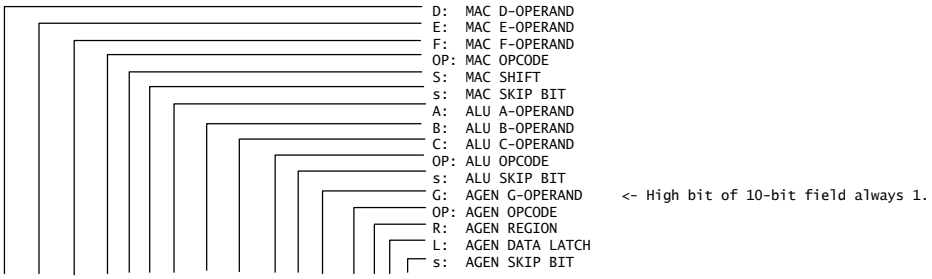
## 1. MICROINSTRUCTION WORD FORMAT.

Following is a list of the fields contained in the microinstruction.

| NAME | bit position | number of bits | description |
|---|---|---|---|
| A | 95:86 | 10 | ALU A operand address. (Also supplies condition code for ALU branching commands.) |
| B | 85:76 | 10 | ALU B operand address. (Also supplies new PC value for ALU branching commands.) |
| C | 75:66 | 10 | ALU destination address. |
| ALU_OP | 65:61 | 5 | ALU opcode. |
| ALU_SKIP | 60 | 1 | ALU skip bit. |
| D | 59:50 | 10 | MAC D operand address. |
| E | 49:40 | 10 | MAC E operand address. |
| F | 39:30 | 10 | MAC destination address. |
| MAC_OP | 29:25 | 5 | MAC opcode. |
| MAC_SHIFT | 24:21 | 4 | MAC shift amount. |
| MAC_SKIP | 20 | 1 | MAC skip bit. |
| G | 19:11 | 9 | AGEN offset register address. $10^{th}$ bit implicitly logical 1. |
| AGEN_OP | 10:8 | 3 | AGEN opcode. |
| AGEN_REG | 7:5 | 3 | AGEN region specifier. |
| AGEN_DATA | 4:1 | 4 | DIL, DOL register specifier for external memory access. |
| AGEN_SKIP | 0 | 1 | AGEN skip bit. |

File : HORNET10.VSD
Date : 1/31/95 CP

# ESP2 MACHINE CODE TO INSTRUCTION SYNTAX MAPPING




<- High bit of 10-bit field always 1.

```
        D   E   F   OP S s  A   B   C   OP s  G  OP R L s    ---------------------- Machine Code ----------------------
000 110 1d4 1cc 3ff 02 6 0  3ff 0f8 082 0b 0  2ff 0 2 0 0 = 0xff 0xcf 0x82 0x09 0x67 0x51 0xcc 0xff 0xc4 0xc7 0xf8 0x40
```

| Ucode | Field | Width | Split | Byte # | Binary | Machine Code |
|---|---|---|---|---|---|---|
| 3ff | A | 10 | 8 2 | 0 | A:**11 1111 11**11 | 0xff |
| | | | | 1 | A:11 1111 11**11** B:**00 1111** 1000 | 0xcf |
| 0f8 | B | 10 | 6 4 | 2 | B:00 1111 **1000** C:**00 10**00 0010 | 0x82 |
| 082 | C | 10 | 4 6 | 3 | C:00 10**00 0010** OP:**0** 1011 | 0x09 |
| 0b | ALU_OP | 5 | 2 3 | | | |
| 0 | ALU_SKIP | 1 | 1 | 4 | OP:0 1**011** s:**0** D:**01 11**01 0100 | 0x67 |
| 1d4 | D | 10 | 4 6 | 5 | D:01 11**01 0100** E:**01** 1100 1100 | 0x51 |
| 1cc | E | 10 | 2 8 | 6 | E:01 **1100 1100** | 0xcc |
| 3ff | F | 10 | 8 2 | 7 | F:**11 1111 11**11 | 0xff |
| 02 | MAC_OP | 5 | 5 | 8 | F:11 1111 11**11** OP:**0 0010** S:**0**110 | 0xc4 |
| 6 | MAC_SHIFT | 4 | 1 3 | | | |
| 0 | MAC_SKIP | 1 | 1 | 9 | S:0**110** s:**0** G:**0 111**1 1111 | 0xc7 |
| 2ff | G | 9 | 4 5 | 10 | G:0 111**1 1111** OP:**000** | 0xf8 |
| 0 | AGEN_OP | 3 | 3 | | | |
| 2 | AGEN_RGN | 3 | 3 | | | |
| 0 | AGEN_DATA | 4 | 4 | 11 | R:**010** L:**0000** s:**0** | 0x40 |
| 0 | AGEN_SKIP | 1 | 1 | | | |

## 2. MICROINSTRUCTION OPCODES.

### MAC

opcode

| | | |
|---|---|---|
| 0 | MACZERO + D X E > MAC >> n | > F |
| 1 | MACZERO - D X E > MAC >> n | > F |
| 2 | MACZERO + D X E >> n | > F |
| 3 | MACZERO - D X E >> n | > F |
| 4 | MACP + D X E > MAC >> n | > F |
| 5 | MACP - D X E > MAC >> n | > F |
| 6 | (MACP + D X E) >> n | > F |
| 7 | (MACP - D X E) >> n | > F |
| 8 | MAC + D X E > MAC >> n | > F |
| 9 | MAC - D X E > MAC >> n | > F |
| A | (MAC + D X E) >> n | > F |
| B | (MAC - D X E) >> n | > F |
| C | reserved | |
| D | reserved | |
| E | reserved | |
| F | reserved | |
| 10 | reserved | |
| 11 | reserved | |
| 12 | reserved | |
| 13 | reserved | |
| 14 | MACP >> n + D X E | > F,MAC |
| 15 | MACP >> n - D X E | > F,MAC |
| 16 | MACP >> n + D X E | > F |
| 17 | MACP >> n - D X E | > F |
| 18 | MAC >> n + D X E | > F,MAC |
| 19 | MAC >> n - D X E | > F,MAC |
| 1A | MAC >> n + D X E | > F |
| 1B | MAC >> n - D X E | > F |
| 1C | reserved | |
| 1D | reserved | |
| 1E | reserved | |
| 1F | reserved | |

Parentheses not required.

## ALU

| Instruction | opcode |
|---|---|
| ADD | $00 |
| ADDV | $01 |
| ADDC | $02 |
| AMDF | $0E |
| AND | $08 |
| AS | $0F |
| ASDH | $11 |
| ASDL | $12 |
| AVG | $0D |
| BIOZ | $19 |
| BREV | $15 |
| DREV | $16 |
| HOST | $18 |
| Jcc | $1C |
| JScc | $1D |
| LIM | $17 |
| LS | $10 |
| LSDH | $13 |
| LSDL | $14 |
| MAX | $06 |
| MIN | $07 |
| MOV | $0B |
| MOVcc | $1B |
| OR | $09 |
| RECT | $0C |
| REPT | $1F |
| RScc | $1E |
| SUB | $03 |
| SUBB | $05 |
| SUBREV | $1A |
| SUBV | $04 |
| XOR | $0A |

## AGEN

| Opcode | Operation |
|---|---|
| 0 | External memory read. |
| 1 | External memory write. |
| 2 | External memory read, update BASE. |
| 3 | External memory write, update BASE. |
| 4 | External memory read, plus one addressing. |
| 5 | External memory write, plus one addressing |
| 6 | NOP cycle on the external memory bus. |
| 7 | Update BASE. |

# Ensoniq ESP2 Object Format Specification

Version 1.0
27 March 1995

This document describes the format of ESP2 object files. An ESP2 object file is composed of several parts. Each part consists of a unique tag followed by the size of the part (excluding the tag and size fields) then one or more records. Each record consists of a unique tag followed by a data block of known organization.

The use of part tags and record tags allows the design of an object loader that supports forward and backward compatibility across revisions of the object format. Forward compatibility can be achieved by ignoring unknown tags. Backward compatibility can be achieved by adding new tags.

## 1. Special Data Formats of Variable Size

For compactness of representation of certain quantities, special variable-size data formats are used. These differ from the data types of, for example, the C programming language, where the size of a given type is fixed. In the special formats defined here, the size depends on the actual data to be represented.

The `NUMBER` format is used to represent integer quantities. The `NAME` format is used to represent ASCII character strings.

### 1.1 `NUMBER` Format

The `NUMBER` format is defined as follows:

```
BYTE        byte_count
BYTE        bytes[]
```

where `BYTE` indicates an 8-bit unsigned quantity, and the square brackets (‘`[]`’) indicate an array of arbitrary size.

If the high bit of `byte_count` is set, then the low-order seven bits give the number of bytes that follow. In that case, `bytes[]` contains the indicated number of bytes of numeric data, with the most-significant byte first.

If the high bit of `byte_count` is cleared, then the low-order seven bits comprise the actual datum (in which case no more bytes follow).

### 1.2 `NAME` Format

The `NAME` format is defined as follows:

```
BYTE        char_count
```

```
CHAR        characters[]
```

where BYTE is defined as above, and CHAR indicates an ASCII character.

char_count holds the number of characters that follow. characters[] contains the indicated number of ASCII characters.

## 2. Other Data Formats

Several additional data formats are defined here.

### 2.1 Word Format: WORD

```
WORD:
BYTE            ms_byte         * most-significant byte
BYTE            ls_byte         * least-significant byte
```

### 2.2 Word-Length-Tag Format: TAG_WORD

```
TAG_WORD:
WORD            tag             * for header_part_tag
```

### 2.3 Byte-Length-Tag Format: TAG_BYTE

```
TAG_BYTE:
BYTE            tag             * all tags but header_part_tag
```

### 2.4 Register-Initialization Format: REG_INIT

```
REG_INIT:
NUMBER          address         * register address
NUMBER          data            * initialization data
```

### 2.5 Register-Array format: REG_ARRAY

```
REG_ARRAY:
NUMBER          address         * base address
NUMBER          array_size      * array size
NUMBER          data            * initialization data
```

### 2.6 Internal-Table Format: INT_TABLE

```
INT_TABLE:
NUMBER          address         * ext-mem absolute address
NUMBER          size            * declared table size
NUMBER          table_id        * internal-table id
```

### 2.7 External-Table Format: EXT_TABLE

```
EXT_TABLE:
NUMBER          address         * ext-mem absolute address
```

```
NUMBER          size            * declared table size
NAME            table_file_name * name of table file
```

### 2.8 RAM-Initialization Format: `RAM_INIT`

```
RAM_INIT:
NUMBER          address         * ext-mem absolute address
NUMBER          size            * ram-block size
NUMBER          data            * initialization data
```

### 2.9 Operand-Reference Format: `OPND_REF`

```
OPND_REF:
BYTE            opnd_bitmap     * relocatable-operand bitmap
```

### 2.10 Microinstruction Format: `UINST`

```
UINST:
BYTE            uinst[12]       * microinstruction word
```

### 2.11 Symbol-Table Format: `SYMTAB`

```
SYMTAB:
NUMBER          value           * sym value (e.g., address)
NAME            name            * symbol name
```

## 3. Object-File Components

The object-file components listed below are described in the following sections.

- Header Part
  - Algorithm ID Part Offset
  - Resource Part Offset
  - Register Part Offset
  - Initialization Part Offset
  - Relocation Part Offset
  - Microinstruction Part Offset
  - Debug Part Offset
  - Trailer Part Offset
- Algorithm ID Part
  - Algorithm ID Record
- Resource Part
  - Resource Record
- Register Part
  - Consecutive-Register Record
  - Non-Consecutive-Register Record
- Initialization Part
  - Consecutive Initialized Registers Record
  - Non-Consecutive Initialized Registers Record
  - Register Array Record
  - Internal Table Record
  - External Table Record
  - RAM Initialization Record

Relocation Part
    Relocation Record
Microinstruction Part
    Microinstruction Record
Debug Part
    Register Debug Record
Trailer Part
    Checksum Record
    Module End Record

## 3.1 Header Part

The Header part contains version information and offsets to the other parts of the object file. The Header part must be present as the first part in the file.

```
TAG_WORD          header_part_tag               = 0x4532
NUMBER            header_size
NUMBER            object_format_version
```

### 3.1.1 Algorithm ID Part Offset

This record contains an offset in bytes to the Algorithm ID part relative to the beginning of the file. An offset of zero indicates that the Algorithm ID part is not included in the file.

```
TAG_BYTE          algo_id_part_offset_tag       = 0x00
NUMBER            algo_id_part_offset
```

### 3.1.2 Resource Part Offset

This record contains an offset in bytes to the Resource part relative to the beginning of the file. An offset of zero indicates that the Resource part is not included in the file.

```
TAG_BYTE          resource_part_offset_tag      = 0x01
NUMBER            resource_part_offset
```

### 3.1.3 Register Part Offset

This record contains an offset in bytes to the Register part relative to the beginning of the file. An offset of zero indicates that the Register part is not included in the file.

```
TAG_BYTE          reg_part_offset_tag           = 0x02
NUMBER            reg_part_offset
```

### 3.1.4 Initialization Part Offset

This record contains an offset in bytes to the Initialization part relative to the beginning of the file. An offset of zero indicates that the Initialization part is not included in the file.

```
TAG_BYTE          init_part_offset_tag          = 0x03
```

```
NUMBER          init_part_offset
```

### 3.1.5 Relocation Part Offset

This record contains an offset in bytes to the Relocation part relative to the beginning of the file. An offset of zero indicates that the Relocation part is not included in the file.

```
TAG_BYTE        reloc_part_offset_tag       = 0x04
NUMBER          reloc_part_offset
```

### 3.1.6 Microinstruction Part Offset

This record contains an offset in bytes to the Microinstruction part relative to the beginning of the file. An offset of zero indicates that the Microinstruction part is not included in the file.

```
TAG_BYTE        uinst_part_offset_tag       = 0x05
NUMBER          uinst_part_offset
```

### 3.1.7 Debug Part Offset

This record contains an offset in bytes to the Debug part relative to the beginning of the file. An offset of zero indicates that the Debug part is not included in the file.

```
TAG_BYTE        debug_part_offset_tag       = 0x06
NUMBER          debug_part_offset
```

### 3.1.8 Trailer Part Offset

This record contains an offset in bytes to the Trailer part relative to the beginning of the file. An offset of zero indicates that the Trailer part is not included in the file.

```
TAG_BYTE        trailer_part_offset_tag     = 0x07
NUMBER          trailer_part_offset
```

## 3.2 Algorithm ID Part

The Algorithm ID part contains a classification of the algorithm that is implemented by the program.

```
TAG_BYTE        algo_id_part_tag            = 0x00
NUMBER          algo_id_part_size
```

### 3.2.1 Algorithm ID Record

```
TAG_BYTE        algo_id_record_tag          = 0x00
NUMBER          algo_family
NUMBER          algo_member
NAME            algo_name
```

### 3.3 Resource Part

The Resource part contains a list of the ESP2 resources that are required by the program.

```
TAG_BYTE          resource_part_tag             = 0x01
NUMBER            resource_part_size
```

3.3.1 Resource Record

```
TAG_BYTE          resource_record_tag           = 0x00
NUMBER            num_inst
NUMBER            num_gprs
NUMBER            num_aors
NUMBER            num_regions
NUMBER            dil_bitmap
NUMBER            dol_bitmap
NUMBER            tbl_mem_size
NUMBER            ddl_mem_size
NUMBER            exc_time
```

### 3.4 Register Part

The Register part contains a list of all relocatable Register (GPR or AOR) addresses referenced by the program. Each tagged record in the Register part can reference a string of consecutive Register addresses or a number of non-consecutive ones.

```
TAG_BYTE          reg_part_tag                  = 0x02
NUMBER            reg_part_size
```

3.4.1 Consecutive Register Record

```
TAG_BYTE          mreg_record_tag               = 0x00
NUMBER            num_regs
NUMBER            start_address
```

3.4.2 Non-Consecutive Register Record

```
TAG_BYTE          sreg_record_tag               = 0x01
NUMBER            num_regs
NUMBER            reg_addrs[]
```

### 3.5 Initialization Part

The Initialization part contains register- and RAM-initialization data.

```
TAG_BYTE          init_part_tag                 = 0x03
NUMBER            init_part_size
```

The first two record types in the Initialization part list ESP2 registers (GPRs, AORs, and SPRs) and their initialization values. The register addresses may be consecutive or non-consecutive.

### 3.5.1 Consecutive Initialized Registers Record

```
TAG_BYTE          init_mreg_record_tag          = 0x00
NUMBER            num_regs
NUMBER            start_address
NUMBER            reg_data[]
```

### 3.5.2 Non-Consecutive Initialized Registers Record

```
TAG_BYTE          init_sreg_record_tag          = 0x01
NUMBER            num_reg_inits
REG_INIT          reg_inits[]
```

The Register Array record lists blocks of consecutive GPRs or AORs that are to be initialized to a common value.

### 3.5.3 Register Array Record

```
TAG_BYTE          reg_array_record_tag          = 0x02
NUMBER            num_reg_arrays
REG_ARRAY         reg_arrays[]
```

The next two record types in the Initialization part list tables to be downloaded to ESP2 RAM. The tables may be internal (referenced by index) or external (contained in a separate file). Each record can reference a number of internal tables or external tables.

### 3.5.4 Internal Table Record

```
TAG_BYTE          internal_table_record_tag     = 0x03
NUMBER            num_internal_tables
INT_TABLE         internal_tables[]
```

### 3.5.5 External Table Record

```
TAG_BYTE          external_table_record_tag     = 0x04
NUMBER            num_external_tables
EXT_TABLE         external_tables[]
```

The RAM Initialization record lists a number of blocks of ESP2 RAM that are to be initialized (to 0, for example).

### 3.5.6 RAM Initialization Record

```
TAG_BYTE          ram_init_record_tag           = 0x05
NUMBER            num_ram_inits
RAM_INIT          ram_inits[]
```

### 3.6 Relocation Part

The Relocation part contains information used in relocating GPRs, AORs, and microinstruction addresses.

```
TAG_BYTE          reloc_part_tag                = 0x04
NUMBER            reloc_part_size
```

3.6.1 Relocation Record

```
TAG_BYTE          reloc_record_tag              = 0x00
NUMBER            num_opnd_refs
OPND_REF          opnd_refs[]
```

### 3.7 Microinstruction Part

The Microinstruction part contains the array of assembled microinstruction bytes.

```
TAG_BYTE          uinst_part_tag                = 0x05
NUMBER            uinst_part_size
```

3.7.1 Microinstruction Record

```
TAG_BYTE          uinst_record_tag              = 0x00
NUMBER            num_uinsts
UINST             uinsts[]
```

### 3.8 Debug Part

The Debug part contains Register (GPR and AOR) symbol tables.

```
TAG_BYTE          debug_part_tag                = 0x06
NUMBER            debug_part_size
```

3.8.1 Register Debug Record

```
TAG_BYTE          reg_debug_record_tag          = 0x00
NUMBER            num_regs
SYMTAB            reg_symtab[]
```

### 3.9 Trailer Part

The Trailer part must be present as the last part in the file.

```
TAG_BYTE          trailer_part_tag              = 0x07
NUMBER            trailer_part_size
```

3.9.1 Checksum Record

The Checksum record contains a 16-bit CRC checksum word. The checksum is computed over all the bytes in the file up to and including the `checksum_record_tag` (but not including any subsequent records).

```
TAG_BYTE          checksum_record_tag          = 0x00
WORD              checksum
```

### 3.9.2 Module End Record

The Module End record must be present as the last record in the file:

```
TAG_BYTE          module_end_record_tag        = 0xff
```

## Appendix A. List of Tags

### Part Tags

Part tags, with the exception of the Header-part tag, are one byte in length. The Header-part tag consists of two bytes: ASCII ‘E’ followed by ASCII ‘2’. The tag for a part designed in the future shall have its high bit set if the part contains information that is essential to correct operation of the program. If a part is optional, or if it is a member of the initial set of parts described in this document, then the high bit of the tag shall not be set.

```
header_part_tag              = 0x4532

algo_id_part_tag             = 0x00
resource_part_tag            = 0x01
reg_part_tag                 = 0x02
init_part_tag                = 0x03
reloc_part_tag               = 0x04
uinst_part_tag               = 0x05
debug_part_tag               = 0x06
trailer_part_tag             = 0x07
```

### Offset Tags

Offset tags are one byte in length. An unknown offset record can be skipped by the object loader, with parsing resuming at the next offset record, since all offset records are of a known format. Each offset tag has a corresponding part tag of identical bit pattern.

```
algo_id_part_offset_tag      = 0x00
resource_part_offset_tag     = 0x01
reg_part_offset_tag          = 0x02
init_part_offset_tag         = 0x03
reloc_part_offset_tag        = 0x04
uinst_part_offset_tag        = 0x05
debug_part_offset_tag        = 0x06
trailer_part_offset_tag      = 0x07
```

### Record Tags

Record tags are one byte in length. To promote forward compatibility, records designed in the future to supplement the records of any existing part shall have the high bit of the tag set, and shall contain a size field in NUMBER format immediately following the tag. An unknown record can thus be skipped by the object loader, with parsing resuming at the next record, by adding the size value to the current object-file offset.

```
algo_id_record_tag            = 0x00
resource_record_tag           = 0x00
mreg_record_tag               = 0x00
sreg_record_tag               = 0x01
init_mreg_record_tag          = 0x00
init_sreg_record_tag          = 0x01
reg_array_record_tag          = 0x02
internal_table_record_tag     =                                0x03
external_table_record_tag     = 0x04
ram_init_record_tag           = 0x05
reloc_record_tag              = 0x00
uinst_record_tag              = 0x00
reg_debug_record_tag          = 0x00
checksum_record_tag           = 0x00
module_end_record_tag         = 0xff
```