Cherry Instruction Set

Designed this page to look like
https://www.dsi.unive.it/~gasparetto/materials/MIPS_Instruction_Set.pdf

18 bit instructions. 5 bits to decide which instruction to use. Remaining bits for parameters. If we need more instruction space, start collapsing unop, binop, and reduceop instructions.

Arithmetic Instructions

Instruction	Example	Meaning	Comments
Matrix Multiply	`matmul`	REG[ACC] += REG[INPUT] @ REG[WEIGHTS]	
Multiply Accumulate	`mulacc`	REG[ACC] += REG[INPUT] * REG[WEIGHTS]	Element wise multiply
Vector Add	`add`	REG[OUTPUT] = REG[INPUT] + REG[WEIGHTS]	
Vector Subtract	`sub`	REG[OUTPUT] = REG[INPUT] - REG[WEIGHTS]	
Vector Multiply	`mul`	REG[OUTPUT] = REG[INPUT] * REG[WEIGHTS]	
Vector Divide	`div`	REG[OUTPUT] = REG[INPUT] / REG[WEIGHTS]	
Vector Power	`pow`	REG[OUTPUT] = REG[INPUT] ** REG[WEIGHTS]	
Vector Reduce Max	`max $1`	max([REG[ACC] if $1 else none, (REG[OUT]])	Gets the max value in out reg or out reg and acc reg
Vector Reduce Sum	`sum $1`	sum([REG[ACC] if $1 else none, (REG[OUT]])	Gets the sum of out reg or out reg + acc
Vector ReLU	`relu`	max(x,0) of every x in REG[INPUT]	
Vector Exp	`exp`	e^x of every x in REG[INPUT]	
Vector Log	`log`	log_2 of every x in REG[INPUT]	
Vector GT0	`gtz`	1 if x > 0 else 0 of every x in REG[INPUT]	

Data Transfer Instructions

Instruction	Example	Meaning	Comments
Copy	`copy $1, $2`	REG[$1] = REG[$2]	Moves/copies from $2 to $1

Zero	`zero $1`	REG[$1] = 0	Zeroes out a register
Load	`load $1, $2, $3, $4, $5, $6`	REG[$2] = ... $3 and $4 are lengths. $5 is zero flag, $6 is skip flag. APU $1	Uses address from an APU
Store	`store $1, $2, $3, $4`	MEM[addr] = REG[$2][:$3][:$4] and APU $1	Use address from an APU

Control Flow Instructions

Instruction	Example	Meaning	Comments
start independent loop	`start_independent_loop ($1)`	start loop with ro_data[$1] params	From ro_data registers gets loop iteration count and number of instructions in the loop body. Loop iterations can be reordered freely. must be no loops nested inside this loop
start loop	`start_loop ($1)`	start loop with ro_data[$1] params	This loop must be executed in order. We can't rearrange when different loop bodies run. This is low performance loop
end loop	`end_loop_or_jump_if_needed ($1)`	Gets jump number from ro_data[$1]	jumps if loop has more more iterations

Registers

Not all instructions can access all registers.

Name	Width (bits)	Read Only	Description
`$d0 - $d7`	36	X	ro_data. Read only data segments compiled into the kernel. Can be loop count and number of instructions in the loop body.
`$d32 - $d63`	54	X	ro_data. Read only data segments compiled into the kernel. Initial addresses, stride_x's, and stride_y's for the APUs to use
`$a0 - $a31`	54		32 addresses, stride_x's, and stride_y's calculated using 32 APUs
`$r0 - $r3`	288		Each one holds a 4x4 matrix. They have names, INPUT, WEIGHTS, OUTPUT, ACC. When this document uses the word REG. i.e. REG[WEIGHTS] == $r1

Thoughts

The best part about this all is that an entire kernel might have 10 instructions and thus take 180 bits. We can fit 100 kernels in a single BRAM18. We don't even have 100 kernels written in

software yet lol. This isn’t completely true, since a kernel has to be recompiled for different input tensor shapes. i.e. you might have 100 matmul kernels on chip.
Since instruction size is 18 bits, maybe we can squeeze it down to 16 bits and feed kernels over off chip ddr3 which is cheap and will fit orders of magnitude more kernels than we can dream of. Only need 18 gigabits/s of bandwidth to read instructions. 4GB DDR3 is $20. SSD is nice since it can cache kernels long term.