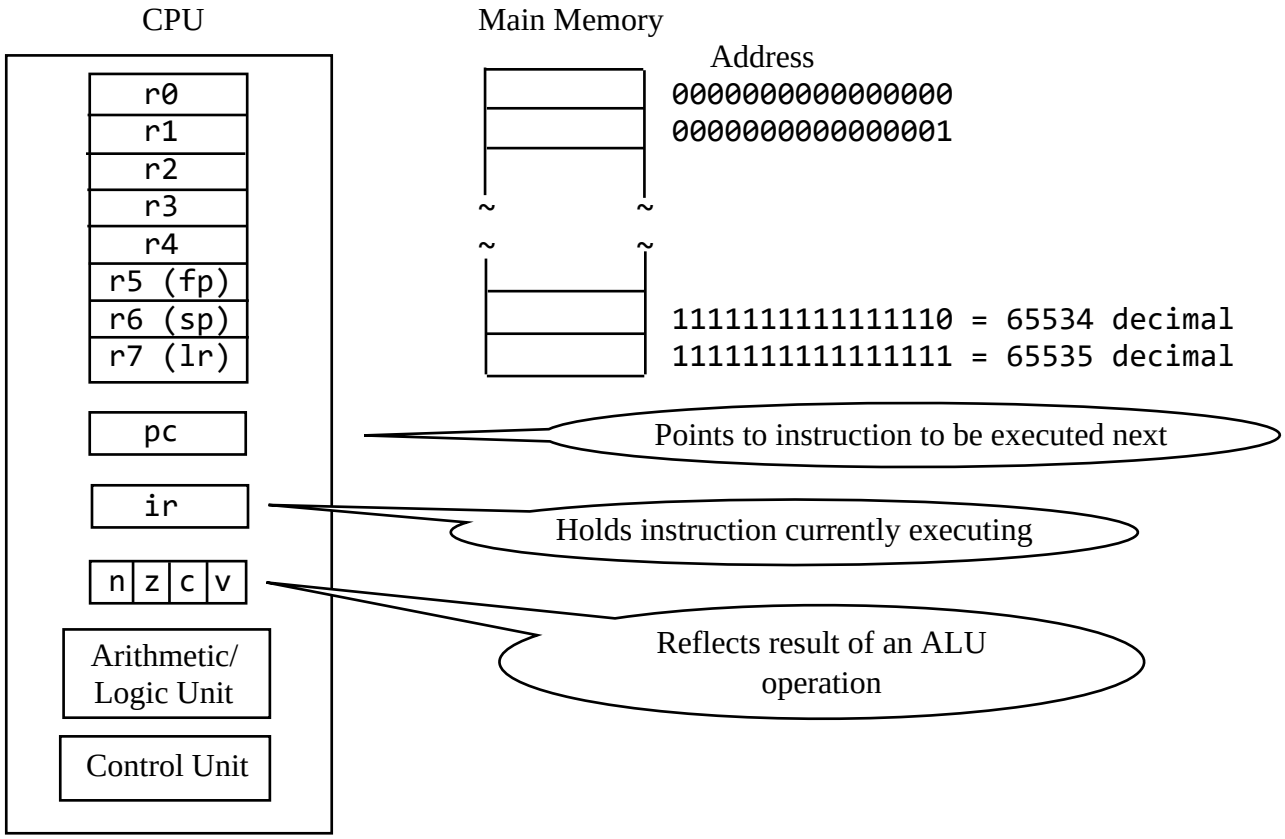
2 Machine Language

Structure of the LCC



CPU Cycle

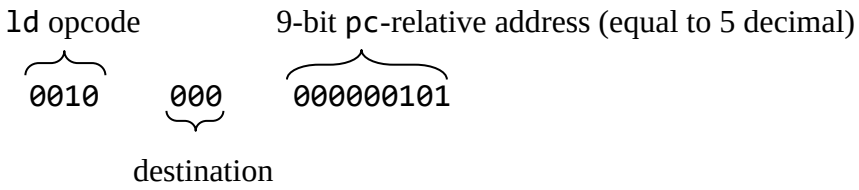
1. *Fetch* the instruction the `pc` register "points to." That is, the CPU loads the `ir` with the instruction in the memory cell whose address is in the `pc` register. The CPU does not remove the instruction from its memory cell. Instead it makes a copy of it. Thus, the contents of the memory cell that the `pc` register points to are unaffected.
2. *Increment* the `pc` register.
3. *Decode* the fetched instruction (i.e., determine its opcode).
4. *Execute* the instruction in the `ir`.

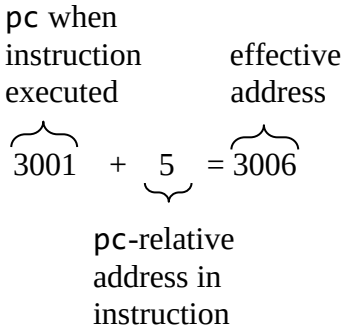
Simple Machine Language Program

Address (hex) Description of Instruction

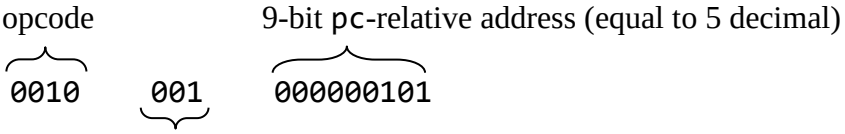
3000: Load `r0` with a copy of the number in memory at address 3006 hex.
3001: Load `r1` with a copy of the number in memory at address 3007 hex.
3002: Add the numbers in `r0` and `r1` and put the sum into `r0`.
3003: Display in decimal the contents of `r0`.
3004: Move the display cursor to the beginning of the next line on the screen.
3005: Halt.
3006: First number (the binary equivalent of 2 decimal)
3007: Second number (the binary equivalent of 3 decimal)

LD Instruction

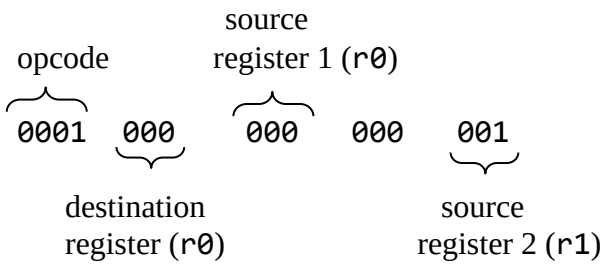




The second `ld` instruction is



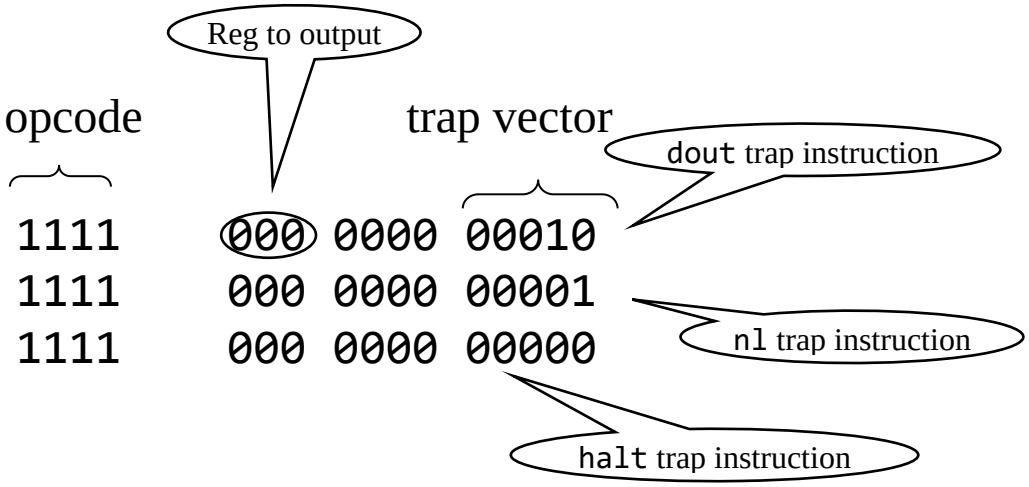
ADD Instruction



Specifying Instruction Formats

```
0010 dr pcoffset9          ld instruction

0001 dr sr1 000 sr2        add instruction
```

Trap Instructions



Data

```
0000000000000010
0000000000000011
```

Complete Programs

```
; ex0201.bin
0010 000  000000101   ; ld
0010 001  000000101   ; ld
0001 000  000 000 001 ; add
1111 000  0000 00010  ; dout
1111 000  0000 00001  ; nl
1111 000  0000 00000  ; halt
0000000000000010      ; data
0000000000000011      ; data
```

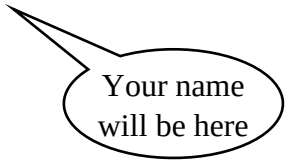
```
; ex0201.hex
2005  ; ld
2205  ; ld
1001  ; add
f002  ; dout
f001  ; nl
f000  ; halt
0002  ; data
0003  ; data
```

Using lcc Program

```
lcc ex0201.hex -L 0x3000

Starting interpretation of ex0201.e
lst file = ex0201.lst
bst file = ex0201.bst
================================================== Output
5
```

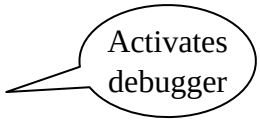
Lst File

```
LCC Assemble/Link/Interpret/Debug Ver 3.3  Mon Jun 1 15:58:21 2021
Dos Reis, Anthony J.

Header
o
C

Loc   Code
0000  2005 ; ld
0001  2205 ; ld
0002  1001 ; add
0003  f002 ; dout
0004  f001 ; nl
0005  f000 ; halt
0006  0002 ; 2
0007  0003 ; 3
====================================================== Output
5

========================================= Program statistics
Input file name       =      ex0201.hex
Instructions executed =    6 (hex)     6 (dec)
Program size          =    8 (hex)     8 (dec)
Max stack size        =    0 (hex)     0 (dec)
Load point            =    0 (hex)     0 (dec)
```



Debugger

```
lcc ex0201.hex -L 0x3000 -d
```



Display registers

```
r
```

To display `r0` only, enter

```
r0
```

To display the contents of all the memory in use, enter

```
m
```

To display the contents of the memory location corresponding to a label, say `sum` (we discuss labels in the next chapter), enter

```
msum
```

To display the contents a memory location, say at the address 3010 hex, enter

```
m3010
```

To display 10 locations, starting from 3000 hex, enter

```
m3000 10
```

To change the number of instructions executed each time you hit the Enter key, enter the number desired. For example, if you want five instructions executed each time you hit the Enter key, enter

```
5
```

To deactivate the debugger (which causes the program to execute to its end without pausing), enter

```
g
```

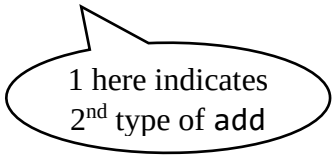
For a complete list of the debugger commands, see the file `lcc.txt` in the software package for this book.

Second Type of Add Instruction

```
  4    3    3    3    3
0001 dr  sr1 000 sr2
```

← Number of bits in the corresponding field

```
  4    3    3   1    5
0001 dr  sr1  1  imm5
```



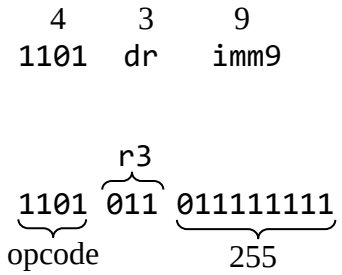

```
       dr    sr1       imm5
0001   000   011   1   00111
```

Store Instruction

```
  4    3       9
0011  sr  pcoffset9
```

`; ex0202.bin`	
`0010 000 000000101  ; ld`	load `r0` from location following `halt`
`0001 000 000 1 00011  ; add`	add immediate value 3 to `r0`
`0011 000 000000011  ; st`	store `r0` into location following `halt`
`1111 000 0000 00010  ; dout`	display `r0` in decimal
`1111 000 0000 00001  ; nl`	move cursor to next line
`1111 000 0000 00000  ; halt`	terminate execution
`0000000000000001  ; x`	initial value is 1

Move Immediate Instruction



And Instruction

```
1100110011001100
0101010101010101
0100010001000100 ← result of bitwise AND operation
```

bit is 1 because the two bits ANDed in this column are both 1.

```
  4     3   3    3    3
0101   dr sr1 000 sr2

  4     3   3   1    5
0101   dr sr1 1   imm5

       dr sr1     sr2
0101 000 000 000 001
```

```
r0: 0000000001100001 represents the letter 'a'
r1: 1111111111011111 mask
r0: 0000000001000001 represents the letter "A"
```

bit 5 in `r0` reset to 0

Not Instruction

```
  4    3   3     6
1001 dr sr1 111111


1001 000 000 111111
```

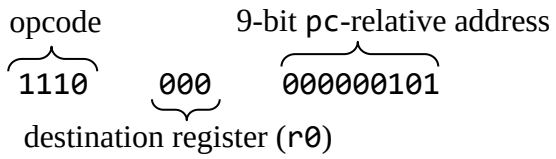
Doing an OR Using AND

A | B = not(not A & not B) DeMorgan's Law

ex0203.bin

`0010 000 000001000`	load first number into `r0`
`1001 000 000 111111`	flip the bits in `r0` with a `not` instruction
`0010 001 000000111`	load second number into `r1`
`1001 001 001 111111`	flip the bits in `r1` with a `not` instruction
`0101 000 000 000 001`	AND `r0` with `r1` , and place result in `r0`
`1001 000 000 111111`	flip the bits in `r0` with a `not` instruction
`1111 000 0000 00100`	display `r0` in hex with a `hout` instruction
`1111 000 0000 00001`	move cursor to the next line with `nl` instruction
`1111 000 0000 00000`	halt
`1100101011001010`	first number
`1010110010101100`	second number

Load Effective Address Instruction



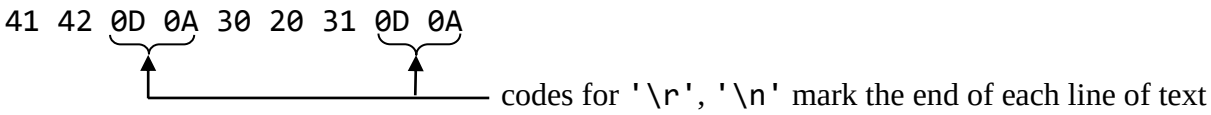
Strings

```
'A':   01000001 (41 hex, 65 decimal)
'a':   01100001 (61 hex, 97 decimal)
'B';   01000010 (42 hex, 66 decimal)
'b':   01100010 (62 hex, 98 decimal)
'0':   00110000 (30 hex, 48 decimal)
'1':   00110001 (31 hex, 49 decimal)
' ':   00100000 (20 hex, 32 decimal)
'\n':  00001010 (0A hex, 10 decimal)
'\r':  00001101 (0D hex, 13 decimal)
```

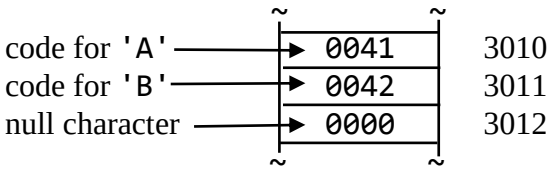
This file

```
AB
0 1
```

is represented with the following sequence of ASCII codes (given in hex):



Sting “AB” in memory:



String Program

ex0204.bin

1110 000 000000010	`lea` instruction
1111 000 0000 00110	`sout` trap instruction
1111 000 0000 00000	`halt` instruction
00000000 01101000	`'h'`
00000000 01101001	`'i'`
00000000 00001010	`'\n'`
0000000000000000	null character

Trap Instructions

Mnemonic	Binary Format	Flags Set	Description
`halt`	`1111 000 0000 00000`	none	Stop execution, return to OS
`nl`	`1111 000 0000 00001`	none	Output newline
`dout`	`1111 sr 0000 00010`	none	Display signed number in `sr`
`udout`	`1111 sr 0000 00011`	none	Display unsigned number in `sr` in decimal
`hout`	`1111 sr 0000 00100`	none	Display hex number in `sr` in hex
`aout`	`1111 sr 0000 00101`	none	Display ASCII character in `sr`
`sout`	`1111 sr 0000 00110`	none	Display string `sr` points to
`din`	`1111 dr 0000 00111`	none	Read decimal number from keyboard into `dr`
`hin`	`1111 dr 0000 01000`	none	Read hex number from keyboard into `dr`
`ain`	`1111 dr 0000 01001`	none	Read ASCII character from keyboard into `dr`
`sin`	`1111 sr 0000 01010`	none	Input string into buffer `sr` points to

If `sr` or `dr` is omitted in a trap assembly language instruction, it defaults to `r0` (000).

Program That Converts Decimal to Binary

ex0205.bin

```
1111 000 0000 00111 din
1111 000 0000 00100 hout
1111 000 0000 00000 halt
```