Copyright © 2021 Anthony J. Dos Reis

**3** Assembly Language

1st vs 2nd editions

* .word in 2nd edition is equivalent to .fill in 1st edition
* bl in 2nd edition is equivalent to jsr in 1st edition
* .asciz and .string in 2nd edition is equivalent to .stringz in 1st edition
* .zero and .space in the 2nd edition is equivalent to .blkw in 1st edition
* The LCC assembler supports both 1st and 2nd editions
* If the shift count (2nd edition) is omitted, it defaults to 1.
* If a register is not specified in an I/O trap instruction, it defaults to r0.

; ex0301.bin ; ex0301.a

0010 000 000000101 ld r0, x

0010 001 000000101 ld r1, y

0001 000 000 000 001 add r0, r0, r1

1111 000 0000 00010 dout r0

1111 000 0000 00001 nl

1111 000 0000 00000 halt

0000000000000010 x: .word 2

0000000000000011 y: .word 3

*Rule*: Mnemonics, directives, and register names, but not labels, are *case insensitive*. That is, they can be in either upper or lower case in an assembly language program. Thus, ld and LD are equivalent but not the labels x and X. *Only labels start in column 1* in an assembly language instruction.

x: .word 3 ; translated to 0000000000000011 (3 in binary)

y: .word 'A' ; translated to 0000000001000001 (ASCII code for 'A')

z: .word x ; translated to the address of x

Insert before halt:

hout r0 ; displays value in r0 in hex

nl ; move cursor to beginning of next line

Assembling and Executing ex0301.a

Run lcc program:

lcc ex0301.a

Starting assembly pass 1

Starting assembly pass 2

Starting interpretation of ex0301.e

lst file = ex0301.lst

bst file = ex0301.bst

====================================================== Output

5

ex0301.lst file Produced by the lcc Program

LCC Assemble/Link/Interpret/Debug Ver 3.3 Mon Jun 14 16:50:46 2021

Dos Reis, Anthony J.

Nothing here so empty

header for this program

File signature

Header

o

C

Terminates header

Loc Code Source Code

Machine code part of executable file

; ex0301.a

0000 2005 ld r0, x

0001 2205 ld r1, y

0002 1001 add r0, r0, r1

0003 f002 dout r0

0004 f001 nl

0005 f000 halt

0006 0002 x: .word 2

0007 0003 y: .word 3

====================================================== Output

5

========================================== Program statistics

Input file name = ex0301.a

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)

Sub Instruction

add r0, r1, r2 ; adds r1 and r2, result goes into r0

sub r0, r1, r2 ; subtracts r2 from r1, result goes into r0

add r0, r1, 1 ; add r1 and 1, result goes into r0

sub r0, r1, 1 ; subtract 1 from r1, result goes into r0

sub r0, r0, r0

.zero (.blkw) and .string (.stringz, .asciz) Directives

buffer: .zero 100

This single directive is equivalent to

buffer: .word 0

.word 0

100 .word directives

.word 0

greeting: .string "Hello, world"

1 ; ex0302.a

2 lea r0, prompt ; get address of prompt message

3 sout r0 ; display prompt message

4 lea r0, buffer ; get address of buffer

5 sin ; read string from keyboard

6 sout r0 ; echo string to display

7 halt

8 prompt: .string "Enter string\n"

9 buffer: .zero 100

Mov Pseudo-Instruction

mvi r0, 5 ; moves 5 into r0

add r1, r0, 0 ; moves contents of r0 into r1

mvr r1, r0

mov r0, 5 ; move 5 into r0

mov r1, r0 ; move contents of r0 into r1

mov r0, 'A'

Branch Instructions and Loops

4 3 9

0000 code pcoffset9

code branch occurs if

brz or bre 000 z = 1 (branch on zero, branch on equal)

brnz or brne 001 z = 0 (branch on nonzero, branch on not equal)

brn 010 n = 1 (branch on negative)

brp 011 n = z (branch on positive)

brlt 100 n v (branch on less than after a compare)

brgt 101 n = v and z = 0 (branch on greater than after compare)

brc 110 c = 1 (branch on carry)

br 111 (unconditional branch)

add r0, r0, r1

brz dog

Add First 10 Odd Numbers

1 ; ex0303.a

2 mov r1, 0 ; sum initially is 0

3 mov r2, 1 ; initialize r2 to first odd number

4 mov r3, 2 ; r3 used to get next odd number in series

5 mov r4, 10 ; 10 is number of odd numbers to sum

6 loop: add r1, r1, r2 ; add odd number in r2 to r1

7 add r2, r2, r3 ; add 2 to r2 to get next odd number

8 sub r4, r4, 1 ; decrement count

9 brp loop ; do loop again if count in r4 is positive

10 lea r0,s ; get address of string

11 sout r0 ; display "Sum = "

12 dout r1 ; display the sum

13 nl ; move cursor to the next line

14 halt

15 s: .string "Sum = "

Calling Subroutines

Calling module Subroutine (called module)

calling instruction

next instruction

return instruction

bl (branch and link) instruction:

4 1 11

0100 1 pcoffset11

1 ; ex0304.a

2 main: bl sub ; saves return address (address of 2nd bl) in r7

3 bl sub ; saves return address (address of halt) in r7

4 halt

5 ;================

6 sub: lea r0, msg

7 sout r0

8 ret ; return to address in r7

9 msg: .string "Hello\n"

Can also use the mnemonics jsr and call in place of bl.

Second form of branch and link (register in place of a label as operand). Use mnemonic blr:

blr r5 ; jump to address in r5

Header in an Executable File

1 ; ex0305.a Infinite loop

2 sub: lea r0, msg Execution starts here but it should NOT!

3 sout r0

4 ret Returns to lea instruction

5 msg: .string "Hello\n"

6 ;================

7 main: bl sub ; saves return address (address of next bl) in r7

8 bl sub ; saves return address (address of halt) in r7

9 halt

1 ; ex0306.a

2 .start main Indicates main is the entry point

3 sub: lea r0, msg

4 sout r0

5 ret

6 msg: .string "Hello\n"

7 ;================

8 main: bl sub Entry point is here

9 bl sub

10 halt

.start Directive

LCC Assemble/Link/Interpret/Debug Ver 3.3 Mon Jun 14 16:02:06 2021

Dos Reis, Anthony J.

Entry point address relative to beginning of program

Header

o

S 000a

C

Loc Code Source Code

Specifies entry point

; ex0306.a

.start main

0000 e002 sub: lea r0, msg

0001 f006 sout r0

0002 c1c0 ret

0003 0048 msg: .string "Hello\n"

0004 0065

0005 006c

0006 006c

0007 006f

0008 000a

0009 0000

; ==========================

000a 4ff5 main: bl sub

Here is the entry point

000b 4ff4 bl sub

000c f000 halt

====================================================== Output

Hello

Hello

========================================== Program statistics

Input file name = ex0306.a

Instructions executed = 9 (hex) 9 (dec)

Program size = d (hex) 13 (dec)

Max stack size = 0 (hex) 0 (dec)

Load point = 0 (hex) 0 (dec)LCC

1 ; ex0307.a

2 ld r0, y ; translated to 0010 000 000000010

3 halt ; translated to 1111 0000 00100101

4 x: .word 7 ; translated to 0000000000000111 (7 decimal)

5 y: .word x ; translated to 0000000000000010 (address of x)

1 ; ex0308.a

2 ld r0, y ; translated to 0010 000 000000010

3 halt ; translated to 1111 0000 00000000

4 x: .word 7 ; translated to 0000000000000111 (7 decimal)

5 y: .word 2 ; translated to 0000000000000010 (2 decimal)

lcc ex0307.e -r -m

lcc ex0307.e -L 0x3000 -r -m

---------------------------------------------- Memory display

3000: 2002

Adjusted

address

3001: f000

3002: 0007

3003: 3002

Adjusted

address

--------------------------------------- End of memory display

-------------------------------------------- Register display

n = 0 z = 0 p = 1 pc = 3002 ir = 0007

r0 = 3002 r1 = 0000 r2 = 0000 r3 = 0000

r4 = 0000 fp = 0000 sp = 0000 lr = 0000

------------------------------------- End of register display

A 0003 is in ex0307.e

Dereferencing Pointers

The last two lines of the program in ex0307.a are

x: .word 7 ; translated to 0000000000000111 (7 decimal)

y: .word x ; translated to 0000000000000010 (the address of x)

Address

2002 0

F000 1

x 0007 2

y 0002 3 This location points to location 2

~ ~

x

0007

y

ld r0, y ; load r0 from y

Thus, after this instruction is executed r0 points to x (i.e., it contains the address of x).

r0 x

0007

y

LDR Instruction

4 3 3 6

0110 dr baser offset6

Suppose we then execute, where r0 contains the address of x.

ldr r1, r0, 0 ; load r1 from address given by r0 + 0

add r1, r1, 1 ; add 1 to r1

str r1, r0, 0 ; stores incremented value into x

r0 x

~~0007~~ 0008

y

x offsets

8 0

-7 1

y 105 2

ld r0, y ; load r0 with the pointer in y

ldr r1, r0, 2 ; load from address given by r0 + 2

Accessing the Stack

push r3

pop r4

sp ~ ~

25

-9

fp 7

100

52

19

-1

~ ~

ldr r2, fp, 2

ldr r2, fp, -3

str r0, fp, 1

sub sp, sp, 1

This instruction transforms the stack configuration from that shown above to the following:

~ ~

sp reserved

25

-9

fp 7

100

52

19

-1

~ ~

Comparing Numbers

Incorrect!!!

ld r0, x

ld r1, y

sub r0, r0, r1 ; subtract r1 from r0, result goes into r0

brn less ; branch if x < y

brz equal ; branch if x = y

brp greater ; branch if x > y

x: .word 5

y: .word 10

Correct!!!

1 ld r0, x ; x is the top number

2 ld r1, y ; y is the bottom number

3 sub r0, r0, r1 ; subtract r1 from r0, result goes into r0

4 brlt less ; branch if x < y

5 brz equal ; branch if x = y

6 brgt greater ; branch if x > y

7

8 x: .word 5

9 y: .word 10

The brlt instruction branches if n v (i.e., the n flag does not equal the v flag). Thus, it branches if

n = 1 and v = 0

or

n = 0 and v = 1.

The brlt instruction does not branch if n = v. Thus, it does not branch if

n = 0 and v = 0

or

n = 1 and v = 1

*Rule*: When a brlt instruction follows a subtraction of two signed numbers, the brlt instruction branches if and only if the top number in the subtraction is less than the bottom number.

Program That Uses the brlt, brz, and br Instructions

1 ; ex0309.a

2 lea r0, prompt ; prompt for numbers

3 sout r0

4 din r0 ; read in first number

5 din r1 ; read in second number

6 sub r0, r0, r1 ; subtract 2nd number from 1st number

7 brlt less ; branch accordin r0g to the result

8 brz equal

9 brgt greater ; can use br here instead of brgt

10 less: lea r0, msglt

11 br display ; unconditional branch to display

12 equal: lea r0, msge

13 br display ; unconditional branch to display

14 greater: lea r0, msgg

15 display: sout r0 ; display result

16 halt

17 prompt: .string "Enter two signed numbers\n"

18 msglt: .string "First number less\n"

19 msge: .string "Numbers equal\n"

20 msggt: .string "First number greater\n"

Comparing Unsigned Numbers

1 ld r0, x ; x is the top number

2 ld r1, y ; y is the bottom number

3 sub r0, r0, r1 ; subtract r1 from r0, result goes into r0

4 brc below ; branch if x < y

5 brz equal ; branch if x = y

6 br above ; otherwise, branch if x > y

7

8 x: .word 5

9 y: .word 10

CMP Instruction

cmp r0, r1 ; subtracts r1 from r2 (r0 and r1 unaffected)

brlt less

cmp r0,5

brlt less

*Rule*: If the result of a subtraction is not needed, compare values using cmp instead of sub.

Assembly Process

To translate an assembly language instruction, an assembler needs to know

1. the opcode for the mnemonic
2. if the instruction includes any register names, the numbers that denote those registers
3. if the instruction specifies an operand using a label, the address corresponding to that label

ld r0, x

Opcode table Register table

br 0000 r0 000

add 0001 r1 001

ld 0010 r2 010

r7 111

fp 101

sp 110

lr 111

Symbol Table (built in Pass 1)

1 ; ex0310.a

2 ld r0, x

3 dout r0

4 halt

5 x: .word 5

Symbol Address

x 0000000000000011

Computing PC-Relative Address (Pass 2)

1 ; ex0310.a

2 ld r0, x

3 dout r0

4 halt

5 x: .word 5

6 y: .word x

Symbol Address

x 0000000000000011

y 0000000000000100

pc-relative address = (address of x from symbol table) – (location\_counter + 1)

Computed from location\_counter

From and the address in the

opcode table symbol table for x

0010 000 0000000000000010

From

register table

An assembler is so called because its principal activity is to assemble (i.e., put together) machine instructions from its component parts.

Label Offsets and the Current Location Marker

1 ld r0, x ; loads 5

2 ld r1, x+1 ; loads 11

3 ld r2, x+2 ; loads 17

4 ld r3, y-2 ; loads 5

5 halt

6 x: .word 5

7 .word 11

8 y: .word 17

9 z: .word x+2 ; assembled to the address of y

brp \*+3 ; branch on positive to hout r0 instruction  
 dout r0  
 br \*+2 ; unconditional branch to halt instruction  
 hout r0   
 halt

More Instructions

jmp

jmp r3

The ret instruction is a special case of the jmp instruction. The assembler translates the ret instruction to

jmp r7

Shift Instructions

r1 before srl r1,1 r1 after srl r1

1111111111111100 0111111111111110 n set to 0, z set to 0, c set to 0

----------------------------------------------------------------------------------------

r1 before sra r1,1 r1 after sra r1

1111111111111101 1111111111111110 n set to 1, z set to 0, c set to 1

----------------------------------------------------------------------------------------

r1 before sll r1,1 r1 after sll r1

1000000000000000 0000000000000000 n set to 0, z set to 1, c set to 1

Extended Opcode Instructions

push 1010 sr 0000 00000 mem[--sp] = sr

pop 1010 dr 0000 00001 nz dr = mem[sp++];

srl 1010 sr ct 00010 nzc sr >> ct (0 inserted on left, c=last out)

nop 1010 000 0000 00010 no operation

sra 1010 sr ct 00011 nzc sr >> ct (sign bit replicated, c=last out)

sll 1010 sr ct 00100 nzc sr << ct (0 inserted on right, c=last out)

rol 1010 sr ct 00101 nzc sr << ct (rotate: bit 15 → bit 0, c=last out)

ror 1010 sr ct 00110 nzc sr << ct (rotate: bit 0 → bit 15, c=last out)

mul 1010 dr sr 0 00111 nz dr = dr \* sr

div 1010 dr sr 0 01000 nz dr = dr / sr

rem 1010 dr sr 0 01001 nz dr = dr % sr

or 1010 dr sr 0 01010 nz dr = dr | sr (bitwise OR)

xor 1010 dr sr 0 01011 nz dr = dr ^ sr (bitwise exclusive OR)

LCC Instruction Set Summary

Mnemonic Format Flags Set Description

br-- 0000 code pcoffset9 on code, pc = pc + pcoffset9

add 0001 dr sr1 000 sr2 nzcv dr = sr1 + sr2

add 0001 dr sr1 1 imm5 nzcv dr = sr1 + imm5

ld 0010 dr pcoffset9 dr = mem[pc + pcoffset9)

st 0011 sr pcoffset9 mem[pc + pcoffset9] = sr

bl 0100 1 pcoffset11 lr= pc; pc = [pc + pcoffset11]

bl 0100 000 baser 000000 lr = pc; pc = baser

and 0101 dr sr1 000 sr2 nz dr = sr1 & sr2

and 0101 dr sr1 1 imm5 nz dr = sr1 & imm5

ldr 0110 dr baser offset6 dr = mem[baser + offset6]

str 0111 sr baser offset6 mem[baser + offset6] = sr

cmp 1000 000 sr1 000 sr2 nzcv sr1 - sr2 (set flags)

cmp 1000 000 sr1 1 imm5 nzcv sr1 - imm5 (set flags)

not 1001 dr sr1 000000 nz dr = ~sr1

push 1010 sr 0000 00000 mem[--sp] = sr

pop 1010 dr 0000 00001 dr = mem[sp++];

srl 1010 sr ct 00010 nzc sr >> ct (0 inserted on left, c=last out)

nop 1010 000 0000 00010 no operation

sra 1010 sr ct 00011 nzc sr >> ct (sign bit replicated, c=last out)

sll 1010 sr ct 00100 nzc sr << ct (0 inserted on right, c=last out)

rol 1010 sr ct 00101 nzc sr << ct (rotate: bit 15 → bit 0, c=last out)

ror 1010 sr ct 00110 nzc sr << ct (rotate: bit 0 → bit 15, c=last out)

mul 1010 dr sr 0 00111 nz dr = dr \* sr

div 1010 dr sr 0 01000 nz dr = dr / sr

rem 1010 dr sr 0 01001 nz dr = dr % sr

or 1010 dr sr 0 01010 nz dr = dr | sr (bitwise OR)

xor 1010 dr sr 0 01011 nz dr = dr ^ sr (bitwise exclusive OR)

sub 1011 dr sr1 000 sr2 nzcv dr = sr1 - sr2

sub 1011 dr sr1 1 imm5 nzcv dr = sr1 - imm5

jmp 1100 000 baser offset6 on code, pc = baser + offset6

ret 1100 000 111 offset6 pc = lr + offset6

mvi 1101 dr imm9 dr = imm9

lea 1110 dr pcoffset9 dr = pc + pcoffset9

mov dr, imm9 is a pseudo-instruction translated to mvi dr, imm9.

mov dr, sr is a pseudo-instruction translated to add dr, sr, 0.

dr, sr, sr1, sr2, baser are 3-bit register fields.

ct is a 4-bit shift count field (if omitted in a shift assembly instruction, it defaults to 1).

pcoffset9, pcoffset11, imm5, imm9, offset6 are signed number fields of the indicated length.

If offset6 is omitted in an assembly language instruction, it defaults to 0.

Trap Instructions (call OS)

Mnemonic 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).

Debugging Instructions

Mnemonic Format Flags Set Description

m 1111 000 0000 01011 none Display all memory in use

r 1111 000 0000 01100 none Display all registers

s 1111 000 0000 01101 none Display stack

bp 1111 000 0000 01110 none Software breakpoint (activates debugger)

Branch Instruction Codes (same suffixes can be used on the jmp instruction)

Mnemonic Code Description

brz or bre 000 Zero or equal

brnz or brne 001 Nonzero or not equal

brn 010 Negative (signed number)

brp 011 Positive (signed number)

brlt 100 Less then (signed comparison)

brgt 101 Greater than (signed comparison)

brc or brb 110 Carry or less than (unsigned comparison)

br 111 Unconditional

Assembler Directives

Directive Description

.word <value> Create word initialized to <value>

.fill <value> Same as .word

.zero <size> Create block of <size> words initialized to 0

.space <size> Same as .zero

.blkw <size> Same as .zero

.string <string> Create null-terminated ASCII <string>

.stringz <string> Same as .string

.asciz <string> Same as .string

.start Mark entry point (or use \_start label)

.global <var> Specify <var> is a global variable

.globl <var> Same as .globl

.extern <var> Specify <var> is an external variable

.orig <address> Reset location counter to higher <address>