ECE3221 Lab Report 5

Stephen Cole

3553803

Part One:

1. Compile lab5a.c

Figure 1.1

A screenshot of a social media post

Description automatically generated

2. Use the timing result and the count value 2n and calculate how many loops per second are happening.

Time = 3.15 seconds, r2(counter after 3.15) = 0x004c0456 or 4981846

Counts per second = r2 / Time ≅ 1581538.41 ≅ 1580000 counts/loops per second

3. Compare the instructions found in the loop to the code that you might have written in assembly language to accomplish the fast counter. Comment on the ability of the C-compiler to write efficient Nios-II assembly language.

a)

i. What register is used for the count variable?

Register 2 is being used for the count variable.

ii. How was the address variable set to 0x10000000?

movhi r4, 4096

movhi r8, 4096

movhi r4, r4, 4100

addi r4, r4, 4100

addi r8, r8, 4096

add r7, r5, r6

iii. How many instructions are in the loop?

The loop includes 3 instructions counting the branch statement. The ‘stwio’ instruction, incrementing the counter with an ‘addi’ and ‘br’ to the location of the ‘stwio’.

Comment on the ability of the C-compiler to write efficient Nios-II assembly language.

The C-Compiler is able to generate very efficient Nios-II assembly language; however, it is slightly more verbose than human written Nios-II assembly.

b)

LPS = 1580000 loops per second, IPL = 3 instructions per loop

Instructions per second = IPL x LPS = 4740000

4. Modify the program by omitting the line that increments count and instead use count++ when the count variable is sent to the LEDs. This change gives the same overall result as the count variable is simply incremented after it is used. Compile the program and again examine the generated assembly language. Comment on your observations.

After incrementing the variable count by using ‘count++;’ instead of ‘count = count + 1;’ and compiling I noticed that in both cases the assembly instruction is the same ‘addi r2, r2, 1’.

Part Two:

Figure 2.1

A screenshot of a social media post

Description automatically generated

Histogram Values:

Figure 2.2

A screenshot of a cell phone

Description automatically generated

320 Barcode Pixels:

Figure 2.3

A screenshot of a social media post

Description automatically generated

Bar Width:

Figure 2.4

A screenshot of a cell phone

Description automatically generated

Gathering Bars in 5 bit groups:

Figure 2.5

A screenshot of a cell phone

Description automatically generated

Assembly Code:

#include <stdlib.h>

#include <stdio.h>

#define NUM\_BLACK\_BARS 100

//-------------------------------------------

// prototypes

void code25writer(char \*data); // supplied below

// develop the following subroutines

void writexy(int x, int y, int color);

int readxy(int x, int y);

int intensity( int vga16 );

void waitonPB0( void );

void code25reader(void);

//-------------------------------------------

//-------------------------------------------

// MAIN PROGRAM

static char LUT[32] = {-1};

int main()

{

LUT[6] = 0;

LUT[17] = 1;

LUT[9] = 2;

LUT[24] = 3;

LUT[5] = 4;

LUT[20] = 5;

LUT[12] = 6;

LUT[3] = 7;

LUT[18] = 8;

LUT[10] = 9;

while(1) // here: br here

{

code25writer("569632"); // generate Code25 barcode

waitonPB0();

code25reader();

waitonPB0();

}

} // end main

//-------------------------------------------

// SUBROUTINES

int readxy(int x, int y)

{

// Ensure x and y are in bounds

if( (x<0) || (x>319) ) return 0;

if( (y<0) || (y>239) ) return 0;

// Load value of x shifted and y shifted to correct value

return \_\_builtin\_ldhio((short\*)((x|16<<0x16|y<<9|0)\*2));

}

void writexy(int x, int y, int rgb)

{

// Ensure x and y are in bounds

if( (x<0) || (x>319) ) return;

if( (y<0) || (y>239) ) return;

// Store value int x shifted and y shifted to correct value

\_\_builtin\_sthio((short\*)((x|16<<0x16|y<<9|0)\*2), rgb);

}

int intensity(int pixel)

{

// Read rgb value from pixel color value

int blue = pixel & 0b11111;

pixel >>= 6;

int green = pixel & 0b11111;

pixel >>= 5;

int red = pixel & 0b11111;

return (red + green + blue) / 3;

}

void waitonPB0()

{

printf("Waiting for user to press push button!\n");

// Check current state

int button = \_\_builtin\_ldwio((void\*)(0x10000050));

// Check if button is pressed

while(button != 1)

{

button = \_\_builtin\_ldwio((void\*)(0x10000050));

}

// Check if button is released

while(button != 0)

{

button = \_\_builtin\_ldwio((void\*)(0x10000050));

}

}

void code25reader()

{

int histogram[32] = {0};

int bY, min = 31; // x and y values of a likely bar

for(int i = 0; i < 320; ++i)

{

for(int j = 0; j < 240; ++j)

{

int bin = intensity(readxy(i,j));

++histogram[bin];

if(min > bin)

{

min = bin;

bY = j;

}

}

}

// histogram values

// for(int i = 0; i < 32; ++i) printf("%d, ", histogram[i]);

// location of bar

// for(int i = 0; i < 320; ++i) writexy(i, bY, 0xF800);

int bar[320] = {0};

for(int i = 0; i < 320; ++i)

{

int bin = intensity(readxy(i,bY));

if(bin <= 11)

{

bar[i] = 0;

}

else

{

bar[i] = 1;

}

}

// Printout of bar

// for(int i = 0; i < 320; ++i) printf("%d", bar[i]); printf("\n");

int width[NUM\_BLACK\_BARS] = {-1};

int lineCount = 0;

int barCount = 0;

// Used to optimize finding the normalized width

int minWidth;

while(lineCount < 320)

{

while(bar[lineCount] == 0)

{

if(width[barCount] == -1) width[barCount]++;

width[barCount]++;

lineCount++;

if(bar[lineCount] == 1)

{

if(minWidth > width[barCount]) minWidth = width[barCount];

barCount++;

}

}

++lineCount;

}

// Print out width

//for(int i = 0; i < NUM\_BLACK\_BARS && width[i] != -1; ++i) printf("%d ", width[i]); printf("\n");

// Normalize width

for(int i = 0; i < NUM\_BLACK\_BARS && width[i] != -1; ++i)

{

width[i] /= minWidth;

width[i] -= 1;

}

// Print out width

//for(int i = 0; i < NUM\_BLACK\_BARS && width[i] != -1; ++i) printf("%d ", width[i]); printf("\n");

int sReg = 0;

int count = 0;

while((sReg & 7) != 6)

{

sReg <<= 1;

sReg += width[count++];

}

int offset = count;

char bit0, bit1, bit2, bit3, bit4;

int res[NUM\_BLACK\_BARS / 5] = {-1};

char resString[NUM\_BLACK\_BARS / 5] = {'\0'};

count = 0;

while(1)

{

bit0 = width[offset];

bit1 = width[offset+1];

bit2 = width[offset+2];

bit3 = width[offset+3];

bit4 = width[offset+4];

int test = bit0 \* 4 + bit1 \* 2 + bit2;

if(test == 0b101 && (bit3 == -1 || bit4 == -1))

break;

sReg = test << 2;

sReg += bit3 \* 2 + bit4;

res[count++] = LUT[sReg];

printf("%c%d%d%d%d%d",count==1?'\0':':',bit0,bit1,bit2,bit3,bit4);

offset += 5;

}

printf("\x1B[31m\n");

for(int i = 0; i < count; ++i)

{

resString[i] = '0' + res[i];

printf("%c", resString[i]);

}

printf("\x1B[30m\n");

}

// END

//-------------------------------------------

//===========================================

// \*\* DO NOT MODIFY CODE BEYOND THIS LINE \*\*

//===========================================

// Lab5 barcode display libary routines

//-------------------------------------------

// outhex7seg May 2020 RTervo

// write 32-bits in hex to the 7-segment displays

void outhex7seg( int x ){

union hex {

char b[4]; // as 4 bytes

int w; // as 1 word

} buf;

char segments[] = {

0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,

0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71

};

buf.b[3] = segments[ (x >> 28) & 0x0F ];

buf.b[2] = segments[ (x >> 24) & 0x0F ];

buf.b[1] = segments[ (x >> 20) & 0x0F ];

buf.b[0] = segments[ (x >> 16) & 0x0F ];

\_\_builtin\_stwio ((void \*)0x10000030,buf.w);

buf.b[3] = segments[ (x >> 12) & 0x0F ];

buf.b[2] = segments[ (x >> 8) & 0x0F ];

buf.b[1] = segments[ (x >> 4) & 0x0F ];

buf.b[0] = segments[ (x >> 0) & 0x0F ];

\_\_builtin\_stwio ((void \*)0x10000020,buf.w);

} // end outhex7seg

//-------------------------------------------

//-------------------------------------------

// fill7seg May 2020 RTervo

// sends the provided segment pattern

// to all eight 7-seg displays

// e.g. fill7seg(0) blanks the displays

void fill7seg(char segs){

int x,four=0;

for(x=0;x<4;x++) four = (four << 8) | segs;

\_\_builtin\_stwio ( (void \*) 0x10000020,four); // fill 7seg hi

\_\_builtin\_stwio ( (void \*) 0x10000030,four); // fill 7seg lo

} // fill7seg

//-------------------------------------------

//-------------------------------------------

// clrchar May 2020 RTervo

// clear the vga character buffer

void clrchar(void) {

int x;

for(x=0;x<5000;x++)

\_\_builtin\_stbio ((void \*)(0x09000000 + x), 0);

} // clrchar

//-------------------------------------------

//-------------------------------------------

// imager May 2020 RTervo

void imager(int x, int y, int z) {

if( (x<0) || (x>319) ) return;

if( (y<0) || (y>239) ) return;

int m=0,n=9,r=rand()&8191>>3;

while(n--){m+=r&1;r>>=1;}

if(z)m=31-m;z=m;z<<=5;z+=m;z<<=6;z+=m;

\_\_builtin\_sthio((short\*)((x|16<<0x16|y<<9|0)\*2),z);

} // imager

//-------------------------------------------

//-------------------------------------------

// plotbar May 2020 RTervo

// plot a one pixel wide vertical bar in color

// (x0,y0) = top of bar location

// h = height of bar

void plotbar(int x0,int y0,int h,int color ){

int x=x0,y;

for(y=y0;y<y0+h;y++) imager(x,y,color);

}

//-------------------------------------------

//-------------------------------------------

// clrscr May 2020 RTervo

// clear the vga screen

void clrscr( void ) {

int x,y;

for(y=0;y<240;y++) {

for(x=0;x<320;x++) {

imager(x,y,1);

} // end for y

} // end for x

} // end clrscr

//-------------------------------------------

//-------------------------------------------

// code25writer May 2020 RTervo

// displays a code25 barcode on the vga screen

// from a supplied numeric string (max 10 digits)

void code25writer(char \*indata) {

// indata is a null terminated string of digits 0..9

// indata == null generates random barcode contents

// flag if barcode is vertically centered or randomly

#define randomy 0

\_\_builtin\_stwio ( (void \*) 0x10000000,0); // clear red LED

\_\_builtin\_stwio ( (void \*) 0x10000010,0); // clear grn LED

printf("\x1B[0m"); // reset Terminal attributes

printf("\x1B[H" ); // home the cursor

printf("\x1B[2J"); // clear the screen

printf("ECE3221 Lab 5 : Code25 Barcodes \n");

int codes[] = // 0 to 9 = 12 bar units each

{0xADA,0xD56,0xB56,0xDAA,0xAD6,0xD6A,0xB6A,0xAB6,0xD5A,0xB5A};

int start = 0x0DA; // 8 bar units right justified in 12

int end = 0xD60; // 8 bar units left justified in 12

clrscr(); // clear the vga display

char \*data = indata; // point to process input string (default)

if( (indata == 0 ) || (indata[1] == 0) ) {

// if null input data then

// create random barcode content from 3 to 10 digits

int clen = (rand() % 13) + 3; // random content length

char cnums[20];

int x; // index

for(x=0;x<clen;x++) { // generate clen digits

int y = (rand() % 10); // random digit 0..9

cnums[x] = y + '0'; // save as ascii

} // for

cnums[x] = 0; // null terminate

data = cnums; // point to process this random string

} // if

char \*p;

int len = 0; // len = number of input characters

for(p=data;\*p;p++) len++;

//-----------------------

// show the barcode contents in hex on the 7-seg display

// this provides the correct decoding in an obscured way

// if the input exceeds 9 digits display only dashes

if( len < 10 ) {

int numeric=0;

for(p=data;\*p;p++) numeric = numeric\*10 + (\*p - '0');

outhex7seg(numeric); // show the barcode contents in hex

} else { fill7seg(0x40); } // dashes

//-----------------------

int h=75; // each bar height in pixels

int d=1; // unit bar width in pixels (default)

if( len < 12) d = 2; // wider bars for shorter data

if( len < 7 ) d = 3; // wider bars for shorter data

if( len < 5 ) d = 4; // wider bars for shorter data

int k,x,y;

//-----------------------

// add start and end bars to the input string

// limit input to 20 digit characters

// while ignoring non-digit characters

char contents[25];

contents[0] = 'S'; // start bars

k=1;

for(p=data;\*p && (k<21);p++) // ignore non-digits

if( (\*p>='0') && (\*p<='9')) contents[k++] = \*p;

contents[k++] = 'E'; // end bars

contents[k] = 0; // null terminate string

int length = k\*12\*d; // total width in bar units

//-----------------------

// (x0,y0) = top lefthand corner of new barcode

int y0 = randomy ? ( rand() % (200-h) ) + 20 : 120 - h/2;

int x0 = 160 - length/2;

x=x0;

y=y0;

int bars;

int color;

for(p=contents;\*p;p++) {

if( \*p == 'S' ) bars = start; // start

else if( \*p == 'E' ) bars = end; // end

else bars = codes[\*p - '0']; // digit

for(k=0x0800; k>0; k>>=1) {

color = ( bars & k ) ? 0 : 1;

int b; // make d parallel lines for wider bars

for(b=0;b<d;b++) plotbar(x++,y,h,color);

} // for k

} // for p

printf("A Code25 barcode image is displayed on the VGA screen\n");

} // code25writer

//-------------------------------------------