HYPOMAC

GENERAL DESCRIPTION

Registers

HYPOMAC has 10 address registers. By convention

```
Reg 0     always contains 0
    1     contains display pointer (DP)
    2     contains stack pointer (SP)
    3     contains local block address
    4     contains argument pointer (AP)
    5 |
    6 |   are available for general use
    7 |       (optimization)
    8 |
    9 |
```

Memory

The memory is divided into two separate areas called ISPACE and DSPACE. The data area, DSPACE, is organized into 32-bit words with 4 8-bit characters per word. The lowest address in DSPACE is 1. The hardware maintains internal type bits associated with each word in DSPACE. The loader sets the type of each word when it is loaded with data, and each operation sets the type of the result. Although the hardware uses the type bits for internal checking (e.g. to insure that a real value is never used as an address), the type is not available to the programmer. Data values may have internal type integer, real, Boolean, alfa or char. Initially all of DSPACE that is not explicitly loaded is undefined.

HYPOMAC contains a separate instruction space, ISPACE, which contains all instructions. An instruction consists of an opcode and up to two operands. The first ISPACE address is 0. Each location in ISPACE can hold exactly 1 instruction. After the instructions have been loaded, ISPACE cannot be modified. All elements of ISPACE that are not explicitly loaded are initialized to 0.

Architecture

HYPOMAC is a stack oriented, one's complement machine. Arithmetic operations, for example, operate on the top 2 elements of the stack and leave the result on the top of the stack. Also, the display is pushed when a procedure is called, and popped when the return is executed. Register 2 always contains the current top of stack address (SP). The stack grows from low to high; a PUSH increases the value in register 2. HYPOMAC strictly enforces type compatibility, any violation of it will cause an error stop. The stack exists in DSPACE, and stack elements have the same characteristics as DSPACE elements.

Addressing

HYPOMAC allows direct and indirect addressing. Calculating an indirect address requires an extra memory access, so direct addressing is more efficient. Each address in an instruction consists of a 9 digit integer of the form IRXXYYYYY, where

```
              I is the indirect digit
              R is the address register
             XX is the preindirect offset
          YYYYY is the postindirect offset
                and any of the values may be 0.
```

```
IF I = 0, ADDRESS <- CONTENTS(R) + YYYYY
IF I <> 0, ADDRESS <- YYYYY + MEM(XX+CONTENTS(R))
```

Operating System

HYPOMAC allows the user to dynamically open and close files. Once a file is opened, there are system functions which allow it to be used for input and/or output. There are also system functions CHR, ORD, and SQR, as well as a set of procedures which allow the user to dump specific parts of the machine's memory and register space. Specific details about use of files are included in the implementation notes at the end of this document.

HYPOMAC also provides a simple loader which loads the user's program into memory.

INSTRUCTION SET

Legal opcodes range from 1 to 45. Attempted execution of an instruction with opcode out of range causes an error stop. Opcodes have 0, 1 or 2 operands. The general types of operands are:

loc1,loc2 - A legal data address. Unless otherwise indicated, data addresses are checked and out-of-range will cause an error stop.

count - A positive integer.

value - An integer, real, boolean, char or alfa value.

reg1,reg2 - A legal register index (0-9). An out-of-range index will cause an error stop.

type - Used with arithmetic operations.

0 => integer operation

1 => real operation

S - Used with unary functions.

0 => do operation on the element on top of the stack

1 => do operation on the second element from the top of the stack

iloc - An instruction address. It is checked when the next instruction fetch is done.

char - Index of a character in an alfa (1-10).

intgr - An integer value.

NOTES ON INSTRUCTIONS

1. Assigning to register 0 is not an error, but it does not change the value of register 0 (which is 0).

2. Registers are of type integer.

3. Any time the stack is pushed, it is checked for overflow and every time it is popped it is checked for underflow. Either condition causes an error stop.

4. Any operation involving the top 2 stack elements produces the result:

   RESULT <- NEXT TO TOP ELEMENT op TOP ELEMENT

5. For Boolean operations, the operands must be Boolean (they cannot be 0 or 1).

6. PUSHR 2 increments the stack pointer and *then* pushes the current value of register 2.

   POPR 2 pops the top of the stack into register 2 and *then* decrements the stack pointer.

7. PUSH with count=0 is allowed, but a warning message is printed since it often is an error.

8. The stack pointer always points to the last reserved (used) position on the stack.

Opcode	**Mnemonic**	**Op1**	**Op2**	**Description**
1	PUSH	loc1	count	If loc1 = 0, SP is incremented by count but nothing is pushed. If loc1 <> 0, DSPACE(loc1) ... DSPACE(loc1+count-1) is pushed on the stack.
2	PUSHR	reg1	-	The contents of reg1 is pushed.
3	PUSHI	value	-	Value is pushed.
4	PUSHC	loc1	char	The charth character in DSPACE(loc1) is pushed.
5	POP	loc1	count	The stack is popped count times. If loc1 <> 0, stack(SP-count+1) ... stack(SP) are stored in DSPACE, starting at loc1.
6	POPC	loc1	char	The stack is popped, and the charth character in the popped value is stored in DSPACE(loc1).
7	POPR	reg1	-	The stack is popped and the value is stored in reg1.
8	MOVE	loc1	loc2	DSPACE(loc1) is assigned the contents of DSPACE(loc2).
9	SWAP	-	-	The contents of the top of the stack and the next to the top of the stack are exchanged.
10	LOAD	reg1	loc1	Register reg1 is loaded with the contents of DSPACE(loc1).
11	LOADR	reg1	reg2	Register reg1 is loaded with the contents of register reg2.
12	LOADA	reg1	loc1	Register reg1 is loaded with loc1. Loc1 is not checked.

Opcode	Mnemonic	Op1	Op2	Description
13	LOADI	reg1	intgr	Register reg1 is loaded with the integer value, intgr.
14	STORE	reg1	loc1	The contents of register reg1 is stored in DSPACE(loc1).
15	STOREREGS	loc1	count	The contents of register ((i+3) mod 10) is stored in DSPACE(loc1+i) for i = 0 to count-1.
16	LOADREGS	loc1	count	Register ((i+3) mod 10) is loaded with the contents of DSPACE(loc1+i) for i = 0 to count-1.
17	HALT	-	-	NORMAL TERMINATION
18	ADD	type	-	The stack is popped twice, the addition is performed, and the result is pushed.
19	SUB	type	-	The stack is popped twice, the subtraction is performed, and the result is pushed.
20	NEGATE	type	-	The real or integer on the top of the stack is negated.
21	MULT	type	-	The stack is popped twice, the multiplication is performed, and the result is pushed.
22	DIV	type	-	The stack is popped twice, the division is performed, and the result is pushed. Attempted division by 0 causes an error stop.

NOTE: For real multiplication and division, the result is tested for UNDEFINED (overflow or underflow). If it is undefined, a 0 is pushed.

Opcode	Mnemonic	Op1	Op2	Description
23	MOD	-	-	The stack is popped twice, the mod is performed, and the result is pushed. Attempted division by 0 causes an error stop.

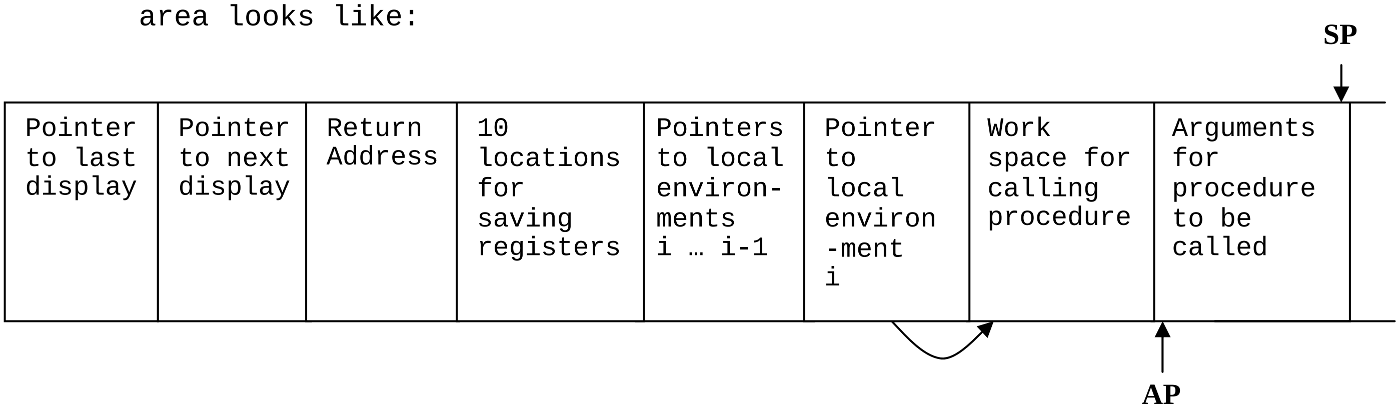
Opcode	Mnemonic	Op1	Op2	Description
24	OR	-	-	The stack is popped twice, the or is performed, and the result is pushed.
25	AND	-	-	The stack is popped twice, the and is performed, and the result is pushed.
26	NOT	-	-	The Boolean value on the top of the stack is complemented.
27	TRUNC	S	-	Depending on S, the top element or the next-to-the-top element on the stack is truncated. The operand must be a real; the result is integer.
28	ROUND	S	-	Depending on S, the top element or the next-to-the-top element on the stack is rounded. The operand must be a real; the result is integer.
29	FLOAT	S	-	Depending on S, the top element or the next-to-the-top element on the stack is floated. The operand must be an integer; the result is real.
30	SHIFT	dir	count	If dir = 0, the integer on the top of the stack is shifted left by count. If dir <> 0, that integer is shifted right by count.
31	GT	-	-	The stack is popped twice, the relation is determined, and the Boolean result is pushed.
32	GE	-	-	The stack is popped twice, the relation is determined, and the Boolean result is pushed.

Opcode	Mnemonic	Op1	Op2	Description
33	LT	-	-	The stack is popped twice, the relation is determined, and the Boolean result is pushed.
34	LE	-	-	The stack is popped twice, the relation is determined, and the Boolean result is pushed.
35	EQ	-	-	The stack is popped twice, the relation is determined, and the Boolean result is pushed.
36	NE	-	-	The stack is popped twice, the relation is determined, and the Boolean result is pushed.
37	B	iloc	-	IPTR is assigned iloc.
38	BCT	iloc	-	The Boolean value on top of stack is popped. If it is true, IPTR is assigned iloc.
39	BCF	iloc	-	The Boolean value on top of stack is popped. If it is false, IPTR is assigned iloc.
40	NOOP	-	-	-
41	PACK	-	-	The 10 characters on the top of the stack are popped and an alfa containing them is pushed. The first character of the alfa will be the last character that was popped.
42	UNPACK	-	-	The alfa on the top of the stack is popped and the 10 characters it contains are pushed. The last character pushed is the first character of the alfa. NOTE that PACK and UNPACK are not reversible.

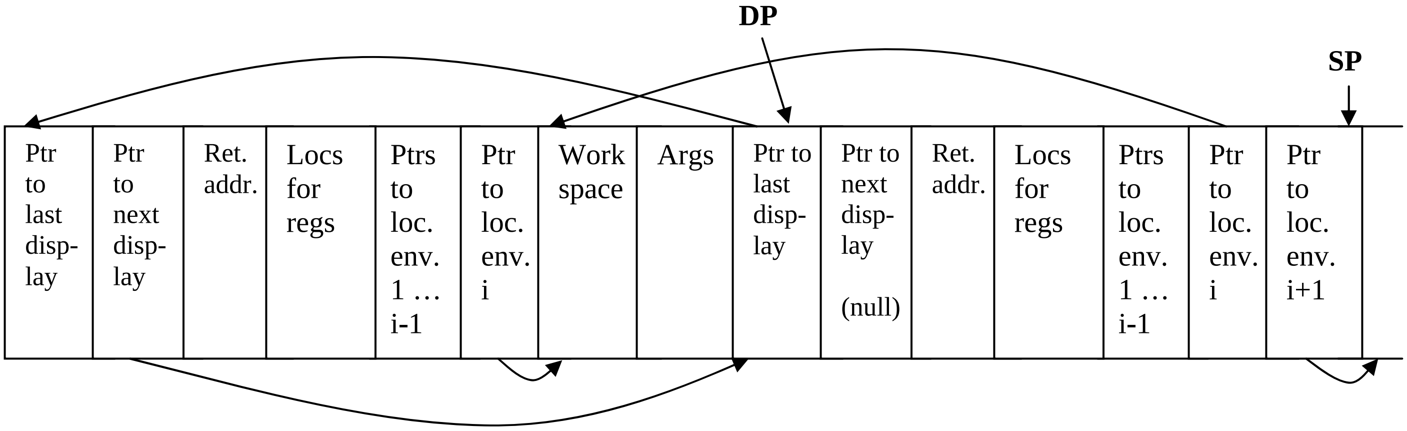
Opcode	Mnemonic	Op1	Op2	Description
43	SYSCALL	i	-	Call system function i. The arguments are on the stack.
44	CALL	i	loc1	See below.
45	RETURN	-	-	See below.

CALL and RETURN INSTRUCTIONS

The CALL instruction is of the form CALL i iloc, where i is the static depth of the procedure (main program has depth 1) and iloc is the address of the first instruction of the called procedure. The caller first pushes the arguments (if any) and then issues the CALL instruction. If, before execution of the CALL instruction, the top portion of the stack area looks like:


SP
Pointer to last display
Pointer to next display
Return Address
10 locations for saving registers
Pointers to local environments i … i-1
Pointer to local environ -ment i
Work space for calling procedure
Arguments for procedure to be called
AP

after the execution of "CALL i iloc" it will look like:



The CALL instruction sets up the links and pointers, but the user must explicitly issue a STOREREGS to store the register values into the locations reserved for them. When the CALL instruction has finished, control has been switched to the called procedure, and it should push its local environment onto the stack.

The RETURN instruction

- pops the stack down to the arguments (DP-1 in the picture)
- resets the forward pointer in the caller's display to 0
- resets DP to point to the caller's display
- resets IPTR to the return address

SYSTEM FUNCTIONS

A system call uses the instruction SYSCALL i. There are 24 system functions. If i is not in range, an error stop occurs. Arguments to system functions are pushed on the stack. The arguments are popped by the function, and any result is left on the top of the stack. For I/O functions, fname must be an alfa which holds the name of the file being accessed.

Note that a file can be used for both input and output. If this is done, the status for OPEN would depend on whether a read or write is done first. Attempting to write on or read from an unopened file causes an error stop, as does executing an input function on fname when EOF(fname) is true.

The file name OUTPUT is reserved by HYPOMAC for its own use. An attempt to open OUTPUT causes an error stop.

When an area of ISPACE is dumped (using the DUMPI system function), only those instructions with non-zero opcodes are written. If all the instructions in the area have opcodes of 0, then only a heading is written. When an area of DSPACE or the stack is dumped (using the DUMPD or DUMPS system function), only defined values are printed. If the whole area is undefined, again, only a heading is printed.

Number	Name	Top-of Stack	Next-to the-top	Description
1	OPEN	status	fname	Opens file fname. If status = 0, fname is set up as an input file. If status <> 0, fname is set up as an output file.
2	CLOSE	fname		Closes fname
3	GET	fname		Advances fname^, assigns value to fname^
4	PUT	fname		Appends fname^ to file
5	READLN	fname		Executes READLN(fname).
6	WRITELN	fname		Executes WRITELN(fname).
7	READBUFF	fname		Pushes fname^.
8	WRITEBUFF	char	fname	Assigns char to fname^.
9	READINT	fname		Reads the next integer on fname and pushes it. If the next thing on fname is not a number, an error stop occurs.
10	READREAL	fname		Reads the next real on fname and pushes it. If the next thing on fname is not a number, an error stop occurs.
11	WRITEINT	numb	fname	Writes numb on fname.
12	WRITEREAL	numb	fname	Writes numb on fname.
13	CLOCK			Pushes the integer "0" (clock function disabled)
14	EOLN	fname		if EOLN(fname) is true, it pushes TRUE, otherwise it pushes FALSE.
15	EOF	fname		if EOF(fname) is true, it pushes TRUE, otherwise it pushes FALSE.

		Top-of	Next-to	
Number	**Name**	**Stack**	**the-top**	**Description**
16	DUMPIT			Dumps ISPACE, DSPACE, and the registers.
17	CHR	numb		Pushes the character with ordinal value numb. If numb < 0 or numb > 127, an error stop occurs.
18	ORD	char		Pushes the ordinal value of char.
19	SQR	numb		Pushes numb * numb. Numb must be an integer or real value.
20	DUMPR	low	high	Dumps register(i), i = low to high.
21	DUMPI	low	high	Dumps ISPACE(i), i = low to high.
22	DUMPD	low	high	Dumps DSPACE(i), i = low to high.
23	LINELIMIT	numb	fname	Sets the LINELIMIT of fname to numb. The default is infinity.
24	INSTLIM	numb		Sets Instruction Limit to numb. The default is infinity.

ERROR DUMP

If an error stop occurs, HYPOMAC prints a message and a dump, and then halts. The contents of the dump is:

- Current value of IPTR.
- Current contents of the registers.
- Contents of the top 10 positions on the stack.
- The last 10 instructions, executed, starting with the least recent one.
- Display dump - traces back through at most 10 displays, dumping out DP-5 ... DP+5 for each one.

If an error occurs during the load, or during execution of the first instruction, the entire defined contents of ISPACE and DSPACE is dumped, along with the contents of the registers. Initializing IPTR to an illegal value is probably the simplest way to dump the entire initial configuration of memory

TERMINATION INFORMATION

If normal termination occurs, HYPOMAC will print out a message to that effect as well as some statistics. These are:

- Number of direct addresses calculated.
- Number of indirect addresses calculated.
- Number of instructions executed.

NOTES ON IMPLEMENTATION AND USE OF HYPOMAC

Loader

HYPOMAC contains a simple loader. It loads the file LOADFILE defined using the following declarations:

```
TYPE
   TYPS         = (INT,RE,CH,ALF,BOOL);
   SPACES       = (ISPCE,DSPCE,RGS);

   VARVAL       = RECORD
                    CASE TYP : TYPS OF
                       INT  : (INTVAL  : INTEGER);
                       RE   : (REALVAL : REAL   );
                       CH   : (CHARVAL : CHAR   );
                       ALF  : (ALFVAL  : ALFA   );
                       BOOL : (BOOLVAL : BOOLEAN)
                       END;

   INSTRUCTION  = RECORD
                       OPCODE : INTEGER;
                       OPER1  : VARVAL;
                       OPER2  : INTEGER
                       END;

   REGISTER     = RECORD
                       REGNUM : INTEGER;
                       REGVAL : INTEGER
                       END;

   LOADFILE     = FILE OF RECORD
                       LOC : INTEGER;
                       CASE SPCE : SPACES OF
                          ISPCE  : (IVALUE : INSTRUCTION);
                          DSPCE  : (DVALUE : VARVAL     );
                          RGS    : (RVALUE : REGISTER   )
                          END;
```

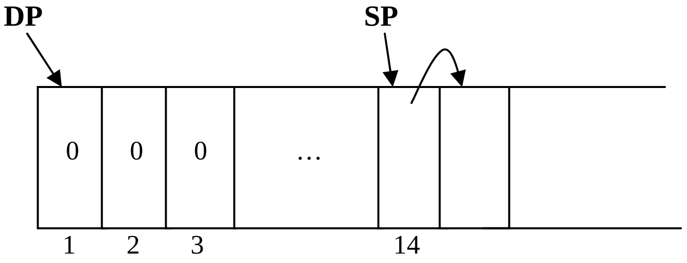
LOC is an index into ISPACE or DSPACE; it is not used for registers.

RGS is used to initialize registers. For this purpose, the loader considers IPTR (the instruction pointer) to be register 10. In general, IPTR cannot be accessed as a register.

LOC must be in range. Illegal opcodes are flagged and replaced with a NOOP instruction.

Initial Layout

We suggest that the initial memory layout look like this:



Use of Hypomac

HYPOMAC is an executable module provided on the class website. Hypomac reads the load files from logical file name "loadfile" and writes error messages and dump to STDOUT.

As part of the operating system, HYPOMAC contains 6 extra files for each job. An OPEN causes one of the files to be allocated to the user and given the name specified by the OPEN command. A CLOSE returns the specified file to the operating system. That file can then be reallocated and renamed.

When the user finishes executing on HYPOMAC, files which have been OPENed but not CLOSEd are still accessible under the names specified by their OPEN commands. The user can then use UNIX file manipulation commands to copy or save these files.