```

. . . results in `R1`'s containing the value at location `0x3102`.

ST (Store PC-Relative)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0011 (ST)				SrcR			PCOffset9								

1. Sign-extends `PCOffset9` to 16 bits, adds it to the PC
2. Stores the value of `SrcR` at that address.

Example

Suppose address x3000 contains: `0011 010 111111101`

Executing this instructions results in address `0x2FFE` containing the value of `R2`.

STI (Store Indirect)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1011 (STI)				SrcR			PCOffset9								

1. Sign-extends `PCOffset9` to 16 bits, adds it to the PC
2. Loads the value at that address.
3. Treats the value as an address and stores the value of `SrcR` at the address.

Example

Suppose address x3009 contains: `1011 111 000000110` and address `0x3010` contains `0x2FFF`

Executing this instructions results in address `0x2FFF` containing the value of `R7`.

STR (Store Base+Offset)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0111 (STR)				SrcR			BaseR			Offset6					

1. Sign-extends `Offset6` to 16 bits, adds it to `BaseR`
2. Stores the value of `SrcR` at the resulting address.

Example

Suppose `R0` contains `0x3100`. Then executing:

```
0111 001 000 000010
```

. . . results in address `0x3102` containing the value of `R1`.

LEA (Load Effective Address)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1110 (LEA)				DestR			PCOffset9								

- Sign-extends `PCOffset9` to 16 bits, adds it to the value of the PC, and places the resulting address in `DestR`.

Example

Suppose address x3000 contains: `0010 010 111111101`

Executing this instructions results in `R2` containing `0x2FFE`.

Computational and Data Movement Instructions

What does this code do???

Example

Address	Instrustion
0x30F6	1110 0011 1111 1101
0x30F7	0001 0100 0110 1110
0x30F8	0011 0101 1111 1011
0x30F9	0101 0100 1010 0000
0x30FA	0001 0100 1010 0101
0x30FB	0111 0100 0100 1110
0x30FC	1010 0111 1111 0111

Control Flow Instructions

Definition

A **branch** is an instruction that alters the sequence of execution by modifying the PC.

- Branches can be **conditional** or **unconditional**.
- If, when executed, a branch is unconditional or its condition is true, then the branch is said to be **taken**.

Example

The LC-3 has three **condition codes**, `N`, `Z`, and `P`, indicating whether the last modified register is negative, zero, or positive.

- The codes are set by `ADD`, `AND`, `NOT`, `LD`, `LDR`, and `LDI`.
- Exactly one of the condition codes will be set at all times.

JMP (Jump, Unconditional, Register-Based)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1100 (JMP)				0	0	0	BaseR			0	0	0	0	0	0

- Unconditionally places the contents of BaseR into the PC.

Example

Suppose the R7 contains 0x3100. Then executing:

1100 000 111 000000

. . . results in the PC's containing 0x3100.

BR (Branch PC-Relative)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0000 (BR)				n	z	p	PCOffset9								

- If any of the following are true:
  - n (bit 11) is '1' and N (condition code) is set
  - z (bit 10) is '1' and Z (condition code) is set
  - p (bit 9) is '1' and P (condition code) is set
- Then, sign-extends PCOffset9 to 16 bits, adds it to the PC, and places the resulting address in the PC.
- Else, does nothing.

BR (Branch PC-Relative)

Example

Suppose the PC, *prior* to executing the instruction, contains 0x3010 and the condition code Z is set. Then executing:

```
0000 110 000001010
```

. . . results in the PC's containing 0x301B.

Example

Suppose the PC, *prior* to executing the instruction, contains 0x3010 and the condition code P is set. Then executing:

```
0000 110 000001010
```

. . . results in the PC's containing 0x3011.