Appendix

A Language Specific Properties Lists

Category	Pre-Defined Name
Keywords	assert, bit, bits, class, code, dag, def, else, false, foreach, defm, defset, defvar field, if, in, include, int, let, list, multiclass, string, then, true
Bang Operators	!add, !and, !cast, !con, !dag, !empty, !eq, !foldl, !foreach, !filter, !ge, !getdagop !gt, !head, !if, !interleave, !isa, !le, !listconcat, !listsplat, !lt, !mul, !ne, !not !or, !setdagop, !shl, !size, !sra, !srl, !strconcat, !sub, !subst, !substr, !tail, !xor
Punctuation	'<', '>', '{', '}', '[', ']', ',', '"', ':', ';', '?', '!', '\', '∼', '=', '-', '—', '&', '*'

B Target Specific Lists

Index	Property Value	Comments for Target-Specific Details
1		Instruction name.
2		Size of encoded instruction.
3		The asm format to print the instruction with.
4		Instruction format classification, e.g., R, I, J.
5		Instruction Set Architecture Name (ISA).
6		The namespace of this target.
7		The name of instruction set architecture, specifial for Mips.
8		The def operand list of this instruction.
9		The use operand list of this instruction.
10		Set to the mapping pattern for this instruction, if we know of one, otherwise, uninitialized.
11		Instruction execution steps used for scheduling.
12		Is this instruciton format field an operand?
13		The name of each instruction format field.
14		The binary value of each instruction format field.
15		The start bit of each instruction format field.
16		The end bit of each instruction format field.
17		The type of each operand, e.g. immediate operand, register operand.
18		The illegal information of each operand, e.g., illegal register operand for specified instruction.Default is null.
19		The precision of each operand.
20		Is the immediate operand a signed operand?
21		Whether this operand is non-zero?
22		The least significant bit of the operand.
23		Field the disassembler can use to provide a way for instructions to not match without killing the whole decode process. Mainly used for ARM.
24		Is it possible for this instruction to read memory?
25		Is it possible for this instruction to write memory?

Index	Property Value	Comments for Target-Specific Details
26		OperandConstraint, e.g. *src* =dst.
27		Is this 3 operand instruction commutable?
28		Enable TableGen to auto-generate a two-operand matcher inst-alias for a three operand instruction. For example, the arm instruction "add r3, r3, r5" can be written as "add r3, r5". The constraint is of the same form as a tied-operand constraint. For example, "*Rn* =Rd".
29		Does this instruction modify any non-operand registers?
30		Does this instruction use any non-operand registers?
31		Is this instruction a branch instruction?
32		Is this instruction a call instruction?
33		Is this instruction a comparison instruction?
34		Is this instruction an indirect branch?
35		Can this be folded as a simple memory operand?
36		Does this instruction have an delay slot?
37		Is this instruction is not expected to be queried for scheduling latencies andtherefore needs no scheduling information?
38		Does the instruction have side effects that are not captured by any operands of the instruction or other flags?
39		Can control flow fall through this instruction?
40		As cheap (or cheaper) than a move instruction?
41		Is this instruction a move register instruction?
42		Is this instruction re-materializable?
43		Is this instruction a return instruction?
44		Is this part of the terminator for a basic block?
45		The "namespace" in which this instruction exists, on targets like ARM which multiple ISA namespaces exist.
46		Describe the instruction decoder method.
47		List of operand names (e.g. "*op*1,op2") that should not be encoded into the output machineinstr.
48		List of predicates which will be turned into isel matching code.
49		Predicates for the encoding scheme in use.
50		Predicates for the instruction group membership such as ISA's.
51		Predicate for the ASE that an instruction belongs to.
52		Predicates for the GPR size.
53		Predicate for marking the instruction as usable in hard-float mode only.
54		Predicates for the FGR size and layout.
55		Predicates for the PTR size.
56		Predicates for anything else.
57		The name of asm match converter used to convert the operandVector into an MCInst.
58		Does this pseudo instruction need special help?
59		Does this instruction have encoding information and can go through to the CodeEmitter unchanged, but duplicates a canonical instruction definition's encoding and should be ignored when constructing the assembler match tables?
60		Does this instruction noeed to be adjusted after isel by target hook?
61		How much added comlexity passed onto matching pattern does this instruction cause?