3 Memory organization

All *short pointers* access memory locations within an MDS. They access all local and some global frames (§3.2.2.2)

3.2.2.2 Local and global frames

Global Frames

Programming Note: Except for the restriction that frames are contained entirely within a 64K bank, the maximum size of a frame is not specified by the architecture.

```
GlobalFrameHandle: TYPE = LONG POINTER TO GlobalVariables;
```

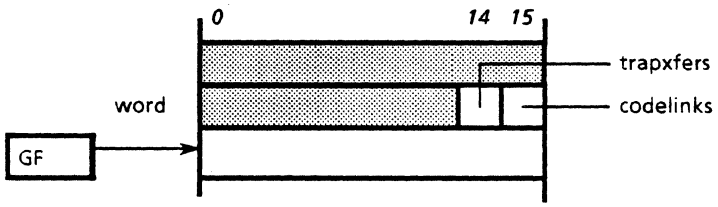

Figure 3.4 Global frame

The overhead words contain the flag bits trapxfers and codelinks used during control transfers (§9.3)

```
GlobalFrameBase: TYPE = LONG POINTER TO GlobalOverhead;

GlobalWord: TYPE = MACHINE DEPENDENT RECORD [
    available (0: 0..13): [0..37777B],
    trapxfers (0: 14..14): BOOLEAN,
    codelinks (0: 15..15): BOOLEAN];

GlobalOverhead: TYPE = MACHINE DEPENDENT RECORD [
    available (0): UNSPECIFIED,
    word (1): GlobalWord,
    global (2): GlobalVariables];
```

Local Frames

The globallink points to the procedure's global frame index. It is used to gain access to the procedure's global variables.

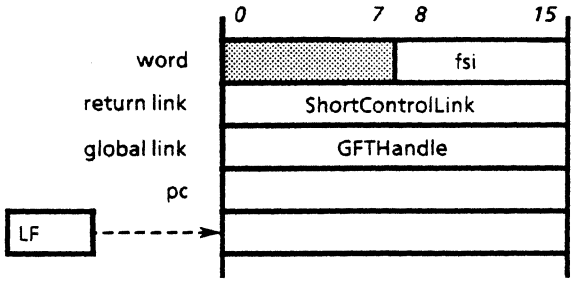

Figure 3.5 Local frame

```
LocalOverhead: TYPE = MACHINE DEPENDENT RECORD [
    word (0): LocalWord,
    returnlink (1): ShortControlLink,
    globallink (2): GFTHandle,
    pc (3): CARDINAL,
    local (4): LocalVariables];
```

3.3.1 Control registers

Notice that a local frame is sufficient to determine all of the other registers: given a local frame pointer, the program counter is obtained from its pc field, the global frame index from its globallink field, and the global frame handle and code segment address from the global frame table.

The global frame index of the current context is contained in the sixteen bit register GFI. Its value is obtained using LocalBase[LF].globallink.

```
GFI: GFTHandle;
```

The address of the global frame of the current context is contained in the 32 bit register GF. Its value is obtained using GFT[GFI].globalFrame.

```
GF: GlobalFrameHandle;
```

The address of the code segment of the current context is contained in the register CB (the code base, a long pointer). Its value is obtained using GFT[GFI].codebase.

7 Assignment instructions

7.2.2 Global frame access

LGAn Long Global Address n

```
LGAn: PROCEDURE [n: [0..1]] =
    BEGIN
    PushLong[GF + n];
    END;
```

LGAB Long Global Address Byte

```
LGAB: PROCEDURE =
    BEGIN
    alpha: BYTE = GetCodeByte[];
    PushLong[GF + alpha];
    END;
```

LGAW Long Global Address Word

```
LGAW: PROCEDURE =
    BEGIN
    word: UNSPECIFIED = GetCodeWord[];
    PushLong[GF + word];
    END;
```

7.2.2.1 Load global

LGn Load Global n

```
LGn: PROCEDURE [n: [0..2]] =
    BEGIN
    Push[Fetch[GF + n] ↑ ];
    END;
```

LGB Load Global Byte

```
LGB: PROCEDURE =
    BEGIN
    alpha: BYTE = GetCodeByte[];
    Push[Fetch[GF + alpha] ↑ ];
    END;
```

LGDn Load Global Double n

```
LGDn: PROCEDURE [n: [0,2]] =
    BEGIN
    Push[Fetch[GF + n] ↑ ];
    Push[Fetch[GF + n + 1] ↑ ];
    END;
```

LGDB Load Global Double Byte

```
LGDB: PROCEDURE =
    BEGIN
```

```
    alpha: BYTE = GetCodeByte[];
    Push[Fetch[GF + alpha] ↑ ];
    Push[Fetch[GF + alpha + 1] ↑ ];
    END;
```

7.2.2.2 Store global

SGB Store Global Byte

```
SGB: PROCEDURE =
    BEGIN
    alpha: BYTE = GetCodeByte[];
    Store[GF + alpha] ↑ ← Pop[];
    END;
```

SGDB Store Global Double Byte

```
SGDB: PROCEDURE =
    BEGIN
    alpha: BYTE = GetCodeByte[];
    Store[GF + alpha + 1] ↑ ← Pop[];
    Store[GF + alpha] ↑ ← Pop[];
    END;
```

7.3.2.1 Read indirect

RGIP Read Global Indirect Pair

```
RGIP: PROCEDURE =
    BEGIN
    pair: NibblePair = GetCodeByte[];
    ptr: POINTER = Fetch[GF + pair.left] ↑ ;
    Push[FetchMds[ptr + pair.right] ↑ ];
    END;
```

RGILP Read Global Indirect Long Pair

```
RGILP: PROCEDURE =
    BEGIN
    pair: NibblePair = GetCodeByte[];
    ptr: LONG POINTER = ReadDbl[GF + pair.left];
    Push[Fetch[ptr + LONG[pair.right]] ↑ ];
    END;
```

9 Data types

9.1 Control links

```
LinkType: TYPE = {
    frame, oldProcedure, indirect, newProcedure};
```

```
ControlLinkType: PROCEDURE [link: ControlLink]
    RETURNS [LinkType] =
    BEGIN
    cl: TaggedControlLink = LOOPHOLE[link];
    RETURN[
      SELECT cl.tag FROM
          0 => frame,
          1 => oldProcedure,
          2 => indirect,
          ENDCASE => newProcedure];
    END;
```

9.1.4 New procedure descriptors

A *procedure descriptor* is used to create a new context. It contains the information necessary to obtain the global frame index GFI, the global frame pointer GF, the code segment pointer CB, the local frame pointer LF, and the starting PC value for the procedure. It consists of two fields:

```
NewProcDesc: TYPE = MACHINE DEPENDENT RECORD [
    taggedGFI(0):UNSPECIFIED,
    pc(1): CARDINAL];

MakeNewProcDesc: PROCEDURE [link: ControlLink]
    RETURNS [NewProcDesc] =
    BEGIN
    IF ControlLinkType[link] # newProcedure THEN ERROR;
    RETURN[LOOPHOLE[link]];
    END;
```

The taggedGFI is the global frame index or'ed with 3. The new value of GFI is computed from taggedGFI. The global frame and the code base are then obtained from the global frame table using GFI.

The following is a sketch of this process (ignoring traps and other types of control transfers). §9.3 contains a complete description.

```
proc: ProcDesc;
...
GFI ← And[proc.taggedGF, 177774B];
GF ← ReadDbl[@GFT[GFI].globalFrame];
CB ← ReadDbl[@GFT[GFI].codebase];
...
```

9.1.4.1 Global frame table

The Global Frame Table (GFT) contains the global frame handle and codebase of each module instance. It is at a fixed location GFT. The table is organized as an array; elements can be accessed by index GFTIndex, or by relative pointer GFTHandle.

```
GFT: LONG POINTER TO GlobalFrameTable =
    LOOPHOLE[mGFT];
GlobalFrameTable: TYPE = LONG BASE POINTER TO
```

```
        ARRAY GFTIndex OF GFTItem;

    GFTIndex: TYPE = [0..16384);
    GFTHandle: TYPE = GlobalFrameTable RELATIVE POINTER TO
        GFTItem;

    GFTItem: TYPE = MACHINE DEPENDENT RECORD[
        globalFrame(0): GlobalFrameHandle,
        codebase(2): LONG POINTER TO CodeSegment];
```

Programming Note: By convention, the first entry of the GFT is not used. Procedure descriptors with a gfi of zero will always result in an **UnboundTrap**.

The GFT can be shorter than the maximum size specified above. In this case, it is the responsibility of the programmer to ensure that no gfi's that exceed the size of the GFT are used; the processor does no dynamic bounds checking on gfi's.

9.1.4.2 Descriptor instruction

The *descriptor* instruction creates a procedure descriptor using GFI and a pc.

DESC Descriptor

```
DESC: PROCEDURE =
    BEGIN
    word: UNSPECIFIED = GetCodeWord[];
    Push[Or[GFI, 3]];
    Push[word];
    END;
```

9.3 Control transfer primitive

```
SELECT ControlLinkType[nDst] FROM
  newProcedure = >
    BEGIN
    word: BytePair;
    proc: NewProcDesc = MakeNewProcDesc[nDst];
    GFI ← And[proc.taggedGF, 177774B];
    IF GFI = LOOPHOLE[0] THEN UnboundTrap[dst];
    GF ← ReadDbl[@GFT[GFI].globalFrame];
    CB ← ReadDbl[@GFT[GFI].codebase];
    IF Odd[LowHalf[CB]] THEN CodeTrap[GFI];
    nPC ← proc.pc;
    IF nPC = 0 THEN UnboundTrap[dst];
    word ← ReadCode[nPC/2];
    nLF ← Alloc[IF nPC MOD 2 = 0 THEN word.left
        ELSE word.right];
    nPC ← nPC + 1;
    StoreMds[@LocalBase[nLF].globallink]↑ ← GFI;
    StoreMds[@LocalBase[nLF].returnlink]↑ ← src;
    END;

  frame = >
    BEGIN
```

```
frame: FrameLink = MakeFrameLink[nDst];
IF frame = LOOPHOLE[0] THEN ControlTrap[src];
nLF ← frame;
GFI ← FetchMds[@LocalBase[nLF].globallink] ↑ ;
IF GFI = LOOPHOLE[0] THEN UnboundTrap[dst];
GF ← ReadDbl[@GFT[GFI].globalFrame];
CB ← ReadDbl[@GFT[GFI].codebase];
IF Odd[LowHalf[CB]] THEN CodeTrap[GFI];
...
```

9.4.1 Local function calls

```
LFC: PROCEDURE =
    ...
    StoreMds[@LocalBase[nLF].globallink] ↑ ← GFI;
    ...
```

9.4.2 External function calls

```
FetchLink: PROCEDURE [offset: BYTE] RETURNS [ControlLink] =
    BEGIN
    word: GlobalWord = Fetch[@GlobalBase[GF].word] ↑ ;
    RETURN[
       IF word.codelinks THEN ReadDbl[CB-LONG[(offset + 1)*2]]
       ELSE ReadDbl[GlobalBase[GF]-(offset + 1)*2]];
    END;
```

Design Note: If the links are stored in the code segment, they must be contained in the same 64K bank as the code base. This ensures that the calculation of the address of the link will not underflow in the low-order word or cause a borrow from the high-order word. Since frames are always completely contained in a 64K bank, this calculation is also accurate if the links are stored in the global frame.

9.5.1 Trap routines

```
CodeTrap: PROCEDURE [gfi: GFTHandle] = {
    TrapOne[@SD[sCodeTrap], gfi]};
```

Appendix A Values of constants

Appendix A.2 Constant memory locations

mGFT Global Frame Table

```
mGFT: LONG CARDINAL = 400000B;
```

Appendix B Opcodes

LGAn Long Global Address n (n = 0..1)
Push 32-bit value G + n.

LGAB Long Global Address Byte
Push 32-bit value G + alpha.

LGAW Long Global Address Word
Push 32-bit value G + alphabeta.

DESC Descriptor
Push 32-bit procedure descriptor with pc alphabeta.