The Virtual CPU Manual

Names: Matthew Angelakos and Elliot Niemann

Pledge: I pledge my honor that I have abided by the Stevens Honor System.

Question: What did each of us do?

Answer: To make this simple

- General Outline: Both
- Encoding System: Both
- CPU: Matthew
- Assembler: Elliot

Assembler Information:

- Use the program in the same folder as the text file named “commands.txt” that contains the instructions.
- Runs through all needed commands when run
- Reads through the text file and outputs image file “CPURAMmem”
- The image file is then loaded into the RAM in the CPU

CPU Information:

- 7 General Registers, 1 Link Register(Can just be used as a General Register as well). From X0-X6(General), X7(Link), Everything 16bit
- 16 Bit instructions(You can make 65k instructions if you wanted!)
- Each register can be referenced like in a regular Armx86 assembly with the X in front followed by the number, i.e., X2.
- Pipelined, no control or data hazard.
- What instructions can we give? (imm) = immediate useable as well, (Z)=can set Zero flag, All do the same as in standard ARM
    - LDR: Loads data from the memory at address Xn offset by an imm
    - STR: Stores data in the memory at address Xt offset by an imm

- MOV(imm) moves data into a register
- ADD(imm)(Z) adds data together and puts it into a register
- SUB(imm)(Z) subtracts data together and puts it into a register
- MUL(imm)(Z) multiplies data together and puts it into a register
- DIV(imm)(Z) divides data together and puts it into a register
- AND(imm)(Z) takes the logical and of two data and puts it into a register
- ORR(imm)(Z) takes the logical or of two data and puts it into a register
- B branches to an instruction indicated by the label
- CBZ branches to an instruction indicated by the label if = 0
- BL branches to an instruction indicated by the label and puts the current address into the Link Register
- RET returns the PC to whatever is in the Link Register

Encoding:

	action	PBr,	CBr,	UBr,	ALUop,	ALUsrc,	MemWrite,	MemRead,	RegWrite,	LinkReg,	MemToReg,	Reg2Loc,				
AND	0000	0	0	0	0	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
AND imm	0000	0	0	0	0	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
ANDS	1000	0	0	0	1	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
ANDS imm	1000	0	0	0	1	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
ORR	0001	0	0	0	0	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
ORR imm	0001	0	0	0	0	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
ORRS	1001	0	0	0	1	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
ORRS imm	1001	0	0	0	1	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
ADD	0010	0	0	0	0	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
ADD imm	0010	0	0	0	0	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
ADDS	1010	0	0	0	1	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
ADDS imm	1010	0	0	0	1	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
SUB	0011	0	0	0	0	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
SUB imm	0011	0	0	0	0	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
SUBS	1011	0	0	0	1	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
SUBS imm	1011	0	0	0	1	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
MUL	0100	0	0	0	0	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
MUL imm	0100	0	0	0	0	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
DIV	0101	0	0	0	0	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)	Rd(5 bits)
DIV imm	0101	0	0	0	0	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)	Rd(5 bits)
LDR	0010	0	0	0	0	1	0	1	1	0	1	0	imm(11 bits)		Rn(5 bits)	Rt(5 bits)
STR	0010	0	0	0	0	1	1	0	0	0	0	0	imm(11 bits)		Rn(5 bits)	Rt(5 bits)
B	0111	0	0	1	0	0	0	0	0	1	0	0	imm(11 bits PC-Relative offset)			00000 00000
BL	0111	0	0	1	1	0	0	0	1	0	0	0	imm(11 bits PC-Relative offset)			00000 00111
CBZ	0111	0	1	0	1	0	0	0	0	0	0	0	imm(11 bits PC-Relative offset)			00000 Rt(5 bits)
RET	0111	1	0	0	1	0	0	0	0	0	0	0	imm(11 bits PC-Relative offset)			00000 Rt(5 bits)
B.EQ	0111	0	1	0	1	0	0	0	0	0	0	0	imm(11 bits PC-Relative offset)			00000 00000
MOV	0111	0	0	0	0	0	0	0	1	0	0	1	Rm(5 bits)	111000	Rn(5 bits)=Rd(5 bits)	
MOV imm	0111	0	0	0	0	1	0	0	1	0	0	1	imm(11 bits)		Rn(5 bits)=Rd(5 bits)	

Our system uses a 36-bit encoding scheme, based on what we went over in class, and an extra 4 for what to do inside the ALU. So that is why every field's size is based on the textbook. Note this could've been shortened, but we wanted not to have an issue adding more instructions or registers, and we knew what was used in the textbook could implement everything I'd like to add; MemRead also could've been removed, but we didn't think we'd make it line address and not byte address. If it's hard to read above, we also attached the text file in our submission under codes.txt.

Instructions Format:

AND Rd,Rn,Rm	/	AND Rd,Rn,imm11
ANDS Rd,Rn,Rm	/	ANDS Rd,Rn,imm11
ORR Rd,Rn,Rm	/	ORR Rd,Rn,imm11
ORRS Rd,Rn,Rm	/	ORRS Rd,Rn,imm11
ADD Rd,Rn,Rm	/	ADD Rd,Rn,imm11
ADDS Rd,Rn,Rm	/	ADDS Rd,Rn,imm11
SUB Rd,Rn,Rm	/	SUB Rd,Rn,imm11
SUBS Rd,Rn,Rm	/	SUBS Rd,Rn,imm11
MUL Rd,Rn,Rm	/	MUL Rd,Rn,imm11
DIV Rd,Rn,Rm	/	DIV Rd,Rn,imm11
LDR Rt,[Rn]	/	LDR Rt,[Rn,imm11]
STR Rt,[Rn]	/	STR Rt,[Rn,imm11]
B label		
BL label		
CBZ Rt,label		
RET		
B.EQ label		
MOV Rd, Rm	/	MOV Rd,imm11

When the labeling line to be branched to the format is “(label): ” before the command