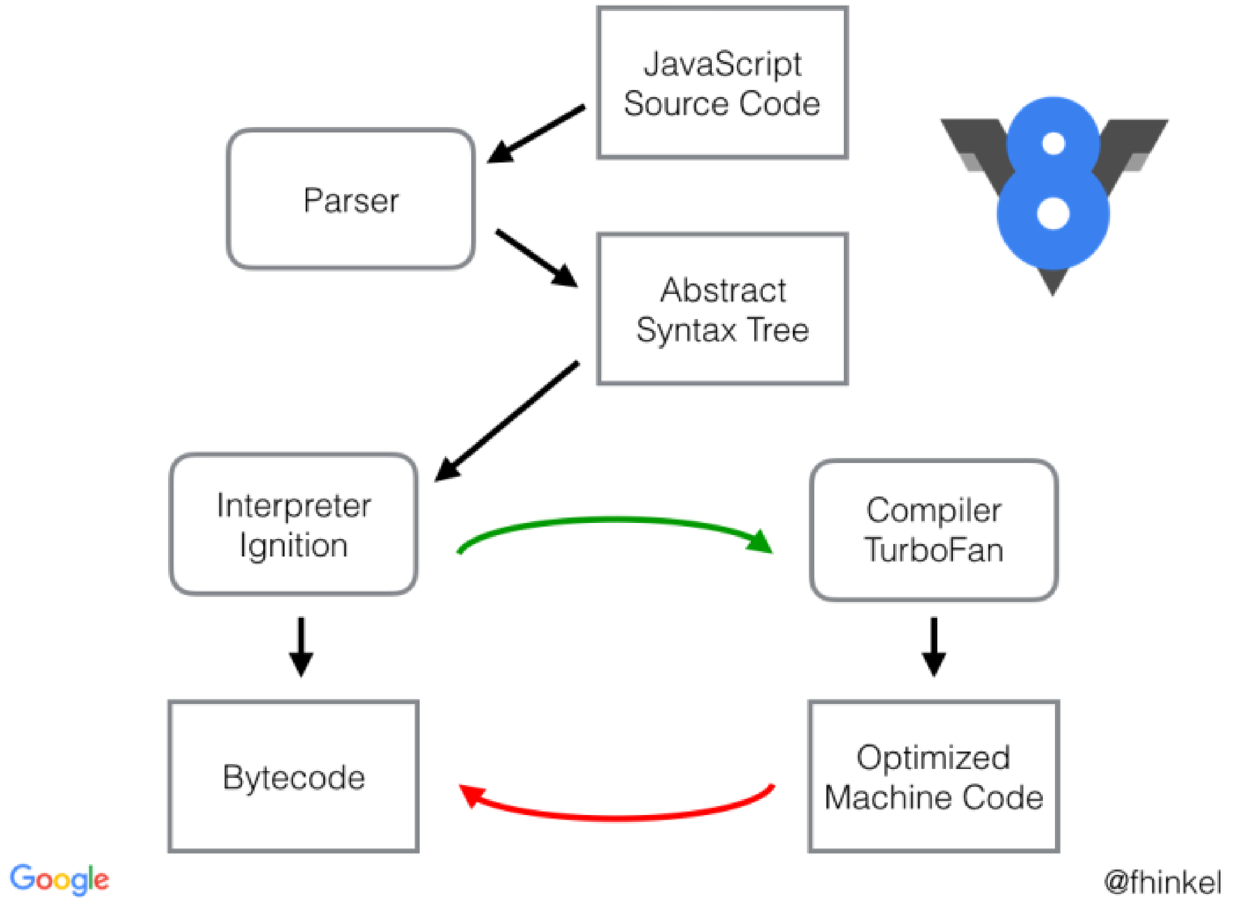
V8 Assembler学习小结

软件所智能软件中心PLCT实验室 实习生 陈家友

2020/04/01

目 录

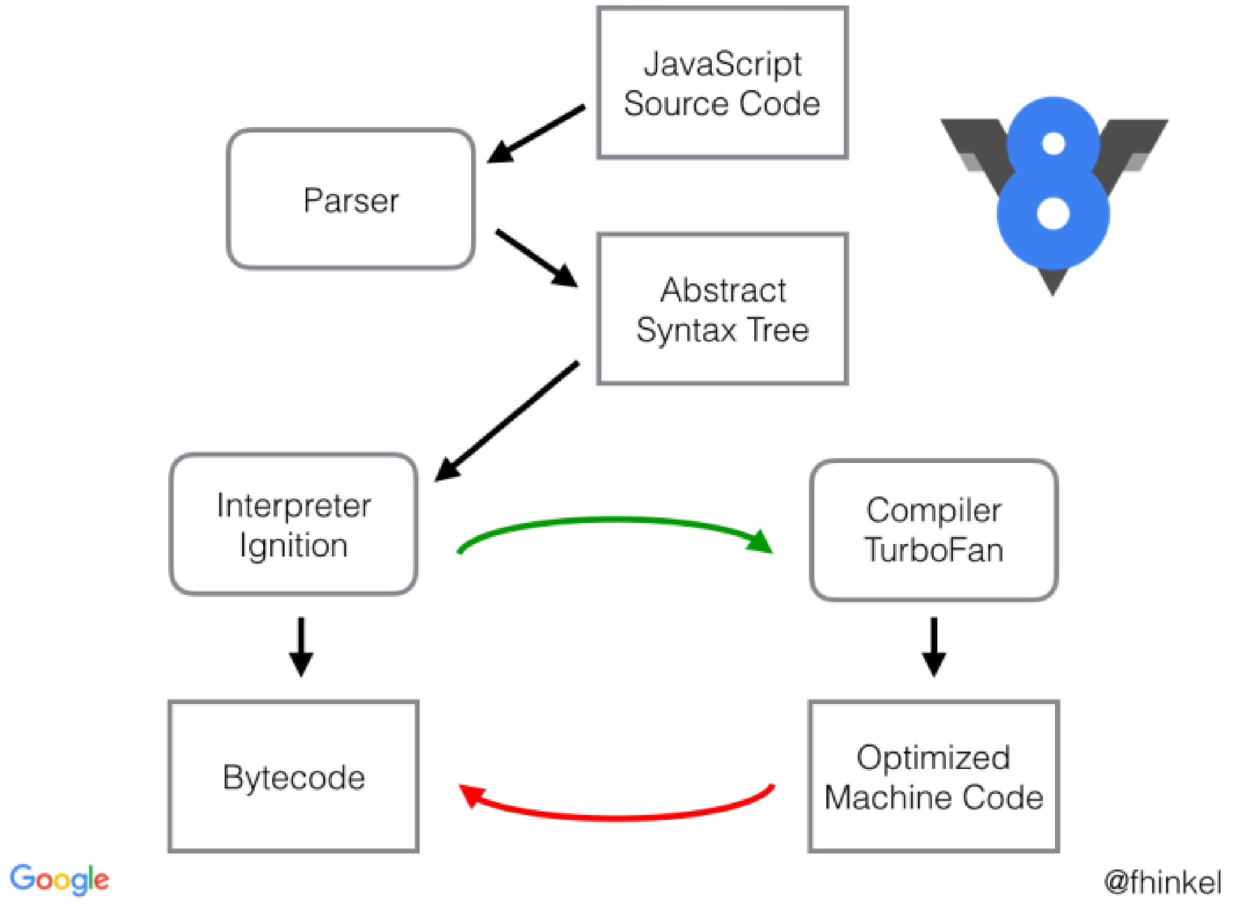
01 V8简介



当 V8 编译 JavaScript 代码时，解析器(parser)将生成一个抽象语法树。语法树是 JavaScript 代码的句法结构的树形表示形式。解释器 Ignition 根据语法树生成字节码。TurboFan 是 V8 的优化编译器，TurboFan 将字节码生成优化的机器代码。

参考资料：
[1] https://zhuanlan.zhihu.com/p/103904567

01 V8简介



　　Ignition 解释器本身由一系列字节码处理程序代码片段组成，每个片段都处理一个特定的字节码，然后调度给下一个字节码的处理程序。这些字节码处理程序使用高层级的、机器架构无关的汇编代码写成，由 CodeStubAssembler 类实现，并由 TurboFan 编译。

　　Turbofan 根据Ignition收集的类型信息识别Hot Function（多次被调用的函数），将Bytecode编译为Optimized Machine Code，以提高代码的执行性能。并当需要进行去优化的时候直接去优化到字节码，而不需要再考虑 JS 源代码。

参考资料：
[1] https://www.lagou.com/lgeduarticle/58193.html
[2] https://www.youtube.com/watch?v=p-iiEDtpy6I&feature=youtu.be

02 Compiler

Compiler pipeline phases

Step A.1. Serialize the data needed for the compilation front-end.

```
  void Serialize();
```

Step A.2. Run the graph creation and initial optimization passes.

```
  bool CreateGraph();
```

Step B. Run the concurrent optimization passes.

```
  bool OptimizeGraph(Linkage* linkage);
```

Step C. Run the code assembly pass.

```
  void AssembleCode(Linkage* linkage,std::unique_ptr<AssemblerBuffer> buffer = {}); // Generate the final machine code.
```

Step D. Run the code finalization pass.

```
  MaybeHandle<Code> FinalizeCode(bool retire_broker = true);
```

Step E. Install any code dependencies.

```
  bool CommitDependencies(Handle<Code> code);
```

03 Assember

Macro-Assembler 与 Assembler

Macro-Assembler

Macro-Assembler中的指令对应IR的操作语义，有些操作数的类型较抽象的，还具有JavaScript的语义。例如：

```
void MacroAssembler::Swap(Register reg1, Register reg2, Register scratch) {
  if (scratch == no_reg) {
    Xor(reg1, reg1, Operand(reg2));
    Xor(reg2, reg2, Operand(reg1));
    Xor(reg1, reg1, Operand(reg2));
  } else {
    mov(scratch, reg1);
    mov(reg1, reg2);
    mov(reg2, scratch);
  }
} //Swap two registers.  If the scratch register is omitted then a slightly less efficient form using xor instead of mov is emitted.

void TurboAssembler::Mov(const Register& rd, Smi smi)
```

03 Assember

Macro-Assembler 与 Assembler

Assembler

Assembler对应指令的函数对应ISA的opcode和operand

Create an assembler. Instructions and relocation information are emitted into a buffer, with the instructions starting from the beginning and the relocation information starting from the end of the buffer.

```
// Arithmetic.
void addu(Register rd, Register rs, Register rt);
void subu(Register rd, Register rs, Register rt);
…
// Logical.
void and_(Register rd, Register rs, Register rt);
void or_(Register rd, Register rs, Register rt);
….
// Shifts.
void srav(Register rt, Register rd, Register rs);
…
// …
```

03 Assember

CodeGenerator::AssembleConstructFrame()

- call_descriptor = linkage()->GetIncomingDescriptor();

//The call descriptor for this compilation unit describes the locations of incoming parameters and the outgoing return value(s).

- StubPrologue(info()->GetOutputStackFrameType()); //Generates function and stub prologue code.

  push(ebp);

  // Caller's frame pointer.将当前的基本指针压入堆栈，以便以后可以恢复。

  mov(ebp, esp);//当前栈帧切换到新栈帧

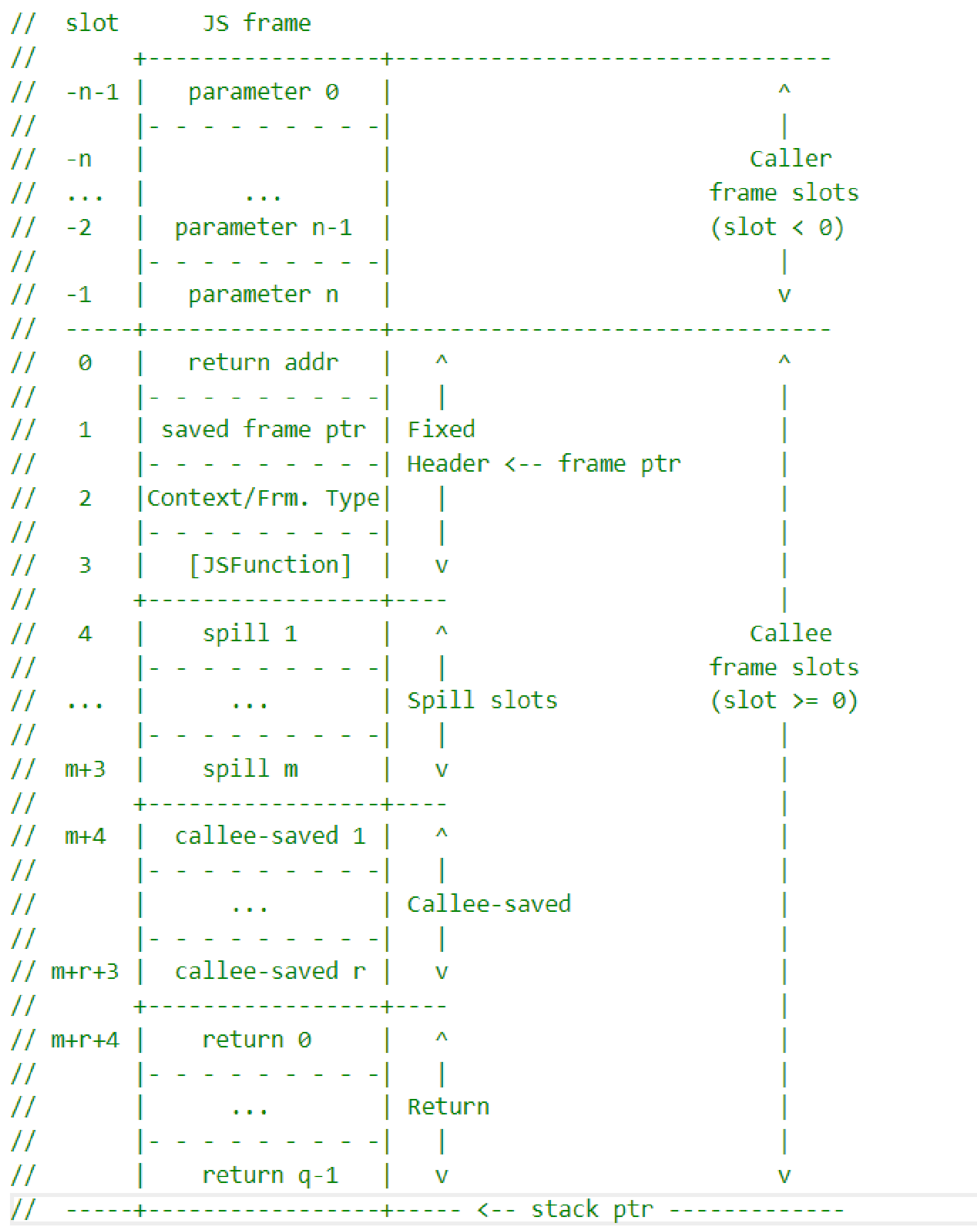
  Subu(sp, sp, Operand(kSystemPointerSize));//给新栈帧分配空间

- AllocateSpillSlot
- Save callee-saved registers.

  MultiPush(saves);

- Create space for returns.

  Subu(sp, sp, Operand(returns * kSystemPointerSize));



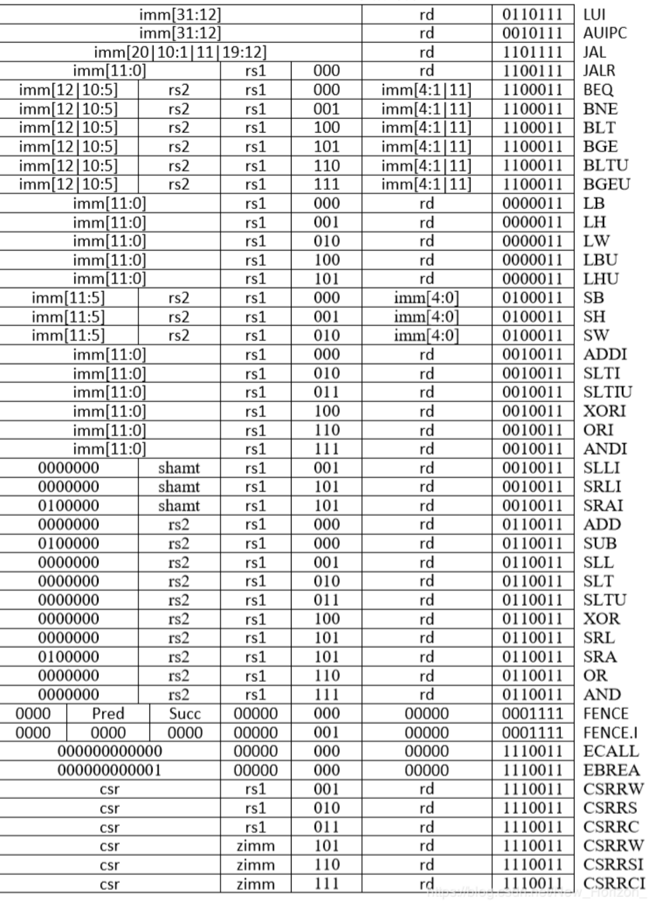
```
//   slot      JS frame
//       +-----------------+--------------------------------
//  -n-1 |   parameter 0   |                            ^
//       |- - - - - - - - -|                            |
//  -n   |                 |                          Caller
//  ...  |       ...       |                       frame slots
//  -2   |  parameter n-1  |                       (slot < 0)
//       |- - - - - - - - -|                            |
//  -1   |   parameter n   |                            v
//  -----+-----------------+--------------------------------
//   0   |   return addr   |   ^                        ^
//       |- - - - - - - - -|   |                        |
//   1   | saved frame ptr | Fixed                      |
//       |- - - - - - - - -| Header <-- frame ptr       |
//   2   |Context/Frm. Type|   |                        |
//       |- - - - - - - - -|   |                        |
//   3   |  [JSFunction]   |   v                        |
//       +-----------------+----                        |
//   4   |    spill 1      |   ^                      Callee
//       |- - - - - - - - -|   |                   frame slots
//  ...  |      ...        | Spill slots           (slot >= 0)
//       |- - - - - - - - -|   |                        |
//  m+3  |    spill m      |   v                        |
//       +-----------------+----                        |
//  m+4  |  callee-saved 1 |   ^                        |
//       |- - - - - - - - -|   |                        |
//       |      ...        | Callee-saved               |
//       |- - - - - - - - -|   |                        |
// m+r+3 |  callee-saved r |   v                        |
//       +-----------------+----                        |
// m+r+4 |    return 0     |   ^                        |
//       |- - - - - - - - -|   |                        |
//       |      ...        | Return                     |
//       |- - - - - - - - -|   |                        |
//       |   return q-1    |   v                        v
//  -----+-----------------+----- <-- stack ptr -------------
```

04 后续工作

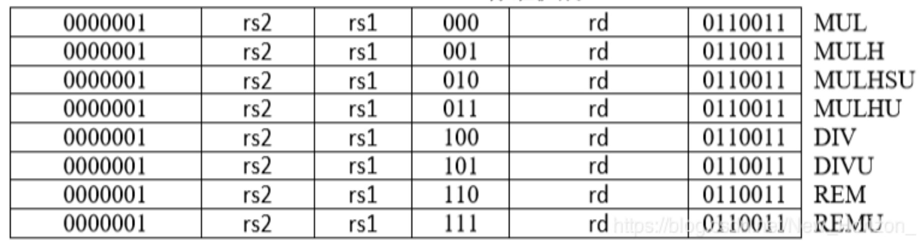
仿照ARM64的Assembler，写一个RISCV32G的Assembler

V8后端先把广义指令选择后的instructions lower成TurboAssembler的instructions (函数)，然后在这些函数中继续lower成 Assembler中的instructions（ 函数），最后产生二进制。

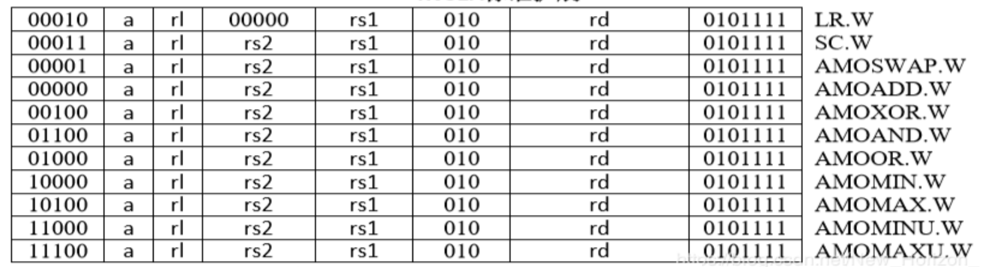
RV32G 是由基本指令子集“I”，及标准扩展指令集整数乘除指令子集“M” +原子操作子集“A” +单精度浮点子集“F “+双精度浮点“D”组成。


RV32I 基本指令集

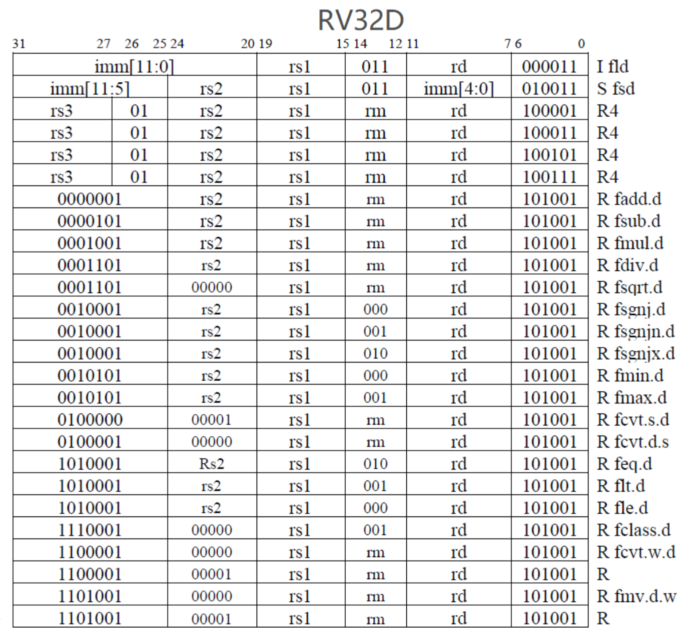
imm[31:12]					rd	0110111	LUI
imm[31:12]					rd	0010111	AUIPC
imm[20|10:1|11|19:12]					rd	1101111	JAL
imm[11:0]			rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2		rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2		rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2		rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2		rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2		rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2		rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]			rs1	000	rd	0000011	LB
imm[11:0]			rs1	001	rd	0000011	LH
imm[11:0]			rs1	010	rd	0000011	LW
imm[11:0]			rs1	100	rd	0000011	LBU
imm[11:0]			rs1	101	rd	0000011	LHU
imm[11:5]	rs2		rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2		rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2		rs1	010	imm[4:0]	0100011	SW
imm[11:0]			rs1	000	rd	0010011	ADDI
imm[11:0]			rs1	010	rd	0010011	SLTI
imm[11:0]			rs1	011	rd	0010011	SLTIU
imm[11:0]			rs1	100	rd	0010011	XORI
imm[11:0]			rs1	110	rd	0010011	ORI
imm[11:0]			rs1	111	rd	0010011	ANDI
0000000	shamt		rs1	001	rd	0010011	SLLI
0000000	shamt		rs1	101	rd	0010011	SRLI
0100000	shamt		rs1	101	rd	0010011	SRAI
0000000	rs2		rs1	000	rd	0110011	ADD
0100000	rs2		rs1	000	rd	0110011	SUB
0000000	rs2		rs1	001	rd	0110011	SLL
0000000	rs2		rs1	010	rd	0110011	SLT
0000000	rs2		rs1	011	rd	0110011	SLTU
0000000	rs2		rs1	100	rd	0110011	XOR
0000000	rs2		rs1	101	rd	0110011	SRL
0100000	rs2		rs1	101	rd	0110011	SRA
0000000	rs2		rs1	110	rd	0110011	OR
0000000	rs2		rs1	111	rd	0110011	AND
0000	Pred	Succ	00000	000	00000	0001111	FENCE
0000	0000	0000	00000	001	00000	0001111	FENCE.I
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREA
csr			rs1	001	rd	1110011	CSRRW
csr			rs1	010	rd	1110011	CSRRS
csr			rs1	011	rd	1110011	CSRRC
csr			zimm	101	rd	1110011	CSRRW
csr			zimm	110	rd	1110011	CSRRSI
csr			zimm	111	rd	1110011	CSRRCI


RV32M标准扩展

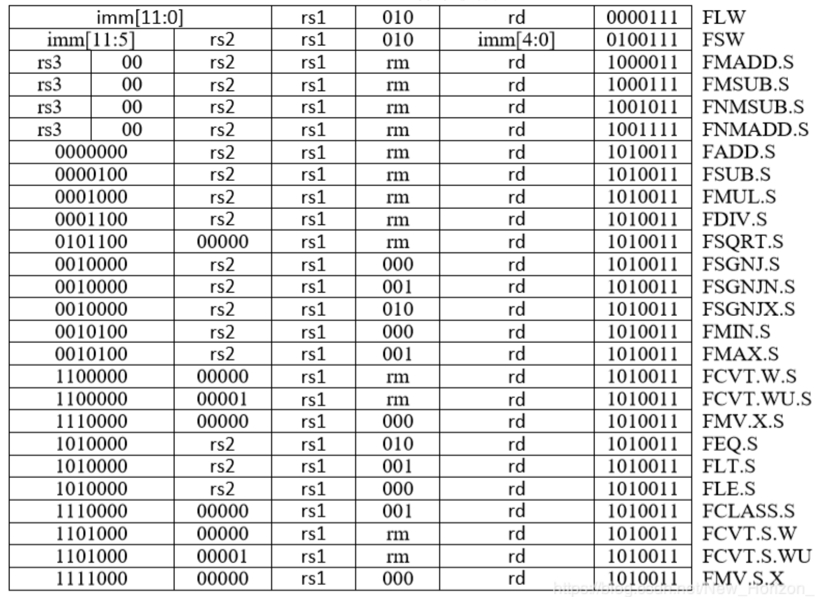
0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU


RV32A标准扩展

00010	a	rl	00000	rs1	010	rd	0101111	LR.W
00011	a	rl	rs2	rs1	010	rd	0101111	SC.W
00001	a	rl	rs2	rs1	010	rd	0101111	AMOSWAP.W
00000	a	rl	rs2	rs1	010	rd	0101111	AMOADD.W
00100	a	rl	rs2	rs1	010	rd	0101111	AMOXOR.W
01100	a	rl	rs2	rs1	010	rd	0101111	AMOAND.W
01000	a	rl	rs2	rs1	010	rd	0101111	AMOOR.W
10000	a	rl	rs2	rs1	010	rd	0101111	AMOMIN.W
10100	a	rl	rs2	rs1	010	rd	0101111	AMOMAX.W
11000	a	rl	rs2	rs1	010	rd	0101111	AMOMINU.W
11100	a	rl	rs2	rs1	010	rd	0101111	AMOMAXU.W


RV32D

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
imm[11:0]			rs1	011	rd	000011	I fld
imm[11:5]		rs2	rs1	011	imm[4:0]	010011	S fsd
rs3	01	rs2	rs1	rm	rd	100001	R4
rs3	01	rs2	rs1	rm	rd	100011	R4
rs3	01	rs2	rs1	rm	rd	100101	R4
rs3	01	rs2	rs1	rm	rd	100111	R4
0000001		rs2	rs1	rm	rd	101001	R fadd.d
0000101		rs2	rs1	rm	rd	101001	R fsub.d
0001001		rs2	rs1	rm	rd	101001	R fmul.d
0001101		rs2	rs1	rm	rd	101001	R fdiv.d
0001101		00000	rs1	rm	rd	101001	R fsqrt.d
0010001		rs2	rs1	000	rd	101001	R fsgnj.d
0010001		rs2	rs1	001	rd	101001	R fsgnjn.d
0010001		rs2	rs1	010	rd	101001	R fsgnjx.d
0010101		rs2	rs1	000	rd	101001	R fmin.d
0010101		rs2	rs1	001	rd	101001	R fmax.d
0100000		00001	rs1	rm	rd	101001	R fcvt.s.d
0100001		00000	rs1	rm	rd	101001	R fcvt.d.s
1010001		Rs2	rs1	010	rd	101001	R feq.d
1010001		rs2	rs1	001	rd	101001	R flt.d
1010001		rs2	rs1	000	rd	101001	R fle.d
1110001		00000	rs1	001	rd	101001	R fclass.d
1100001		00000	rs1	rm	rd	101001	R fcvt.w.d
1100001		00001	rs1	rm	rd	101001	R
1101001		00000	rs1	rm	rd	101001	R fmv.d.w
1101001		00001	rs1	rm	rd	101001	R


RV32F标准扩展

imm[11:0]			rs1	010	rd	0000111	FLW
imm[11:5]		rs2	rs1	010	imm[4:0]	0100111	FSW
rs3	00	rs2	rs1	rm	rd	1000011	FMADD.S
rs3	00	rs2	rs1	rm	rd	1000111	FMSUB.S
rs3	00	rs2	rs1	rm	rd	1001011	FNMSUB.S
rs3	00	rs2	rs1	rm	rd	1001111	FNMADD.S
0000000		rs2	rs1	rm	rd	1010011	FADD.S
0000100		rs2	rs1	rm	rd	1010011	FSUB.S
0001000		rs2	rs1	rm	rd	1010011	FMUL.S
0001100		rs2	rs1	rm	rd	1010011	FDIV.S
0101100		00000	rs1	rm	rd	1010011	FSQRT.S
0010000		rs2	rs1	000	rd	1010011	FSGNJ.S
0010000		rs2	rs1	001	rd	1010011	FSGNJN.S
0010000		rs2	rs1	010	rd	1010011	FSGNJX.S
0010100		rs2	rs1	000	rd	1010011	FMIN.S
0010100		rs2	rs1	001	rd	1010011	FMAX.S
1100000		00000	rs1	rm	rd	1010011	FCVT.W.S
1100000		00001	rs1	rm	rd	1010011	FCVT.WU.S
1110000		00000	rs1	000	rd	1010011	FMV.X.S
1010000		rs2	rs1	010	rd	1010011	FEQ.S
1010000		rs2	rs1	001	rd	1010011	FLT.S
1010000		rs2	rs1	000	rd	1010011	FLE.S
1110000		00000	rs1	001	rd	1010011	FCLASS.S
1101000		00000	rs1	rm	rd	1010011	FCVT.S.W
1101000		00001	rs1	rm	rd	1010011	FCVT.S.WU
1111000		00000	rs1	000	rd	1010011	FMV.S.X

中国科学院软件研究所
Institute of Software Chinese Academy of Sciences

智能软件研究中心
Intelligent Software Research Center

谢 谢

欢迎交流合作

2020/04/01