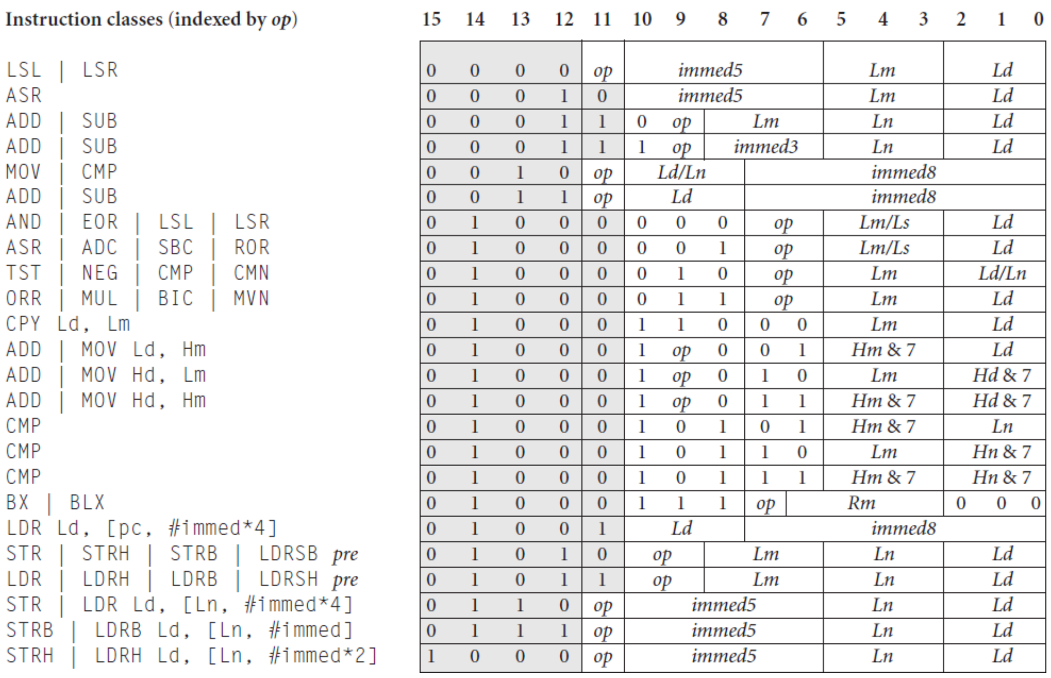
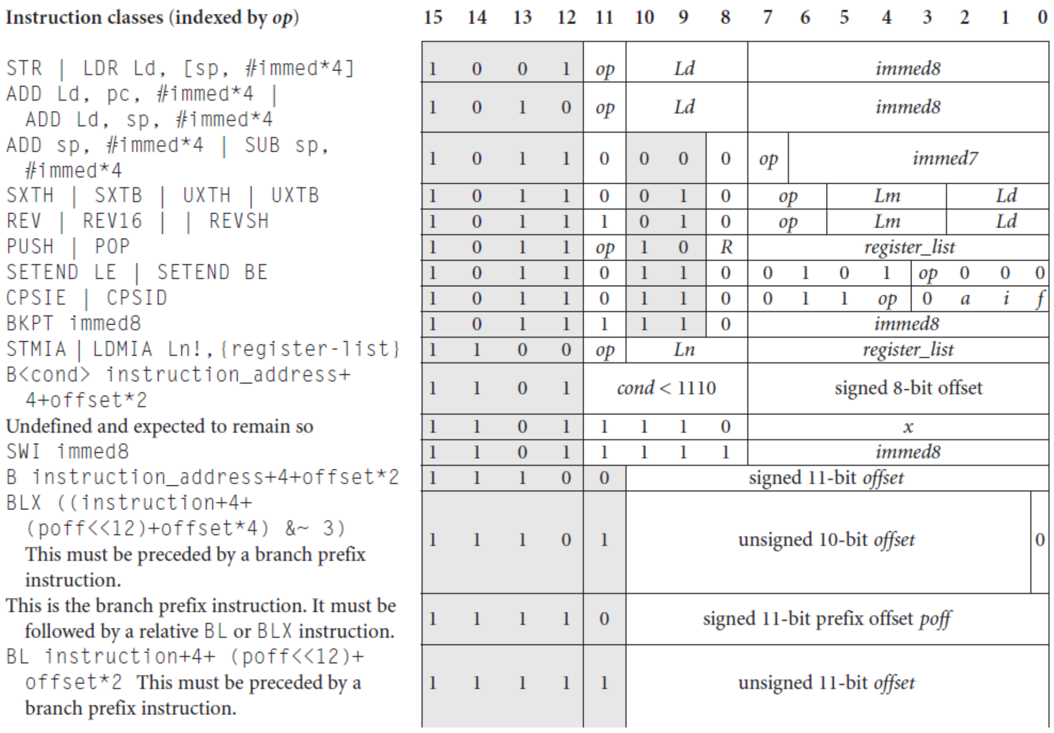
Homework 3 – Pipelined THUMB CPU for Color Conversion and Edge Detection

Due Dec. 8 , 2015

The course ftp side has RTL codes of a pipelined microprocessor that can execute 16-bit THUMB instructions. The following lists the THUMB instruction encoding.



Instruction classes (indexed by *op*)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LSL | LSR	0	0	0	0	*op*	*immed5*					*Lm*			*Ld*		
ASR	0	0	0	1	0	*immed5*					*Lm*			*Ld*		
ADD | SUB	0	0	0	1	1	0	*op*	*Lm*			*Ln*			*Ld*		
ADD | SUB	0	0	0	1	1	1	*op*	*immed3*			*Ln*			*Ld*		
MOV | CMP	0	0	1	0	*op*	*Ld/Ln*			*immed8*							
ADD | SUB	0	0	1	1	*op*	*Ld*			*immed8*							
AND | EOR | LSL | LSR	0	1	0	0	0	0	0	0	*op*		*Lm/Ls*			*Ld*		
ASR | ADC | SBC | ROR	0	1	0	0	0	0	0	1	*op*		*Lm/Ls*			*Ld*		
TST | NEG | CMP | CMN	0	1	0	0	0	0	1	0	*op*		*Lm*			*Ld/Ln*		
ORR | MUL | BIC | MVN	0	1	0	0	0	0	1	1	*op*		*Lm*			*Ld*		
CPY Ld, Lm	0	1	0	0	0	1	1	0	0	0	*Lm*			*Ld*		
ADD | MOV Ld, Hm	0	1	0	0	0	1	*op*	0	0	1	*Hm* & 7			*Ld*		
ADD | MOV Hd, Lm	0	1	0	0	0	1	*op*	0	1	0	*Lm*			*Hd* & 7		
ADD | MOV Hd, Hm	0	1	0	0	0	1	*op*	0	1	1	*Hm* & 7			*Hd* & 7		
CMP	0	1	0	0	0	1	0	1	0	1	*Hm* & 7			*Ln*		
CMP	0	1	0	0	0	1	0	1	1	0	*Lm*			*Hn* & 7		
CMP	0	1	0	0	0	1	0	1	1	1	*Hm* & 7			*Hn* & 7		
BX | BLX	0	1	0	0	0	1	1	1	*op*	*Rm*				0	0	0
LDR Ld, [pc, #immed*4]	0	1	0	0	1	*Ld*			*immed8*							
STR | STRH | STRB | LDRSB *pre*	0	1	0	1	0	*op*		*Lm*			*Ln*			*Ld*		
LDR | LDRH | LDRB | LDRSH *pre*	0	1	0	1	1	*op*		*Lm*			*Ln*			*Ld*		
STR | LDR Ld, [Ln, #immed*4]	0	1	1	0	*op*	*immed5*					*Ln*			*Ld*		
STRB | LDRB Ld, [Ln, #immed]	0	1	1	1	*op*	*immed5*					*Ln*			*Ld*		
STRH | LDRH Ld, [Ln, #immed*2]	1	0	0	0	*op*	*immed5*					*Ln*			*Ld*		



Instruction classes (indexed by *op*)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STR | LDR Ld, [sp, #immed*4]	1	0	0	1	*op*	*Ld*			*immed8*							
ADD Ld, pc, #immed*4 | ADD Ld, sp, #immed*4	1	0	1	0	*op*	*Ld*			*immed8*							
ADD sp, #immed*4 | SUB sp, #immed*4	1	0	1	1	0	0	0	0	*op*	*immed7*						
SXTH | SXTB | UXTH | UXTB	1	0	1	1	0	0	1	0	*op*		*Lm*			*Ld*		
REV | REV16 | | REVSH	1	0	1	1	1	0	1	0	*op*		*Lm*			*Ld*		
PUSH | POP	1	0	1	1	*op*	1	0	*R*	*register_list*							
SETEND LE | SETEND BE	1	0	1	1	0	1	1	0	0	1	0	1	*op*	0	0	0
CPSIE | CPSID	1	0	1	1	0	1	1	0	0	1	1	*op*	0	*a*	*i*	*f*
BKPT immed8	1	0	1	1	1	1	1	0	*immed8*							
STMIA | LDMIA Ln!,{register-list}	1	1	0	0	*op*	*Ln*			*register_list*							
B<cond> instruction_address+4+offset*2	1	1	0	1	*cond* < 1110				signed 8-bit offset							
Undefined and expected to remain so	1	1	0	1	1	1	1	0	*x*							
SWI immed8	1	1	0	1	1	1	1	1	*immed8*							
B instruction_address+4+offset*2	1	1	1	0	0	signed 11-bit *offset*										
BLX ((instruction+4+(poff<<12)+offset*4) &~ 3) This must be preceded by a branch prefix instruction.	1	1	1	0	1	unsigned 10-bit *offset*										0
This is the branch prefix instruction. It must be followed by a relative BL or BLX instruction.	1	1	1	1	0	signed 11-bit prefix offset *poff*										
BL instruction+4+(poff<<12)+offset*2 This must be preceded by a branch prefix instruction.	1	1	1	1	1	unsigned 11-bit *offset*										

1. Use THUMB instructions to write an assembly code that performs the color conversion from RGB to YC_bC_r (YUV) using the following equation:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

   for all the pixels in an image file.
2. The above computation requires nine multiplications and additions. A low-cost implementation is to approximate the constants in the above conversion matrix by sum of power of two. For example, $0.299 \approx 2^{-2} + 2^{-5} + 2^{-6} + 2^{-8}$. Thus the multiplication of R by 0.299 can be replaced by shift-and-adds. Repeat the conversion using the pure shift-and-add operations. Write the corresponding THUMB assembly instructions that process the color conversion for an image file.
3. After generating the intensity value (Y) for each pixel, use the following Sorbel convolution masks to find the vertical edges and horizontal edges from the intensity image:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

   The edge detection algorithm is as follow:
   (1) First, center one convolution mask over successive pixels in the original intensity image.
   (2) Multiply the coefficient in each mask by the intensity value of the underlying pixel and sum the 9 products together.
   (3) Select a threshold value by yourself to determine whether the pixel belongs to an edge by comparing the value calculated in (2) with the threshold value.
   (4) Repeat the above operations for the other convolution mask.
   (5) Note that you might repeat the intensity values of the boundary pixels and create extra pixels that surround the original image for Sorbel edge calculation of the boundary pixels.
4. Trace the given Verilog RTL codes of the 16-bit pipelined THUMB processor given, and use the given test bench tb_thumb.v to verify the RTL code.
5. Modify the testbench to verify the correctness of your assembly instructions. Hint: Find an image file of proper resolution with each pixel represented by RGB and make conversion to generate the corresponding image file with each pixel represented by YC_bC_r. Print out the image with only the illumination (Y) component only (C_bC_r are chrominance components).
6. Then, based on the generated intensity image Y, find the binary image of the vertical edges image that contain only two values (255 for edges and 0 for non-edges). Similarly, find the find the binary image containing only the horizontal edges.
7. In the assembly programs of color conversion, only a few THUMB instructions are used. Make modifications for the Verilog RTL codes by only keeping the part of the hardware that is needed for executing the instructions required in the above color conversion assembly programs and edge detection programs. Re-run the test bench to make sure that your modified RTL code functions correctly.

8. Compare the area cost of the synthesized THUMB microprocessor before and after your modification.
9. Synthesize the processor before and after modification using Synopsys Design Compiler. Report the timing and area cost. Compare the area cost of the simplified THUMB processor with the original one.
10. Perform the post-synthesis verification again to make sure that the synthesized gate-level netlist functions correctly.
11. Repeat the above steps but using the Xilinx FPGA synthesis tool.
12. (Bonus) The Sorbel difference operator generates the edges in 1-D direction, i.e., along the x-direction or y-direction. To detect edges in 2-D space, we can use Laplacian of Gaussian (LOG) or Difference of Gaussian (DOG). The following is an example of 3x3 DOG mask:

$$DOG = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

    Redo the previous problems for the DOG operator.

Report Requirements

Describe the difficulty you encounter and how you solve the problems. Email your report to TA with title: HDL HW3 [your ID] [Your name]. Your attachment should include all the necessary codes and the homework report.

References:

1. S. Lee, *Advanced Digital Logic Design Using Verilg, State Machines, and Synthesis for FPGAs*, Nelson, 2006. (Chap. 9: Verilog code of the pipelined 16-bit THUMB CPU; Appendix A: THUMB instructions)
2. S. Lee, *Advanced Digital Logic Design Using VHDL, State Machines, and Synthesis for FPGAs*, Nelson, 2006. Chap. 9: VHDL code of the pipelined 16-bit THUMB CPU; Appendix A: THUMB instructions)
3. D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, The Hardware/Software Interface, ARM edition*, Appendices B1, B2, 4th ed., Elsevier, 2010.
4. S. Fuevwe, *ARM, Systen-on-Chip Architecture,* 2nd ed., Chap. 7: The Thumb Instruction Set, Pearson Edu. Limited, 2000.
5. P. J. Ashenden, Digital Design, and Embedded Systems Approach Using Verilog, Morgan Kaufmann Pub., 2008. (Chap. 9: Edge detection)