This describes the aspects of the assembler that are not directly related to the hardware. That is, the assembly directives, formats for constants, and assembly options.

Assembly directives all begin with a dot so that the will not be mistaken for instruction codes. Some of them do result in data being added to the executable or object file, so care must be taken to ensure that data is not accidentally executed. For example, the `.DATA` directive deposits numeric values into the executable file.

This assembly code		is translated as (hexadecimal)
`LOAD`	`R1, 65`	`02100041`
`LOAD`	`R2, 3`	`02200003`
`.DATA`	`0x220005`	`02200005`
`.DATA`	`33`	`00000021`
`ADD`	`R1, R2`	`0C120000`

Clearly the first item of data is indistinguishable from an executable instruction.

The assembler maintains a *Location Counter*, which is initially zero, and incremented for each word deposited in the executable file.

Instructions.

An instruction must begin with one of the recognised op-codes:

`ADD`	`AND`	`ATAS`	`BREAK`	`CALL`	`CBIT`	`COMP`	`COMPZ`
`DEC`	`DIV`	`FADD`	`FCOMP`	`FDIV`	`FEXP`	`FFO`	`FIX`
`FLAGSJ`	`FLOAT`	`FLOG`	`FLZ`	`FMUL`	`FRND`	`FSUB`	`GETFL`
`GETSR`	`HALT`	`INC`	`IRET`	`JCOND`	`JNEG`	`JPOS`	`JUMP`
`JZER`	`LDCH`	`LOAD`	`LOADH`	`MOD`	`MOVE`	`MUL`	`NOT`
`OR`	`PAUSE`	`PERI`	`PHLOAD`	`PHSTORE`	`POP`	`PUSH`	`RAND`
`RDIV`	`RET`	`RMOD`	`RSUB`	`SBIT`	`SETFL`	`SETSR`	`SHL`
`SHR`	`STCH`	`STORE`	`SUB`	`SYSCALL`	`TBIT`	`VTRAN`	`WAIT`
`XOR`							

After the op-code, and separated from it by at least one space, appears the name of the primary register, which may be one of `R0`, `R1`, `R2`, `R3`, `R4`, `R5`, `R6`, `R7`, `R8`, `R9`, `R10`, `R11`, `R12`, `SP`, `FP`, `PC`. If an operand is to follow, it must be separated from the primary register by a comma.

Exception 1: The `JCOND` instruction has a condition code instead of a primary register. The condition code may be one of `EQL`, `NEQ`, `LSS`, `LEQ`, `GTR`, `GEQ`, `ERR`.

Exception 2: The following eleven instructions accept neither a primary register nor a condition code: `HALT`, `INC`, `DEC`, `COMPZ`, `JUMP`, `PUSH`, `POP`, `CALL`, `RET`, `IRET`, `SYSCALL`.

Next appears the operand, if one is needed. The operand may be surrounded by square brackets to indicate that the indirect bit is to be set (for a memory access). The rest of the operand must be in one of these four forms: `REG`, `NUM`, `LAB`, `REG+NUM`, `REG-NUM`. Here, `REG` may be any of the 16 listed primary registers except `R0`, `LAB` represents a label as defined using the `name:` notation or by the `.IMPORT` directive, and `NUM` may be a named constant as defined using the `name=` notation, an integer constant, a floating point constant, a character constant, or a value produced by the `#` notation. All of these options are described below.

Exception 1: The `GETFL` and `SETFL` instructions take a flag name as their operand. A flag name is one of `$R`, `$Z`, `$N`, `$ERR`, `$SYS`, `$IP`, `$VM`.

Exception 2: The GETSR and SETSR instructions take a special register name as their operand. The names of the special registers are $FLAGS, $PDBR, $INTVEC, $CGBR, $CGLEN, $DEBUG, $TIMER, $SYSSP, $SYSFP.

Predefined Constants.

The assembler predefines two groups of named constants. One group gives names to interrupt numbers and positions within the interrupt vector, the other gives names to the commands used in the PERI instruction's control blocks.

Interrupts

```
IV$NONE      IV$MEMORY      IV$PAGEFAULT   IV$UNIMPOP
IV$HALT      IV$DIVZERO     IV$UNWROP      IV$TIMER
IV$PRIVOP    IV$KEYBD       IV$BADCALL     IV$PAGEPRIV
IV$DEBUG     IV$INTRFAULT
```

PERI Commands

```
$READDISC    $WRITEDISC     $SIZEDISC      $TERMINC
$TERMINW     $TERMOUTC      $TERMOUTW      $SECONDS
$DATETIME    $MTLOAD        $MTUNLOAD      $MTREAD
$MTWRITE
```

Directives.

name:

Defines name as a label associated with the current value of the location counter. The value is always considered to be relative: all references to this label will be converted by the assembler to program counter relative form PC+n, where n is the value associated with the name minus the value of the location counter where the reference appears minus 1.

name = value

Defines name as a constant equal to the given value. The value is always considered to be absolute: the assembler all references to the name with the value.

.MAKEEXE

Demands one-step assembly and linking. The default behaviour of the assembler is to create an object file which may later be linked with other object files to produce an

executable. If .MAKEEXE appears, the assembler directly produces an executable file, and no object file is made.

.DATA a, b, c, d...

The values a, b, c, d are deposited in successive locations in the executable file. There may be any number of values, and they may be any constants.

.SPACE n

N zeros are deposited in successive locations in the executable file. N may be any constant, but beware of excessively sized executable files

.STRING "text"

The text characters are converted to their ASCII codes, and compressed four per word, in successive locations in the executable file. An extra zero character appears at the end of the string. This produces strings compatible with the STCH and LDCH instructions.

The conversion of characters to ASCII codes treats the character \ specially, as described under “Escape Characters” below.

.ALIGN n

Enough zeros are deposited in successive locations in the executable file to ensure that the location counter is an exact multiple of N. If the location counter is already a multiple of N, there is no effect.

.INCLUDE "filename"

The contents of the indicated file are processed and assembled as though they appeared at this location in the current file. Except that the included file must not result in anything being added to the executable file. Included files are intended to contain constant definitions and similar directives. They may not include any instructions or the .DATA, .SPACE, .STRING, or .ALIGN directives. If no extension is given with the filename, “.ash” is assumed.

.IMPORT name

Only available when an object file is being produced - incompatible with .MAKEEXE.
The name is to be treated as a normal label, but will not be defined within this file. All references to the name produce a special entry in the object file that causes the linker to substitute the correct value supplied by an .EXPORT directive in another object file.

.EXPORT name

Only available when an object file is being produced - incompatible with .MAKEEXE.
The name must be a normal label that is defined at some point in the current file. An entry recording the correct address for this label is added to the object file. This means that code in other files can use this label when the two object files are linked into a single executable.

.LINK "filename"

Only available when an object file is being produced - incompatible with .MAKEEXE.
The file named must be an object file also produced by the assembler.
A command is inserted into the object file instructing the linker to combine the contents of the given file with this one when an executable is produced.

Names, Lines, Spaces, and Comments.

Spaces are not permitted within any lexeme (the symbols of the language), except when within quotes. Spaces may always be used to separate lexemes, but are not always necessary.

Blank lines are always permitted.

Any characters following the // notation up to the end of the line are completely ignored, except when the // appears in quotes of course.

There may only be one instruction or directive per line, and no instruction or directive may spread over more than one line.

Names used for labels and constants must begin with one of the following characters

```
A-Z  a-z  _  %  $  @
```

and must consist of only the characters

```
A-Z  a-z  0-9  _  %  .  $  @  #
```

Case is not significant.

Constants.

Integer Constants

Integer constants must begin with one of the ten decimal digits 0-9. There is no limit to the size accepted by the assembler, but only 32 bits are used, or in the case of instruction operands only 16 bits.

If the first two characters are 0x or 0X, the number is treated as hexadecimal, base 16. The rest of the number may be any combination of the digits 0-9, A-F, a-f.

If the first two characters are 0o or 0O, the number is treated as octal, base 8. The rest of the number may be any combination of the digits 0-7.

If the first two characters are 0b or 0B, the number is treated as binary, base 2. The rest of the number may be any combination of the digits 0 or 1.

If the first two characters are `0d` or `0D`, the number is treated as decimal, base 10. The rest of the number may be any combination of the digits `0-9`.

In all other cases, the number is treated as decimal, base 10, and may consist only of the digits `0-9`.

If the letter `H` or `h` is appended to the number (with no intervening spaces) then only the 16 most significant bits are used. If the letter `L` or `l` is appended then only the least significant 16 bits are used. This is compatible with the operation of the `LOADH` instruction.

Floating Point Constants

Floating point constants must begin with at least one decimal digit followed by a decimal point. After the decimal point there may be any number (including zero) of decimal digits.

The decimal number may be followed by an optional "times ten to the power of" sequence which consists of the letter `E` or `e`, an optional `+` or `-` sign, and finally at least one decimal digit.

If the letter `H` or `h` is appended to the number (with no intervening spaces) then only the 16 most significant bits are used. If the letter `L` or `l` is appended then only the least significant 16 bits are used. This is compatible with the operation of the `LOADH` instruction.

Character Constants

A single character within single quotes, e.g. `'x'`, is a numeric constant equal to the ASCII code for the quoted character. If multiple characters appear, they are processed in turn from left to right: as each character appears, the numeric value so far is shifted 8 bits to the left, and the ASCII value of the new character is added.

The conversion of characters to ASCII codes treats the character `\` specially, as described under "Escape Characters" below.

Numeric Values of Symbols.

The numeric part of the value of any symbol may be accessed by adding a `#` character before its name. For example

```
LOAD  R1, #ADD
LOAD  R2, #FP
```

would load register 1 with 6 and register 2 with 14, because the operation code for the `ADD` instruction is 6, and `FP` is register 14.

Escape Characters.

Within quotes, 'single' or "double", the \ character modifies the meaning of the next character to appear. The two character sequence of \ followed by another character is always treated by the assembler as a single character.

- `\'` represents ', but is not recognised as a closing quote; it may appear inside strings.
- `\"` represents ", but is not recognised as a closing quote; it may appear inside strings.
- `\\` represents \, but does not modify the meaning of the next character.
- `\0` represents the ASCII code 0.
- `\n` represents the newline character, ASCII code 10.
- `\t` represents the tab character, ASCII code 9.
- `\b` represents the backspace character, ASCII code 8.
- `\r` represents the carriage-return character, ASCII code 13.

All other characters represent themselves, so for example:

- `\x` represents `x` itself.

Command-Line Options.

When running the assembler, certain options may appear on the command line along with the name of the file to be assembled.

- `-e` commands one-step assembly and linking. It is equivalent to having the directive `.MAKEEXE` appear within the program.
- `-l` commands that a listing be produced, showing first the names and values of all defined labels, then each line of the assembly file (with its line number) followed by the exact translation shown in hexadecimal
- `-lb` exactly the same as `-l`, except that values are shown in both hexadecimal and binary.

If the name of the assembly program file ends in "`.ass`", then the "`.ass`" does not need to be typed - "`assemble abc`" is automatically converted to "`assemble abc.ass`".

If an object file is produced, it will have the same name as the assembly file, but with the "`.ass`" (or other extension) replaced by "`.obj`".

If an executable file is produced instead, it will have the same name as the assembly file, but with the "`.ass`" (or other extension) replaced by "`.exe`".