```
/* M-Machine opcodes file */

#ifndef MM_OPCODES_INCLUDED
#define MM_OPCODES_INCLUDED

typedef unsigned long codebits;
#define Operation2Codebits(x) ((codebits)x)
#define Codebits_pack(x) ((int)(x >> 30))

/* interface between operation and 32-bit encoding */

/***************\
 * Integer ALU *
\***************/

typedef enum {

    NOP_ACTION,
    ILL_ACTION,

    IALU_ALIAS_ACTION,
    IALU_RENAME_ACTION,

    IALU_ILL_ACTION,

    IALU_ADD_ACTION,
    IALU_SUB_ACTION,
    IALU_ASH_ACTION,
    IALU_LSH_ACTION,
    IALU_ROT_ACTION,
    IALU_AND_ACTION,
    IALU_OR_ACTION,
    IALU_XOR_ACTION,

    IALU_NOT_ACTION,
    IALU_EXTB_ACTION,
    IALU_EXTH_ACTION,
    IALU_INSB_ACTION,
    IALU_INSH_ACTION,

    IALU_ILT_ACTION,
    IALU_ILE_ACTION,
    IALU_IGT_ACTION,
    IALU_IGE_ACTION,
    IALU_IEQ_ACTION,
    IALU_ULT_ACTION,
    IALU_ULE_ACTION,
    IALU_UGT_ACTION,
    IALU_UGE_ACTION,

    IALU_CCAND_ACTION,
    IALU_CCOR_ACTION,
    IALU_CCNAND_ACTION,

    IALU_MOV_ACTION,
    IALU_CCMOV_ACTION,
    IALU_MOVCC_ACTION,

    IALU_EMPTY_ACTION,
    IALU_CCEMPTY_ACTION,

    IALU_BR_ACTION,
    IALU_JMP_ACTION,

    IALU_LEA_ACTION,
    IALU_LEAB_ACTION,
    IALU_SETPTR_ACTION,
    IALU_ISPTR_ACTION,
    IALU_SEND0_ACTION,
    IALU_SEND0P_ACTION,
    IALU_SEND0PT_ACTION,
    IALU_SEND1P_ACTION,
    IALU_SEND1PT_ACTION,

    MEMU_ALIAS_ACTION,
    MEMU_RENAME_ACTION,
    MEMU_LD_ACTION,
    MEMU_ST_ACTION,
    MEMU_FST_ACTION,
    MEMU_LDS_ACTION,
    MEMU_LDSU_ACTION,
    MEMU_LDSCND_ACTION,
    MEMU_STS_ACTION,
    MEMU_STSU_ACTION,
    MEMU_STSCND_ACTION,
    MEMU_FSTS_ACTION,
    MEMU_FSTSU_ACTION,
    MEMU_FSTSCND_ACTION,
    MEMU_LUC_ACTION,
    MEMU_STUC_ACTION,
    MEMU_FSTUC_ACTION,
    MEMU_LEA_ACTION,
    MEMU_LEAB_ACTION,
    MEMU_LBY_ACTION,
    MEMU_SBY_ACTION,
    MEMU_LBSU_ACTION,
    MEMU_SBSU_ACTION,
    MEMU_LMS_ACTION,
    MEMU_SMS_ACTION,
    MEMU_FSMS_ACTION,
    MEMU_FLNE_ACTION,
    MEMU_FCA_ACTION,
    MEMU_GETCSTAT_ACTION,
    MEMU_PUTCSTAT_ACTION,
    MEMU_GPRB_ACTION,
    MEMU_MEMBAR_ACTION,
    MEMU_HFORK_ACTION,
    MEMU_HEXIT_ACTION,
    MEMU_VEXIT_ACTION,
    MEMU_VGSLOTU_ACTION,
    MEMU_VACQUIRE_ACTION,
    MEMU_SSTPRB_ACTION,
    MEMU_ADD_ACTION,
    MEMU_SUB_ACTION,
    MEMU_MOV_ACTION,
    MEMU_CCMOV_ACTION,
    MEMU_AND_ACTION,
    MEMU_OR_ACTION,
    MEMU_XOR_ACTION,
    MEMU_NOT_ACTION,

    FALU_ALIAS_ACTION,
    FALU_RENAME_ACTION,
    FALU_FADD_ACTION,
    FALU_FSUB_ACTION,
    FALU_FMUL_ACTION,
    FALU_FDIV_ACTION,
    FALU_FMULA_ACTION,
    FALU_FSQRT_ACTION,
    FALU_IMUL_ACTION,
```

```
    FALU_HMUL_ACTION,
    FALU_IDIV_ACTION,
    FALU_MOV_ACTION,
    FALU_EMPTY_ACTION,
    FALU_ITOF_ACTION,
    FALU_FTOI_ACTION,
    FALU_FLT_ACTION,
    FALU_FGT_ACTION,
    FALU_FEQ_ACTION
  } MM_ACTION;

#define NOP_OPCODE          0x01
#define ILL_OPCODE          0x00

#define IALU_ILL_OPCODE     0x00

#define IALU_ADD_OPCODE     0x10
#define IALU_SUB_OPCODE     0x11
#define IALU_ASH_OPCODE     0x18
#define IALU_LSH_OPCODE     0x19
#define IALU_ROT_OPCODE     0x1a
#define IALU_AND_OPCODE     0x12
#define IALU_OR_OPCODE      0x13
#define IALU_XOR_OPCODE     0x14
#define IALU_NOT_OPCODE     0x04

#define IALU_EXTB_OPCODE    0x1c
#define IALU_EXTH_OPCODE    0x1d
#define IALU_INSB_OPCODE    0x1e
#define IALU_INSH_OPCODE    0x1f

#define IALU_ILT_OPCODE     0x10
#define IALU_ILE_OPCODE     0x11
#define IALU_IGT_OPCODE     0x12
#define IALU_IGE_OPCODE     0x13
#define IALU_IEQ_OPCODE     0x14

#define IALU_ULT_OPCODE     0x15
#define IALU_ULE_OPCODE     0x16
#define IALU_UGT_OPCODE     0x17
#define IALU_UGE_OPCODE     0x18

#define IALU_CCAND_OPCODE   0x1c
#define IALU_CCOR_OPCODE    0x1d
#define IALU_CCNAND_OPCODE  0x1e

#define IALU_MOV_OPCODE     0x05
#define IALU_CCMOV_OPCODE   0x02
#define IALU_MOVCC_OPCODE   0x08
#define IALU_EMPTY_OPCODE   0x09
#define IALU_CCEMPTY_OPCODE 0x0a

#define IALU_BR_OPCODE      0x0c
#define IALU_JMP_OPCODE     0x0d

#define IALU_LEA_OPCODE     0x16
#define IALU_LEAB_OPCODE    0x17
#define IALU_SETPTR_OPCODE  0x06
#define IALU_ISPTR_OPCODE   0x19
#define IALU_SEND0_OPCODE   0x08
#define IALU_SEND0P_OPCODE  0x0a
#define IALU_SEND0PT_OPCODE 0x0b
#define IALU_SEND1P_OPCODE  0x0c
#define IALU_SEND1PT_OPCODE 0x0d

/***************\
 * Mem    ALU *
\***************/

#define MEMU_LD_OPCODE       0x10
#define MEMU_ST_OPCODE       0x12
#define MEMU_FST_OPCODE      0x13
#define MEMU_LDS_OPCODE      0x10
#define MEMU_LDSU_OPCODE     0x12
#define MEMU_LDSCND_OPCODE   0x11
#define MEMU_STS_OPCODE      0x14
#define MEMU_STSU_OPCODE     0x16
#define MEMU_STSCND_OPCODE   0x15
#define MEMU_FSTS_OPCODE     0x18
#define MEMU_FSTSU_OPCODE    0x1a
#define MEMU_FSTSCND_OPCODE  0x19

#define MEMU_LUC_OPCODE      0x14
#define MEMU_STUC_OPCODE     0x16
#define MEMU_FSTUC_OPCODE    0x17

#define MEMU_LEA_OPCODE      0x0e
#define MEMU_LEAB_OPCODE     0x0f

#define MEMU_LBY_OPCODE      0x1e
#define MEMU_SBY_OPCODE      0x1f
#define MEMU_LBSU_OPCODE     0x13
#define MEMU_SBSU_OPCODE     0x17
#define MEMU_LMS_OPCODE      0x18
#define MEMU_SMS_OPCODE      0x1a
#define MEMU_FSMS_OPCODE     0x1b
#define MEMU_FLNE_OPCODE     0x0d
#define MEMU_FCA_OPCODE      0x03
#define MEMU_GETCSTAT_OPCODE 0x1c
#define MEMU_PUTCSTAT_OPCODE 0x1d
#define MEMU_GPRB_OPCODE     0x1e
#define MEMU_MEMBAR_OPCODE   0x02

#define MEMU_HFORK_OPCODE    0x0c
#define MEMU_HEXIT_OPCODE    0x06
#define MEMU_VEXIT_OPCODE    0x07
#define MEMU_VGSLOTU_OPCODE  0x14

#define MEMU_VACQUIRE_OPCODE 0x15
#define MEMU_SSTPRB_OPCODE   0x1d
#define MEMU_ADD_OPCODE      0x08
#define MEMU_SUB_OPCODE      0x09
#define MEMU_MOV_OPCODE      0x15
#define MEMU_AND_OPCODE      0x0a
#define MEMU_OR_OPCODE       0x0b
#define MEMU_XOR_OPCODE      0x0c
#define MEMU_NOT_OPCODE      0x11

/***************\
 * Float  ALU *
\***************/

#define FALU_FADD_OPCODE    0x10
#define FALU_FSUB_OPCODE    0x11
#define FALU_FMUL_OPCODE    0x12
#define FALU_FDIV_OPCODE    0x13
#define FALU_FMULA_OPCODE   0x0c
#define FALU_FSQRT_OPCODE   0x04
```

```
#define FALU_IMUL_OPCODE    0x16
#define FALU_HMUL_OPCODE    0x14
#define FALU_IDIV_OPCODE    0x17

#define FALU_MOV_OPCODE     0x05
#define FALU_EMPTY_OPCODE   0x09

#define FALU_ITOF_OPCODE    0x06
#define FALU_FTOI_OPCODE    0x07

#define FALU_FLT_OPCODE     0x10
#define FALU_FGT_OPCODE     0x12
#define FALU_FEQ_OPCODE     0x14

/********************\
 * operand bitmasks *
\********************/
#define  A_NONE   0x000000
#define  I_DEST   0x000001 /* integer register destination     */
#define  F_DEST   0x000002 /* fp register destination          */
#define  A_DEST   0x000004 /* either fp or int register        */
#define CC_DEST   0x000008 /* conditional register destination */

#define  I_SRC1   0x000010 /* int register source              */
#define  F_SRC1   0x000020 /* fp register source               */
#define CC_SRC1   0x000040 /* conditional register source      */
#define IMME07A   0x000080 /* 7-bit immediate                  */
#define IMME11A   0x000100 /* 11-bit immediate                 */
#define IMME18A   0x000200 /* 18-bit immediate                 */

#define  I_SRC2   0x000400 /* integer register source          */
#define  F_SRC2   0x000800 /* fp register source               */
#define CC_SRC2   0x001000 /* conditional register source      */
#define  O_SRC2   0x002000 /* optional source                  */
#define IMME07B   0x004000 /* 7-bit immediate source           */
#define IMME11B   0x008000 /* 11-bit immediate source          */

#define  F_SRC3   0x010000 /* src3 is a 5-bit fp reg           */

#define    BANK   0x020000 /*  */
#define  LENGTH   0x040000 /*  */

#define   A_PRE   0x080000 /* 2-bit precondition */
#define  A_POST   0x100000 /* 1-bit postcondition */

#define LOCAL_RF  0x200000 /* Local regfile - Mask bit which only
                              allows local destinations
                              used with mem ops */
#define LOCAL_CC  0x400000 /* Local condition-register only.
                              Mask bit which only allows local
                              cc-destinations */

/* CC Registers */
#define GLOBAL_CC_FLAG   0x8
#define GLOBAL_CC_HTMASK 0x6

/* fix to the definitions to make GCC happy... */
#define IsGlobalCCReg(x)   (x & GLOBAL_CC_FLAG)
#define GlobalCCHThread(x) ((x & GLOBAL_CC_HTMASK) >> 1)
#define CCRegValue(x)      ((x & GLOBAL_CC_FLAG) ? (x & 0x1) : (unsigned)x)

#if 0
#define IsGlobalCCReg(x)   (x & GLOBAL_CC_FLAG)
#define GlobalCCHThread(x) ((x & GLOBAL_CC_HTMASK) >> 1)
#define CCRegValue(x)      (unsigned char)((x & GLOBAL_CC_FLAG) ? (x & 0x1) : x)
#endif
/**********************************\
 * operation description bitmasks *
\**********************************/
#define GENERIC_OP      0x1000
#define CONTROL_FLOW_OP 0x0001
#define DATA_LOAD_OP    0x0002
#define DATA_STORE_OP   0x0004
#define MEMREF_OP       0x0008
#define SEND_OP         0x0010

#define Operation_is_branch(x) (x->opdesc & CONTROL_FLOW_OP)
#define Operation_is_send(x) (x->opdesc & SEND_OP)
#define Operation_is_load(x) (x->opdesc & DATA_LOAD_OP)
#define Operation_is_memref(x) (x->opdesc & MEMREF_OP)

typedef enum {
    IALU,
    FALU,
    MEMU
  } MM_FUNCUNIT;

typedef enum {
    CLASS_A, /* dest:4, src1:4, src2:7 */
    CLASS_B, /* dest:4, src1:11        */
    CLASS_C, /* ccdest:4, src1:4, src2:7 */
    CLASS_D,
    CLASS_E, /* ccdest:4, ccsrc1:4, ccsrc2:7 */
    CLASS_F, /* dest:4                       */
    CLASS_G, /* src1:18                      */
    CLASS_H,
    CLASS_I,
    CLASS_J,
    CLASS_K,
    CLASS_AMBIG_JG,
    CLASS_AMBIG_AB,
    CLASS_AMBIG_CDEFHI,
    CLASS_AMBIG_CH,
    CLASS_AMBIG_CHI,
    CLASS_UNKNOWN
  } MM_CLASS;

typedef enum {
    ABS_AECI_RELOC,
    ABS_B_RELOC,
    ABS_G_RELOC,
    REL_G_RELOC,

    DATA_CHAR_RELOC,
    DATA_ASCIIZ_RELOC,
    DATA_I8_RELOC,
    DATA_I16_RELOC,
    DATA_I32_RELOC,
    DATA_I64_RELOC,
    DATA_F32_RELOC,
    DATA_F64_RELOC,
    DATA_PTR_RELOC,
    DATA_OTHER_RELOC
  } MM_RELOCATION;

/* define debug-output tag bits */
#if 0
#define DAT_FLAG 0x40 /* 01000000 */
#define M_PTR    0x00 /* 00000000 */
#define M_INT    0x41 /* 01000001 */
```

```c
#define M_UNIT   0x43 /* 01000011 */
#define M_FP     0x45 /* 01000101 */
#endif

#define OPS_FLAG 0x80 /* 10000000 */
#define OPS_IALU 0x81 /* 10000001 */
#define OPS_FALU 0x83 /* 10000011 */
#define OPS_MEMU 0x85 /* 10000101 */

#define DATA_U32 0x61 /* 01100001 */
#define DATA_S32 0x63 /* 01100011 */

/* zeroes out the src immediate */
#define CLASS_AECI_MASK  0xffffff80
#define CLASS_B_MASK     0xfffff800
#define CLASS_DFHJK_MASK 0xffffffff
#define CLASS_G_MASK     0xfffc0000

#define CLASS_A_FILL     0x00800000
#define CLASS_B_FILL     0x00200000
#define CLASS_C_FILL     0x00000000
#define CLASS_D_FILL     0x00000000
#define CLASS_E_FILL     0x00000000
#define CLASS_F_FILL     0x00000000
#define CLASS_G_FILL     0x00400000
#define CLASS_H_FILL     0x00000000
#define CLASS_I_FILL     0x00000000
#define CLASS_J_FILL     0x00000000 /* ??? */
#define CLASS_K_FILL     0x00200000

#define IALU_NOP_MASK    0x00010000
#define FALU_NOP_MASK    0x00010000
#define MEMU_NOP_MASK    0x00010000

#define IALU_ILL_MASK    0x00000000
#define FALU_ILL_MASK    0x00000000
#define MEMU_ILL_MASK    0x00000000

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 5;
    unsigned imm    : 1;
    unsigned dh     : 2;
    unsigned dr     : 1;
    unsigned dest   : 4;
    unsigned src1   : 4;
    unsigned src2   : 7;
} OpcodeBitsA;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 5;
    unsigned imm    : 1;
    unsigned dh     : 2;
    unsigned dr     : 1;
    unsigned dest   : 4;
    unsigned src1   : 11;
} OpcodeBitsB;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 8;
    unsigned imm    : 1;
    unsigned ccdest : 4;
    unsigned src1   : 4;
    unsigned src2   : 7;
} OpcodeBitsC;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 8;
    unsigned bank   : 1;
    unsigned length : 4;
    unsigned src1   : 4;
    unsigned ccdest : 3;
    unsigned src2   : 4;
} OpcodeBitsD;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 8;
    unsigned imm    : 1;
    unsigned ccdest : 4;
    unsigned ccsrc1 : 4;
    unsigned ccsrc2 : 7;
} OpcodeBitsE;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 8;
    unsigned dh     : 2;
    unsigned dr     : 1;
    unsigned dest   : 4;
    unsigned        : 9;
} OpcodeBitsF;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 5;
    unsigned imm    : 1;
    unsigned src1   : 18;
} OpcodeBitsG;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 8;
    unsigned        : 16;
} OpcodeBitsH;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 8;
```

```
    unsigned imm    : 1;
    unsigned ccdest : 4;
    unsigned dest   : 4;
    unsigned src1   : 7;
} OpcodeBitsI;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 5;
    unsigned dh     : 2;
    unsigned dr     : 1;
    unsigned dest   : 4;
    unsigned src1   : 4;
    unsigned src2   : 4;
    unsigned src3   : 4;
} OpcodeBitsJ;

typedef struct {
    unsigned pack   : 2;
    unsigned cond   : 2;
    unsigned ccreg  : 4;
    unsigned opcode : 7;
    unsigned ccdest : 3;
    unsigned dh     : 2;
    unsigned dr     : 1;
    unsigned dest   : 4;
    unsigned pre    : 2;
    unsigned post   : 1;
    unsigned src1   : 4;
} OpcodeBitsK;

#ifndef MSIM
static unsigned long MaskArray[] = {
    CLASS_AECI_MASK,
    CLASS_B_MASK,
    CLASS_AECI_MASK,
    CLASS_DFHJK_MASK,
    CLASS_AECI_MASK,
    CLASS_DFHJK_MASK,
    CLASS_G_MASK,
    CLASS_DFHJK_MASK,
    CLASS_AECI_MASK,
    CLASS_DFHJK_MASK,
    CLASS_DFHJK_MASK
  };

static unsigned long FillArray[] = {
    CLASS_A_FILL,
    CLASS_B_FILL,
    CLASS_C_FILL,
    CLASS_D_FILL,
    CLASS_E_FILL,
    CLASS_F_FILL,
    CLASS_G_FILL,
    CLASS_H_FILL,
    CLASS_I_FILL,
    CLASS_J_FILL,
    CLASS_K_FILL
  };
#endif

/* define different types of operation bit encodings */
#if 0
typedef enum {
    MM_OP0, /* nop, or operation which takes no arguments */
    MM_OP1, /* dest, src1, src2 */
    MM_OP2, /* dest, src1 */
    MM_OP3, /* src1 */
    MM_OP4, /* UNSUPPORTED */
    MM_OP5, /* ccdest:6, bank:1, length:4, src1:9, src2:9 */
    MM_OP6, /* UNSUPPORTED */
    MM_OP7, /* dest, src1, ccdest:7, pre:2, post:1 */
    MM_OP8, /* dest, src1:5, src2:5, src3:5 */
    MM_OPU /* UNSUPPORTED */
  } MM_OP;
#endif

/* OPCODE_DISPATCH_TABLE_ENTRY -- A structure for making opcode tables. */
typedef struct {
    char *name;                /* operation name */
    MM_ACTION action;          /* operation action */
    unsigned long opcode;      /* opcode bitstring */

    unsigned long arguments;   /* operation arguments */
    unsigned long opdesc;      /* operation description */
    MM_CLASS opclass;          /* operation class */
} Opcode_Dispatch_Table_Entry;

Opcode_Dispatch_Table_Entry ialu_opcode_table[]
#ifdef DEFINE
  = {
      { "nop", NOP_ACTION, NOP_OPCODE,
          A_NONE, GENERIC_OP, CLASS_H },

      { "ill", ILL_ACTION, ILL_OPCODE,
          A_NONE, GENERIC_OP, CLASS_H },

      { "add", IALU_ADD_ACTION, IALU_ADD_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "sub", IALU_SUB_ACTION, IALU_SUB_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "ash", IALU_ASH_ACTION, IALU_ASH_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "lsh", IALU_LSH_ACTION, IALU_LSH_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "rot", IALU_ROT_ACTION, IALU_ROT_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "and", IALU_AND_ACTION, IALU_AND_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "or", IALU_OR_ACTION, IALU_OR_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "xor", IALU_XOR_ACTION, IALU_XOR_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },
```

```
    { "not", IALU_NOT_ACTION, IALU_NOT_OPCODE,
        A_DEST | I_SRC1 | IMME11A,
        GENERIC_OP, CLASS_B },

    { "extb", IALU_EXTB_ACTION, IALU_EXTB_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_A },

    { "exth", IALU_EXTH_ACTION, IALU_EXTH_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_A },

    { "insb", IALU_INSB_ACTION, IALU_INSB_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_A },

    { "insh", IALU_INSH_ACTION, IALU_INSH_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_A },

    { "ilt", IALU_ILT_ACTION, IALU_ILT_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "ile", IALU_ILE_ACTION, IALU_ILE_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "igt", IALU_IGT_ACTION, IALU_IGT_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "ige", IALU_IGE_ACTION, IALU_IGE_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "ieq", IALU_IEQ_ACTION, IALU_IEQ_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "ult", IALU_ULT_ACTION, IALU_ULT_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "ule", IALU_ULE_ACTION, IALU_ULE_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "ugt", IALU_UGT_ACTION, IALU_UGT_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "uge", IALU_UGE_ACTION, IALU_UGE_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_C },

    { "ccand", IALU_CCAND_ACTION, IALU_CCAND_OPCODE,
        CC_DEST | CC_SRC1 | CC_SRC2 | IMME07B,
        GENERIC_OP, CLASS_E },

    { "ccor", IALU_CCOR_ACTION, IALU_CCOR_OPCODE,
        CC_DEST | CC_SRC1 | CC_SRC2 | IMME07B,
        GENERIC_OP, CLASS_E },

    { "ccnand", IALU_CCNAND_ACTION, IALU_CCNAND_OPCODE,
        CC_DEST | CC_SRC1 | CC_SRC2 | IMME07B,
        GENERIC_OP, CLASS_E },

    { "mov", IALU_MOV_ACTION, IALU_MOV_OPCODE,
        A_DEST | I_SRC1 | IMME11A,
        GENERIC_OP, CLASS_B },

    { "ccmov", IALU_CCMOV_ACTION, IALU_CCMOV_OPCODE,
        A_DEST,
        GENERIC_OP, CLASS_F },

    { "movcc", IALU_MOVCC_ACTION, IALU_MOVCC_OPCODE,
        I_SRC1 | IMME18A,
        GENERIC_OP, CLASS_G },

    { "empty", IALU_EMPTY_ACTION, IALU_EMPTY_OPCODE,
        I_SRC1 | IMME18A,
        GENERIC_OP, CLASS_G },

    { "ccempty", IALU_CCEMPTY_ACTION, IALU_CCEMPTY_OPCODE,
        I_SRC1 | IMME18A,
        GENERIC_OP, CLASS_G },

    { "br", IALU_BR_ACTION, IALU_BR_OPCODE,
        I_SRC1 | IMME18A,
        CONTROL_FLOW_OP, CLASS_G },

    { "jmp", IALU_JMP_ACTION, IALU_JMP_OPCODE,
        I_SRC1 | IMME18A,
        CONTROL_FLOW_OP, CLASS_G },

    { "lea", IALU_LEA_ACTION, IALU_LEA_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_A },

    { "leab", IALU_LEAB_ACTION, IALU_LEAB_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP, CLASS_A },

    { "setptr", IALU_SETPTR_ACTION, IALU_SETPTR_OPCODE,
        A_DEST | I_SRC1 | IMME11A,
        GENERIC_OP, CLASS_B },

    { "isptr", IALU_ISPTR_ACTION, IALU_ISPTR_OPCODE,
        CC_DEST | I_SRC1,
        GENERIC_OP, CLASS_I },

    { "send0", IALU_SEND0_ACTION, IALU_SEND0_OPCODE,
        CC_DEST | LOCAL_CC | BANK | LENGTH | I_SRC1 | I_SRC2,
        SEND_OP, CLASS_D },

    { "send0p", IALU_SEND0P_ACTION, IALU_SEND0P_OPCODE,
        CC_DEST | LOCAL_CC | BANK | LENGTH | I_SRC1 | I_SRC2,
        SEND_OP, CLASS_D },

    { "send0pt", IALU_SEND0PT_ACTION, IALU_SEND0PT_OPCODE,
        CC_DEST | LOCAL_CC | BANK | LENGTH | I_SRC1 | I_SRC2,
        SEND_OP, CLASS_D },

    { "send1p", IALU_SEND1P_ACTION, IALU_SEND1P_OPCODE,
        CC_DEST | LOCAL_CC | BANK | LENGTH | I_SRC1 | I_SRC2,
        SEND_OP, CLASS_D },

    { "send1pt", IALU_SEND1PT_ACTION, IALU_SEND1PT_OPCODE,
```

```
          CC_DEST | LOCAL_CC | BANK | LENGTH | I_SRC1 | I_SRC2,
          SEND_OP, CLASS_D },
      { "" }
  }
#endif
;

Opcode_Dispatch_Table_Entry falu_opcode_table[]
#ifdef DEFINE
  = {
      { "nop", NOP_ACTION, NOP_OPCODE,
          A_NONE, GENERIC_OP, CLASS_H },

      { "ill", ILL_ACTION, ILL_OPCODE,
          A_NONE, GENERIC_OP, CLASS_H },

      { "fadd", FALU_FADD_ACTION, FALU_FADD_OPCODE,
          A_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP | CLASS_A },

      { "fsub", FALU_FSUB_ACTION, FALU_FSUB_OPCODE,
          A_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP | CLASS_A },

      { "fmul", FALU_FMUL_ACTION, FALU_FMUL_OPCODE,
          A_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP | CLASS_A },

      { "fdiv", FALU_FDIV_ACTION, FALU_FDIV_OPCODE,
          A_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP | CLASS_A },

      { "fmula", FALU_FMULA_ACTION, FALU_FMULA_OPCODE,
          A_DEST | F_SRC1 | F_SRC2 | F_SRC3,
          GENERIC_OP, CLASS_J },

      { "fsqrt", FALU_FSQRT_ACTION, FALU_FSQRT_OPCODE,
          A_DEST | F_SRC1 | IMME11A,
          GENERIC_OP, CLASS_B },

      { "imul", FALU_IMUL_ACTION, FALU_IMUL_OPCODE,
          A_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "hmul", FALU_HMUL_ACTION, FALU_HMUL_OPCODE,
          A_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "idiv", FALU_IDIV_ACTION, FALU_IDIV_OPCODE,
          A_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP, CLASS_A },

      { "mov", FALU_MOV_ACTION, FALU_MOV_OPCODE,
          A_DEST | F_SRC1 | IMME11A,
          GENERIC_OP, CLASS_B },

      { "empty", FALU_EMPTY_ACTION, FALU_EMPTY_OPCODE,
          F_SRC1 | IMME18A,
          GENERIC_OP, CLASS_G },

      { "itof", FALU_ITOF_ACTION, FALU_ITOF_OPCODE,
          A_DEST | F_SRC1 | IMME11A,
          GENERIC_OP, CLASS_B },

      { "ftoi", FALU_FTOI_ACTION, FALU_FTOI_OPCODE,
          A_DEST | F_SRC1 | IMME11A,
          GENERIC_OP, CLASS_B },

      { "flt", FALU_FLT_ACTION, FALU_FLT_OPCODE,
          CC_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP, CLASS_C },

      { "fgt", FALU_FGT_ACTION, FALU_FGT_OPCODE,
          CC_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP, CLASS_C },

      { "feq", FALU_FEQ_ACTION, FALU_FEQ_OPCODE,
          CC_DEST | F_SRC1 | F_SRC2 | IMME07B,
          GENERIC_OP, CLASS_C },
      { "" }
  }
#endif
;

/* NOTE:
   I_SRC2 | O_SRC2 means that src2 argument is optional, but if provided
   must be an int reg.
   IMME07 | O_SRC2 means that src2 argument is optional, but if provided
   must be an immediate.

   So even if O_SRC2 is a flag, it does not specify the type completely.
   Please be aware.
   */

Opcode_Dispatch_Table_Entry memu_opcode_table[]
#ifdef DEFINE
  = {
      { "nop", NOP_ACTION, NOP_OPCODE,
          A_NONE, GENERIC_OP, CLASS_H },

      { "ill", ILL_ACTION, ILL_OPCODE,
          A_NONE, GENERIC_OP, CLASS_H },

      { "ld", MEMU_LD_ACTION, MEMU_LD_OPCODE,
          A_DEST | I_SRC1 | I_SRC2 | O_SRC2 | IMME07B,
          DATA_LOAD_OP | MEMREF_OP,
          CLASS_A },

      { "st", MEMU_ST_ACTION, MEMU_ST_OPCODE,
          I_DEST | LOCAL_RF | I_SRC1 | O_SRC2 | IMME07B,
          DATA_STORE_OP | MEMREF_OP,
          CLASS_A },

      { "fst", MEMU_FST_ACTION, MEMU_FST_OPCODE,
          I_DEST | LOCAL_RF | F_SRC1 | O_SRC2 | IMME07B,
          DATA_STORE_OP | MEMREF_OP,
          CLASS_A },
      /* ??? NO CC DEST ? */
      { "lds", MEMU_LDS_ACTION, MEMU_LDS_OPCODE,
          A_PRE | A_POST | I_SRC1 | A_DEST,
          DATA_LOAD_OP | MEMREF_OP,
          CLASS_K },

      { "ldsu", MEMU_LDSU_ACTION, MEMU_LDSU_OPCODE,
          A_DEST | CC_DEST | LOCAL_CC | I_SRC1 | A_PRE | A_POST,
          DATA_LOAD_OP | MEMREF_OP,
          CLASS_K },

      { "ldscnd", MEMU_LDSCND_ACTION, MEMU_LDSCND_OPCODE,
          A_DEST | CC_DEST | LOCAL_CC | I_SRC1 | A_PRE | A_POST ,
```

```
        DATA_LOAD_OP | MEMREF_OP,
        CLASS_K },

    { "sts", MEMU_STS_ACTION, MEMU_STS_OPCODE,
        I_DEST | LOCAL_RF | I_SRC1 | A_PRE | A_POST,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_K },

    { "stsu", MEMU_STSU_ACTION, MEMU_STSU_OPCODE,
        I_DEST | LOCAL_RF | CC_DEST | LOCAL_CC | I_SRC1 | A_PRE | A_POST,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_K },

    { "stscnd", MEMU_STSCND_ACTION, MEMU_STSCND_OPCODE,
        I_DEST | LOCAL_RF | CC_DEST | LOCAL_CC | I_SRC1 | A_PRE | A_POST,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_K },

    { "fsts", MEMU_FSTS_ACTION, MEMU_FSTS_OPCODE,
        I_DEST | LOCAL_RF | F_SRC1 | A_PRE | A_POST,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_K },

    { "fstsu", MEMU_FSTSU_ACTION, MEMU_FSTSU_OPCODE,
        I_DEST | LOCAL_RF | CC_DEST | LOCAL_CC | F_SRC1 | A_PRE | A_POST,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_K },

    { "fstscnd", MEMU_FSTSCND_ACTION, MEMU_FSTSCND_OPCODE,
        I_DEST | LOCAL_RF | CC_DEST | LOCAL_CC | F_SRC1 | A_PRE | A_POST,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_K },

    { "luc", MEMU_LUC_ACTION, MEMU_LUC_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | O_SRC2 | IMME07B,
        DATA_LOAD_OP | MEMREF_OP,
        CLASS_A },

    { "stuc", MEMU_STUC_ACTION, MEMU_STUC_OPCODE,
        I_DEST | LOCAL_RF | I_SRC1 | O_SRC2 | IMME07B,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_A },

    { "fstuc", MEMU_FSTUC_ACTION, MEMU_FSTUC_OPCODE,
        I_DEST | LOCAL_RF | F_SRC1 | O_SRC2 | IMME07B,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_A },

    { "lea", MEMU_LEA_ACTION, MEMU_LEA_OPCODE,
        I_DEST | LOCAL_RF | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP,
        CLASS_A },

    { "leab", MEMU_LEAB_ACTION, MEMU_LEAB_OPCODE,
        I_DEST | LOCAL_RF | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP,
        CLASS_A },

    { "lby", MEMU_LBY_ACTION, MEMU_LBY_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | O_SRC2 | IMME07B,
        DATA_LOAD_OP | MEMREF_OP,
        CLASS_A },

    { "sby", MEMU_SBY_ACTION, MEMU_SBY_OPCODE,
        I_DEST | LOCAL_RF | I_SRC1 | O_SRC2 | IMME07B,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_A },

    { "lbsu", MEMU_LBSU_ACTION, MEMU_LBSU_OPCODE,
        A_PRE | A_POST | I_SRC1 | A_DEST | CC_DEST | LOCAL_CC,
        /* DATA_LOAD_OP */ GENERIC_OP,
        CLASS_K },

    { "sbsu", MEMU_SBSU_ACTION, MEMU_SBSU_OPCODE,
        A_PRE | A_POST | I_SRC1 | I_DEST | LOCAL_RF | CC_DEST | LOCAL_CC,
        DATA_STORE_OP,
        CLASS_K },

    { "lms", MEMU_LMS_ACTION, MEMU_LMS_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | IMME07B,
        DATA_LOAD_OP | MEMREF_OP,
        CLASS_A },

    { "sms", MEMU_SMS_ACTION, MEMU_SMS_OPCODE,
        I_DEST | LOCAL_RF | I_SRC1 | IMME07B,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_A },

    { "fsms", MEMU_FSMS_ACTION, MEMU_FSMS_OPCODE,
        I_DEST | LOCAL_RF | F_SRC1 | IMME07B,
        DATA_STORE_OP | MEMREF_OP,
        CLASS_A },

    { "flne", MEMU_FLNE_ACTION, MEMU_FLNE_OPCODE,
        I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP | MEMREF_OP,
        CLASS_A },

    { "fca", MEMU_FCA_ACTION, MEMU_FCA_OPCODE,
        A_NONE, GENERIC_OP | MEMREF_OP, CLASS_H },

    { "getcstat", MEMU_GETCSTAT_ACTION, MEMU_GETCSTAT_OPCODE,
        A_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP | MEMREF_OP,
        CLASS_A },

    { "putcstat", MEMU_PUTCSTAT_ACTION, MEMU_PUTCSTAT_OPCODE,
        I_DEST | LOCAL_RF | I_SRC1 | IMME11A,
        GENERIC_OP | MEMREF_OP,
        CLASS_B },

    { "gprb", MEMU_GPRB_ACTION, MEMU_GPRB_OPCODE,
        A_DEST | CC_DEST | LOCAL_CC | I_SRC1,
        GENERIC_OP | MEMREF_OP,
        CLASS_K },

    { "mbar", MEMU_MEMBAR_ACTION, MEMU_MEMBAR_OPCODE,
        A_NONE, GENERIC_OP, CLASS_H },

    { "hfork", MEMU_HFORK_ACTION, MEMU_HFORK_OPCODE,
        CC_DEST | I_SRC1 | I_SRC2 | IMME07B,
        GENERIC_OP | MEMREF_OP,
        CLASS_C },

    { "hexit", MEMU_HEXIT_ACTION, MEMU_HEXIT_OPCODE,
        A_NONE, GENERIC_OP | MEMREF_OP, CLASS_H },

    { "vexit", MEMU_VEXIT_ACTION, MEMU_VEXIT_OPCODE,
        A_NONE, GENERIC_OP | MEMREF_OP, CLASS_H },
```

```
      { "vgsu", MEMU_VGSLOTU_ACTION, MEMU_VGSLOTU_OPCODE,
          I_DEST | LOCAL_RF | CC_DEST,
          GENERIC_OP | MEMREF_OP,
          CLASS_I },

      { "vac", MEMU_VACQUIRE_ACTION, MEMU_VACQUIRE_OPCODE,
          A_DEST | CC_DEST | I_SRC1 | IMME07A,
          GENERIC_OP | MEMREF_OP,
          CLASS_I },

      { "sstprb", MEMU_SSTPRB_ACTION, MEMU_SSTPRB_OPCODE,
          A_DEST | CC_DEST | LOCAL_CC | I_SRC1,
          GENERIC_OP | MEMREF_OP,
          CLASS_K },

      { "add", MEMU_ADD_ACTION, MEMU_ADD_OPCODE,
          I_DEST | LOCAL_RF | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP,
          CLASS_A },

      { "sub", MEMU_SUB_ACTION, MEMU_SUB_OPCODE,
          I_DEST | LOCAL_RF | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP,
          CLASS_A },

      { "mov", MEMU_MOV_ACTION, MEMU_MOV_OPCODE,
          I_DEST | LOCAL_RF | I_SRC1 | IMME11A,
          GENERIC_OP,
          CLASS_B },

      { "and", MEMU_AND_ACTION, MEMU_AND_OPCODE,
          I_DEST | LOCAL_RF | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP,
          CLASS_A },

      { "or", MEMU_OR_ACTION, MEMU_OR_OPCODE,
          I_DEST | LOCAL_RF | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP,
          CLASS_A },
      { "xor", MEMU_XOR_ACTION, MEMU_XOR_OPCODE,
          I_DEST | LOCAL_RF | I_SRC1 | I_SRC2 | IMME07B,
          GENERIC_OP,
          CLASS_A },
      { "not", MEMU_NOT_ACTION, MEMU_NOT_OPCODE,
          I_DEST | LOCAL_RF | I_SRC1 | IMME11A, /* changed from IMME11B */
          GENERIC_OP,
          CLASS_B },
      { "" }
  }
#endif
;

/*
 * Globally visible routines.
 */

/* Void normalize_opcode_destination_vectors (); */

#endif
```