Instruction Set Design Document

Wei Hong and Tengyu Sun
{weihong, tsun}@cs.umass.edu

In this design document, we describe the instruction set for the CS535 project.

The data types supported in the ISA are 8-bit byte, 32-bit word integer and 32-bit single precision floating point. The byte ordering is big-endian.

There are 16 32-bit general purpose registers for integers ($g0-$g31), 16 32-bit registers for floating point ($f0-$f31), 16 64-bit vector registers for SIMD ($v0-$v31) and one special 32-bit register for program counter ($pc).

$g0 is fixed to 0. $g1-$g15 and $f0-$f15 can be accessed by load/store instructions. $g15 is used by flow control instructions as the default register.

$v0-$v15 can hold 8 8-bit byte integers. The vector element index starts from the least-significant bit.

$pc cannot be directed accessed and can only be changed by special instructions.

The cache has 2 layers. L0 cache has one 32KB data cache and one 32KB instruction cache. They are direct mapped. L1 cache only has one 512KB cache. It is 4-way associative. The cache line is 64 bytes. The write policy for both layers are write through and write allocate.

The instruction set supports four basic types of instructions, data transfer, arithmetic and logical on integers, flow control and floating point operations. In addition, we also support cache prefetch, atomic operations and SIMD. The details about the supported instructions are listed in Table 2.

The instruction set uses fixed length encoding. The whole instruction is 32-bit long. The addressing mode are immediate and displacement. Register index requires 4 bits and immediate operand can have up to 17 bits. The operation code is 7 bit. First 3 bits are for distinguishing types. Each instruction can have up to three operands.

Like in MIPS, depending on the number of operands there are three types of instructions. Type 1 only has an immediate offset. Type 2 has two register operands and one immediate and Type 3 has three register operands. The instruction format is described in Table 1.

Table 1. Instruction Format				
Type 1	op offset			
opcode (7)	offset (25)			
Type 2	op $1,$2,immediate			
opcode (7)	$1 (4)	$2 (4)	immediate (17)	
Type 3	arithmetic and logical, sync, syscall,break			
opcode (7)	$1 (4)	$2 (4)	$3 (4)	

Table 2. Instruction List			
Type	**Instruction**	**Format**	**Description**
Data Transfer op=100XXXX	LB	op $1, $2, offset $1, $2 are integer/floating point registers	load a byte as a signed value
	LBU		load a byte as an unsigned value
	SB		store the least significant 8-bit byte
	LW		load a word
	SW		store word
	LSP		load a word as single precision
	WITF	op $1, $2	convert word integer to single precision floating point
	WFTI		convert single precision floating point to word integer
Arithmetic & Logical op=00XXXXX	ADD	op $1,$2,$3 $1, $2, $3 are integer registers	add words
	SUB		subtract words
	ADDI	op $1,$2,im $1, $2 are integer registers	add immediate to word
	SUBI		subtract immediate to word
	MUL	op $1,$2,$3 $1, $2, $3 are integer registers	multiply words store lower word part
	MUH		multiply words store higher word part
	MULU		multiply unsigned words store lower word part
	MUHU		multiply unsigned words store higher word part
	DIV		divide signed word
	DIVU		divide unsigned word
	MODU		modulus unsigned word
	AND	op $1,$2,$3 $1, $2, $3 are	bitwise and
	OR		bitwise or

Table 2. Instruction List			
Type	**Instruction**	**Format**	**Description**
	NOT	$1, $2, $3 are integer registers	bitwise not (flip)
	XOR		bitwise exclusive or
	RR	op $1,$2,$3 $1, $2, $3 are integer registers	rotate word right
	SRL		word logical shift right
	SRA		word arithmetic shift right
	SL		word shift left
	SLT	op $1,$2,$3 $1, $2, $3 are integer registers	compare registers as signed value
	SLTU		compare registers as unsigned value
	SLTI	op $1,$2,im $1, $2 are integer registers	compare registers as signed value
	SLTIU		compare registers as unsigned value
Control op=101XXXX	J	op offset	jump
	JAL		jump and place return address in $g31
	BEQ	op $1,$2	branch if equal
	BNEQ		branch if not equal
	BGEZ	op $1	branch if greater than or equal to 0
	BGTZ		branch if greater than 0
	BLEZ		branch if less than or equal to 0
	BLTZ		branch if less than 0
	BREAK	op	breakpoint
Floating Point op=110XXXX	ADDSP	op $1,$2,$3	single precision add
	SUBSP		single precision subtract
	MULSP		single precision multiply
	DIVSP		single precision divide
	SLTSP	op $1,$2,$3	single precision compare
Cache op=111XXXX	PREF	op hint,$1,offset	cache prefetch
	MOVE	op $1,$2	move between vector registers
	COPYS	op $1,$2,n	copy signed vector element to $g0-$g31
	COPYU		copy unsigned vector element to $g0-$g31

Table 2. Instruction List			
Type	**Instruction**	**Format**	**Description**
SIMD op=01XXXXX	INSERTB	op $1,$2,n	insert value from $g0-$g31 into byte vector
	FILLB	op $1,$2	byte vector replicated from register
	VADDB	op $1,$2,$3	bytes vector add
	VSUBB		bytes vector subtract
	VMULB		bytes vector multiply
	VDIVB		bytes vector divide
	VMODB		bytes vector unsigned modulus remainder
	CEQB	op $1,$2,$3	compare if byte vectors equal
	CLEB		compare if signed byte vectors less than or equal
	CLEUB		compare if unsigned byte vectors less than or equal
	CLTB		compare if signed byte vectors less than
	CLTUB		compare if unsigned byte vectors less than

Software engineering & tools

We are using Github (https://github.com/Tengyu-Sun/MIPS_simulator) for collaboration on coding and version control and Slack (https://www.slack.com) for team communication and file sharing. The simulator with UI will follow the Model View Control architecture. The workload is roughly divided into four parts and each person will work on two separately. There will be unit tests for these parts before integration. The whole system will be implement in C/C++ with Qt.

Timeline

2/7 – 2/26
Implement the basic simulator with Memory, cache and timing. (Wei)
Implement a basic UI for debugging and demonstration purposes. (Tengyu)
2/27 – 3/15
Implement basic instruction set: Data transfer, Arithmetic and logical,(Tengyu)
and Control. Start working on the assembler (Wei)
3/16 – 4/15
ISA advanced cache. Implement benchmark programs in assembly language (Wei)
ISA atomic and SIMD. Finish UI (Tengyu)
4/16 – 4/21
Wei & Tengyu: Final touchup / preparation for final demo