ECOAR Lab – Winter 2021

group 101

updated 09.01.2022

Grzegorz Mazur

Institute of Computer Science

Warsaw University of Technology

List of students for project assignment

The “big” projects (RISC-V and x86) are assigned by positions on the list below.

1	K-6265	Alonso Caturla	Miguel
2	303850	Badurek	Dawid
3	K-6269	Benito Serrano	Marta
4	K-6358	Díaz Benavente	Javier
5	303853	Domański	Maciej
6	K-6361	Gökpınar	Mustafa Çığ
7	309416	Karwicki	Hubert
8	303870	Kupniewski	Piotr Jan
9	316998	Li	Jiashu
10	K-6398	Matias	Daniel Alves
11	K-6260	Montero Martínez	Javier
12	K-6408	Paiva	Francisco José Santos Rosa
13	300409	Szczepkowski	Jan Marek
14	317462	Zhu	Jihao

RISC-V project – exercise 3

Write a program in RISC-V assembly using RARS. The program should not rely on uninitialized register values. The program should not contain any obviously inefficient actions. Do NOT use multiply/divide instructions for multiplication division by constants being powers of 2. Avoid the sequences of consecutive branches, esp. conditional branches followed by unconditional ones (unless necessary).
All the text processing programs with file i/o should define getc and putc functions for single-character i/o, providing proper buffering of input and input operations with at least 512-byte buffers.
Graphic programs should display images using MARS graphic display mapped to heap address range.
A proper test data should be prepared, covering some interesting cases of program operation.
Maximum score is 6 points. The project may be shown during two lab sessions, 04.01 and 11.01. Each week of delay started will introduce a penalty of one point.
Projects are assigned according to the position on the group students list, shown above.

1. Write a C language source preprocessor, converting the binary integer constants supported in the new C2x standard (like 0b10100000101) into hexadecimal constants accepted by compilers supporting the older standards. The program should not change the comments and strings.

2. Write a C language source preprocessor, removing the digit separators in numeric constants supported in the new C2x standard (like 123’456, 0x12’34’56’78) to make the source compatible with compilers supporting the older standards. The program should not change the comments and strings.

3. Program evaluating an arithmetic expression entered as single line of text. The expression may contain unsigned decimal numbers and characters +, -, *, /, =. It should be evaluated left to right, with all operators having the same priority. = operator causes the output of the current value of expression.

4. Flood fill. A program reads a 24 bpp bmp image file containing black lines on a white background. The user enters the file name, starting coordinates and fill color. The program fills the white background with a specified color, treating black lines and image edges as as borders, displaying the image during processing using RARS graphic screen.

5. Display a number entered by user on a graphic screen using your own bitmap font (digits only).

6. Write a program capable of partially disassembling itself by reading its ow instructions from text section. The program should display the text representation of at least one instruction used in the program in at least 4 of 6 basic RISC-V instruction formats. The immediate arguments should be displayed as hexadecimal or signed decimal numbers. The unsupported instructions may be displayed in any possible way.

7. Unary constants extension for C language. Assume that C program may contain unary constants, starting with 0u followed by some number of ones, like 0u1, 0u111, 0u11111111 / representing the values 1, 3 and 8 respectively. Write a program converting these constants found in a C source to hexadecimal ones. The program should properly handle text strings and comments. Note that 0u alone, not followed by ones, is a valid notation of an unsigned integer constant.

8. Display a grayscale .BMP image in three shades – black, gray and white using dithering. Each pixel should be represented by one of three output colors, with color representation error propagating to the next pixel to the right/left in odd/even lines. The error from the last/first pixel in a line should propagate to the pixel above it.

9. Display a 24 bpp .BMP image, rotated 90 degrees clockwise.

10. Write an assembly source file preprocessor converting constants from classic assembly style notation (hexadecimal – 123h, octal – 123q) to C-syntax (0x123, 0123).

11. Collect all the identifiers from a valid C program and determine the minimum number of characters required to preserve their uniqueness. Produce the replacement list and display it. Assume that the minimum length of a modified identifier is equal to or greater than the maximum length of a C language keyword.

12. Display an image containing a circle of a given diameter < 1024. The circle should be drawn using Bresenham's algorithm, 8 pixels at a time.

13. Write a set of routines for console input and output (in decimal form), addition and multiplication of 64-bit unsigned numbers.

14. Write a program converting any file into a C language file representing the content of the original file as an array of bytes. The C program file should be properly formatted (16 bytes per line, offsets in comments before each 16 lines, total size in final comment.

x86 projects

Write a program containing two source files: main program written in C and assembly module callable from C. The C declaration of an assembly routine is given for each project task. Use NASM assembler (nasm.sf.net) to assemble the assembly module. Use C compiler driver to compile C module and link it with the output of assembler. The C program should use command line arguments to supply the parameters to an assembly routine and perform all I/O operations. No system functions nor C library functions should be called from assembly code. Arguments for bit manipulation routines should be entered in hexadecimal. Routines processing .BMP files may receive as arguments either the pointer to the whole .BMP file image in memory or the pointer to bitmap and its sizes read by main program. The routines should correctly process images of any sizes unless stated otherwise. Either C or assembly code should compute the stride parameter - size of horizontal line rounded up to the boundary of 4 bytes.
Argument list shown for assembly routines may be modified or expanded if needed. Image width and height may be passed from C program to an assembly routine as arguments.
The program should be implemented in two versions: 32-bit and 64-bit, conforming to the respective Unix calling convention. Both conventions are also described in a document available from www.agner.org. While converting from x86 to x86-64, try to remove memory variables if they were used in x86 version and place the variables in extra processor's registers.
Maximum score for (any) single version is 6 points. The second version is worth 2 points.
The program should not contain any obviously inefficient actions. Do NOT use multiply/divide instructions for multiplication division by constants being powers of 2. Avoid the sequences of consecutive branches, esp. conditional branches followed by unconditional ones (unless necessary).
Any attempt to submit the project explicitly violating the calling convention will be punished by subtracting one point from the final score.
Projects are assigned based on the position on the list.
The project are to be shown during the lab sessions on 25.01 or 01.02

1. `void horthin(void *img, int width, int height)`
   Implement a horizontal “thinning” filter for a 1 bpp .BMP image The filter should remove the first and last pixel from each horizontal run of black pixels, leaving at least one black pixel in a horizontal run. Handle any image width properly, not only multiples of 8. Test the program using images of various widths.

2. `void filter(void *img, int width, int height, unsigned char *mtx;`
   Filter a 24 bpp .BMP using 3×3 matrix filter. Coefficients of a filter are given as 8-bit fixed point unsigned numbers in 0.8 format, representing fractions from 0/256 to 255/256, which are entered as integers in range 0..255. The matrix may be hard coded in C program. Simple blurring filter may be implemented using matrix like {10, 15, 10, 15, 156, 15, 10, 15, 10}.

3. `int mandel(int re, int im);`
   Compute a Mandelbrot set image using fixed-point number representation (16.16 or 8.24 format). The assembly routine should return the number of iterations needed to check if the point specified by its coordinates belongs to Mandelbrot set. C program should call the routine in a loop executed once for each pixel and produce a .BMP file.

4. `void uquantize(void *img, int levels);`
   Quantize uniformly each primary of a 24 bpp .BMP image to the specified number of levels. Example: with levels = 4, values 0..63, 64..127, 128..191 and 192..255 should quantize to 4 intensities: 32, 96, 160 and 224.

5. `int cachesize(unsigned int level);`
   Return the size of processor’s cache reported by CPUID instruction based on arg. value: 0 - L1 code, 1 – L1 data, 2 – L2, 3 – L3.

6. `void aaline8(void *img, int xs, int ys, int xe, int ye);`
   Draw an antialiased 1 pixel wide white line in an 8 bpp grayscale .BMP image. Assume the image is initially black. Start and end coordinates are expressed as fixed point format 16.16 (16 bits for integer and fractional parts).

7. `void plotxn(void *image, int width, int height, unsigned int n, unsigned int color);`
   Draw a plot of $y=x^n$ function in a 24 bpp .BMP bitmap using the specified color value. Assume that the image represents the square -1 <= x <= 1, -1 <= y <= 1. Calculations should be performed using fixed-point numbers.

8. `void emphasize8(void *img, int width, int height);`
   Enhance the dynamic range of 8 bpp grayscale image so that the darkest shade becomes black and the lightest – white. Change the intermediate shades proportionally.

9. `int product(unsigned int x, unsigned int *factors, unsigned int nfactors);`
   Check if x may be expressed as a product of values contained in factors. nfactors argument is used to pass the number of possible factors. Return the powers of respective factors in place of factors in the same vector. Assume that input vector is sorted in descending order.

10. `void reduce_contrast(void *img, unsigned int rfactor);`
    Reduce the contrast of a 24 bpp .BMP image by scaling primary colors from the range 0..255 to 128-rfactor..128+rfactor, where rfactor = 1..127.

11. `void aspiral(void *image, int width, int height, int xc, int yc, unsigned int pxperrot)`
    Draw an Archimedean spiral in a 24 bpp .BMP bitmap using the specified color/gray value. The C program should read an image from the specified file, get the spiral parameters from the command line, invoke the routine and save its results into a file. Use some reasonable algorithm for drawing the spiral (all points at a given angle in a single pass).

12. `void shrinkbmp1(void *img, unsigned int scale);`
    Scale down a 1 bpp .BMP image by an integer factor. Select target image pixel color based on majority voting.

13. `void shrinkbmp24(void *img, unsigned int scale_num, unsigned int scale_den);`
    Scale down a 24 bpp .BMP image by fractional factor given as scale_num/scale_den. Assume scale < 1.

14. `int iround(double d);`
    Round the floating point number to the nearest 32-bit integer without using the floating point unit. INT_MIN/INT_MAX values should be returned for negative/positive input arguments out of range .