Table of Cont

1 Machine Language Compiler

1.1 Before you start

This section will talk about the MASD syntax as used with tl
from the HPTools v3.0.

There are some differences between the MASD syntax us
the one used with the HPTools.

The main limitations are the expressions that are compiled c
the MK v2.0) and operations can't be performed on two exte

1.2 Generalities on ML (Machine Language)

As the Saturn processor directly executes ML, the operati
what a ML program is doing.

On the HP 48 calculator, user data are stored in the sam
When there is a bug in a ML program, you have best chanc
very careful when programming in ML.

ML is a processor dependent language, so what you will lea
not be useful on other processor. On the other hand, the pr
will acquire are not dependent of the hardware and then will

1.3 Generalities on Masd Syntax

Masd expects a character string (called source) on the top c

A source is a set of instructions, comments, and separatior
a carriage return and an arobas @.

Masd is case sensitive, so be careful, as « b o u c l e » a
different labels.

Separation characters are those with an ASCII number
spaces, tabs, line feed and carriage return.

Some instructions need a parameter, called field. Separati
instruction and the field, are spaces, tabs, and points. There
instead of A+B A.

Comments can be placed everywhere between two instru
or ; and finish at the end of the current line.

1.4 Using labels

A label is a marker in the program. The principal use of la
destinations.

A label is a set of less than 128 characters different from s
label begins with a star '*' and ends with a separation chara

*B i g L o o p is the BigLoop label declaration.

Be careful about upper and lower cases!

f labels can be used:

l is a label that can be used everywhere in the project, like global ascal or C.

a label that is only accessible in a local section like local variables in

ı starts at the beginning of a source, after a global label, after a link (see

ı finishes at the end of a source, before a link, before a global label.

identified by a ‘ . ’ as the first character.

a label that exists only in the link where it is declared, like a private ct Pascal.

identified by a ‘_’ as the first character.

ects, using less global labels is better because a global label is longer to ecause it gives a better program structure. A good habit is to use global the program in subroutines, and to use local labels inside these

ants

o define constants. It is useful to identify a memory address by a name, address itself.

instead of typing D1=80100 every time it is needed, it is better to Result 80100 at the beginning of the project and then to type ult when needed.

aration:

me Expression or

ame Expression or

tName Expression

e CstValue or EQU *CstName CstValue*

hexadecimal number. A decimal number can be typed with a leading #

is same as DC Foo #16

stant cannot be given the same name as a declared label.

ame of a constant follows the same rules as the name of a label.

es a ‘programming register’ called CP (Constant Pointer) which helps to nts. CP is defined by:

PE=*Expression*

on 5 nibbles, its initial value is 80100.

DCCP *Increment ConstantName*

declares a constant with the current CP value and then incre

Note: Increment is a hexadecimal value, to use a decimal va

For example, if CP equals to $10

```
DCCP 5 Foo
```

defines a Foo constant with a value of $10 and then change

Several constants can be defined, starting from CP.

: *Inc CstName0 CstName1 ... CstNameN-1* :

defines *N* constants CstName*x* with a value of CP+*x*Inc* value to CP+*N*Inc*.

Warning: By default, *Inc* is an hexadecimal number. Meta decimal.

1.6 Expressions

An expression is a mathematical operation that is calculated

Terms of this operation are hexadecimal or decimal values,

An expression stops on a separation character.

```
DCCP 5 @Data
...
D1=(5)@Data+$10/#2 D0=(5)$5+DUP
```

are correct expressions.

Notes:

- A hexadecimal value must begin with a $ and a decimal valu
- There are no priorities (precedences) in operations. $1+$2* parenthesis to set precedences.
- You can't use more than three level of parenthesis.
- Calculations are done on 32 bits.

1.7 Skips

Skips are a first step from ML to a third generation langu another way to write SATURN instructions.

The basement of Skips is the Block structure.

A block is enclosed in { and }, and can be inside another b

The following instructions deal with blocks.

SKIPS instructions	Equivalents
SKIP { ... }	GOTO .S ... *.S
SKIPL { ... }	GOTOL .S ... *.S
SKIPC { ... }	GOC .S ... *.S
SKC { ... }	GOC .S ... *.S
SKIPNC { ... }	GONC .S ... *.S
SKNC { ... }	GONC .S ... *.S

	Test GOYES .S ... *.S
	Test GOYES .S ... *.S
	/Test GOYES .S ... *.S
	GOSUB .S ... *.S
	GOSUBL .S ... *.S
	Defines a block (generates no code)
	$/02A2C GOIN5 *.S ... *.S (to create a character string)
	$/02DCC GOIN5 *.S ... *.S (to create a code object)
..	$/*PROLOG* GOIN5 .S ... *.S (to create a 'prolog – length' object)

pposite of Test. For example if Test is ?A<C.A, /Test is ?A>=C.A. structions dealing with the hardware register (?HST=0, ?MP=0, =0 and ?SB=0) cannot be inversed.

ıre defined, special instructions can be used in them. These instructions nd UP allow to jump to the end or to the beginning of a block.

ctions	are equivalent to
EXIT	*.Beginning GOTO.End GOC.End GONC.End ?A=0.A ¨.End GOTO.Beginning GOC.Beginning GONC.Beginning
UP	?A=0.A ¨.Beginning *.End

make confusion between EXIT and UP instructions, which are GOTOs, UP after a test, which are GOYES's.

can jump to the beginning or to the end of an upper-level block by number of blocks to exit, after the UP or EXIT instructions.

ctions	Are equivalent to
	*.Beg3 *.Beg2 *.Beg1
2	GOTO.Beg2
3	GOTO.Beg3
IT1	GOTO.End1
IT3	GOTO.End3
	*.End1 *.End2 *.End3

s equivalent to EXIT, and UP1 is equivalent to UP.

LSE, SKEC or SKENC instructions, two blocks create an IFNOT-structure.

Are equivalent to	Or in high-level language
?A=0.A GOYES.Beg2 *.Beg1	IF NOT A=0 THEN BEGIN

```
   EXIT
   UP
}
SKELSE
{
   A+1.A
   EXIT
   UP
}
```

```
   GOTO.End2 % and not End1
   GOTO.Beg1
*.End1
GOTO.End2
*.Beg2
   A+1.A
   GOTO.End2
   GOTO.Beg2
*.End2
```

```
   EN
ELSE
   BE
   EN
```

1.8 SATURN instructions syntax

In this section:

x is an integer number between 1 and 16.

h is a hexadecimal digit.

a is a 1 to 16 or a 0 to 15 number depending of the current r

f is a field A, B, X, XS, P, WP, M or S.

Reg is a working register A, B, C or D.

SReg is a save register R0, R1, R2, R3 or R4.

Exp is an expression.

Cst is a constant. The value is given in hexadecimal or deci respectively.

DReg is a pointer register D0 or D1.

Data is memory data pointed by D0 or D1. It means DAT0 c

Note: For instructions that use two working registers, only and A-C are available.

For instructions like *Reg1=Reg1…* you can write only *Reg1* the same as A+C.A.

1.8.1 Assigning 0 to a register

Syntax: *Reg*=0.*f*

Example: A=0.M

1.8.2 Loading a value in A or C

LC and LA instructions allow to load a constant value into A

LC *hhh...hh* loads x nibbles into C.

LA *hhh...hh* loads x nibbles into A.

Example: LC 80100

Note: LC #12 allow to load 12 decimal into the 3 first nib nibbles used is the number of characters necessary to wri #). So #12 will take three nibbles.

LCASC(*x*) *Characters* loads the hexadecimal value must be between 1 and 8. LAASC(*x*) if the counterpart for

Example: LCASC(7) HP_MASD

or LA(x) Exp load the result of an expression into C or A, using x

5)@Buffer+DataOffset

gister value into another register

=*Reg2.f*

B.X

tween two registers

*Reg2*EX.*f*

EX.W

=*Reg1+Reg2.f* or *Reg1+Reg2.f*

C+A.A or C+A.A

and *Reg2* are same, this cause to multiply the register by two.

=Reg1-Reg2.*f* or *Reg1-Reg2.f*

C-B.A or C-B.A

owing instructions are also available:

B=C-B.*f* C=A-C.*f* D=C-D.*f*

nd decrement

Reg+Cst.f or *Reg+Cst.f* *Reg=Reg-Cst.f* or *Reg-Cst.f*

A+10.A or A+10.A A=A-10.A or A-10.A

Saturn processor is not able to add a constant greater than 16 to a *cst* is greater than 16, Masd will generate as many instructions as

if adding constants to a register is very useful, big constants should be use this will slow down execution, and generate a big program.

ng a constant greater than 1 to a P, WP, XS or S field is a bugged ruction (problem with carry propagation). Use these instructions with

adding a constant greater than 16 to a register, the carry should not be

s shifting (divide by 16)

R.*f*

R.W

shifting (multiply by 16)

L.*f*

..W

1.8.10 Right bit shifting (divide by 2)

Syntax: *Reg*SRB.*f*

Example: ASRB . W

1.8.11 Right circular nibble shifting

Syntax: *Reg*SRC.*f*

Example: ASRC . W

1.8.12 Left circular nibble shifting

Syntax: *Reg*SLC.*f*

Example: ASLC . W

1.8.13 Logical AND

Syntax: *Reg1=Reg1&Reg2.f* or *Reg1&Reg2.f*

Example: C=C&B . A or C&B . A

1.8.14 Logical OR

Syntax: *Reg1=Reg1* !*Reg2.f* or *Reg1* !*Reg2.f*

Example: C=C ! B . A or C ! B . A

1.8.15 Logical NOT

Syntax: *Reg1=-Reg1*-1.*f*

Example: C= - C - 1 . A

1.8.16 Mathematical NOT

Syntax: *Reg1=-Reg1.f*

Example: C= - C . A

1.8.17 Loading value into a R Register

Syntax: *RReg=Reg*.f

Example: R0=A . W

Note: *Reg* can only be A or C.

1.8.18 Loading value into A or C from a R register

Syntax: *Reg=RReg*.f

Example: A=R1 . X

Note: *Reg* can only be A or C.

1.8.19 Exchange between A or C and a R register

Syntax: *RegRRegEX*.f

Example: AR1EX . X

Note: *Reg* can only be A or C.

e (POKE)

ctions write the value of A or C at the address pointed to by D0 or D1.

Reg.f	or	*Data=Reg.x*
1=C.A	or	DAT0=A.10

only be A or C.

l (PEEK)

ctions load into A or C the data pointed to by D0 or D1.

Data.f	or	*Reg=Data*.x
DAT1.A	or	A=DAT0.10

only be A or C.

odifications

nd D1

=hh or	*DReg=hhhh*	or	*DReg=hhhhh*	or
=(2)E*xp*	or	*DReg*=(4)*Exp*	or	*DReg*=(5)*Exp*
FF D0=12345		D1=(5)toto+$5		

etween A or C and D0 or D1

r C, field A, into D0 or D1

=Reg

A

only be A or C.

four low nibbles of A or C into D0 or D1

*=Reg*S

AS

only be A or C.

A or C, field A, and D0 or D1.

RegEX

.EX

only be A or C.

the 4 first nibbles of A or C and D0 or D1

RegXS

.XS

only be A or C.

d decrement of D0 and D1

=DReg+Cst	or	*DReg+Cst*
=DReg-Cst	or	*DReg-Cst*
D0+12	D1-50	

Note 1: The Saturn processor is not able to add a cons
register but if *cst* is greater than 16, Masd will generate
needed.

Note 2: Even if adding constants to a register is very usefu
avoided because this will slow down execution, and generate

Note 3: After adding a constant greater than 16, the carry sh

1.8.23 Working registers tests

Notes:

- A test is always followed by RTNYES, GOYES, SKIP
GOTOL, GOVLNG, GOSUB, GOSUBL or GOSBVL.
- RTY is the same as RTNYES.
- An arrow (¨) may be followed by a label name, then repl
followed by a skip block, which is equivalent to the inver
SKIPYES, to reproduce a IF-THEN structure. Example: ?A
as ?A#C . A { }.
- SKIPYES may be omitted if followed by a skip block ({ }).
- If the test if followed by a GOTO, GOTOL, GOVLNG, GOSU
Masd compiles the inverse of the test, to reproduce a GO
Example: ?A=C . A GOTO B is the same as ?A#C . A {
- GOTO, GOTOL, GOVLNG, GOSUB, GOSUBL, GOSBVL or
a HST test.
- A label name must follow a GOYES, GOTO, GOTOL, GOVL
GOSBVL.

a) Equality and inequality tests

Syntax: ?*Reg1=Reg2.f* ?*Reg1#Reg2.f*

Example: ?A=C . B ?C#D . A

Note: The HP inequality character may be used.

b) Lower and greater tests

Syntax: ?*Reg1<Reg2.f* ?*Reg1<=Reg2.f*

Example: ?A<C . B ?C>=D . A

Note: The HP lower or equal and greater or equal characters

c) Nullity tests

Syntax: ?*Reg=0.f* ?*Reg#*0.*f*

Example: ?A=0 . B ?C#0 . XS

Note: The HP inequality character may be used.

1.8.24 Working with some bits of A or C register

*Reg*BIT=*v.a* *?Reg*BIT=*v.a* where Reg is A or C, *v* is
is the bit number.

Examples: ABIT=0 . 5, ?CBIT=1 . 3 GOYES TOTO

1.8.25 Operations on PC

A=PC C=PC PC=A PC=C APCEX CPCEX

ı the Hardware Status Register

SR=0 MP=0 HST=*a*
=0 ?SR=0 ?MP=0 ?HST=*a*

ı P

P=P-1 P-1

.a CPEX.*a*
+P+1

ːtions

l
e l or GOLONG *l a b e l*

t *Cst* is an hexadecimal number.
a b e l *label* is a constant, or a label in absolute mode
COMMAND" *Command* is an entry in the STARTEXT table.

e l
b e l

t *Cst* is a hexadecimal number.
a b e l *label* is a constant, or a label in absolute mode.
COMMAND" *COMMAND* is an entry in the STARTEXT table.

l
e l same as SKIPNC { GOTO label }
be l same as SKIPC { GOTO label }

M RTNCC RTNSC RTI
C
ʁTY after a test.

ıetween C and RSTK

RSTK=C instructions allow to push to or pop data from the Saturn

ːt instructions

T=C, A=IN and C=IN

and C=IN instructions are bugged (they only work on even addresses).
2 and C=IN2, which are ROM calls to A=IN and C=IN instructions.

beginning of ROM is not usable (because it is recovered by RAM), use
ː=IN3, which are calls to A=IN and C=IN instructions in the **Meta**

C=IN is a ROM call that does OUT=C C=IN.

Note 4: OUT=C=IN3 is the same, but in the **Meta Kernel** ROM is recovered).

1.8.31 Processor control instructions

Working mode modification

SETDEC SETHEX

other instructions

UNCNGF CONFIG RESET C=ID

SHUTDN INTON INTOFF RSI

SREQ?

BUSCB BUSCC BUSCD

Appendixes

1 Introduction to assembly lang

1.1 Introduction

Our purpose here is to help you translate an algorithm in S using Masd. You are supposed to already know the progra language.

If you want to learn how to program, there are several book Art of Programming'.

We will try to give you the basis to build little programs. experience, look at other people's programs, and read mor subject.

Many informations here are taken from the excellent Frenc de la HP48' by Paul Courbis (Angkor).

1.2 What is assembly language (asm) ?

Asm is the only language directly readable by the micropro fastest and the most powerful language. Each processor composed of binary numbers interpreted by the processo giving instructions to be processed.

To make it easily readable by humans, each group of nur mnemonic. Assemblers (like Masd) translate these mnem the processor can execute. They sometimes add ma structures to ease the programmer's work.

1.3 The Saturn processor

1.3.1 Generalities

SATURN is the name of a processor made by NEC. It is u bit processor (it can process 4 bits at each clock tick), with 5 64-bit saving registers, 2 20-bit memory pointers, an inter software flags, 4 hardware flags, 1 field register, 1 output re 1 20-bit program counter.

A bit is a binary digit, it can takes only two values 0 or 1.

A nibble is a 4-bit value, from 0 to 15.

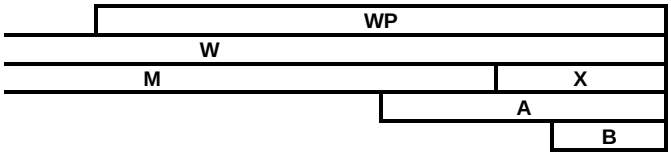
1.3.2 Working and saving registers

These registers are 64-bit (16-nibble) registers divided into f

A field is a part of a register; a group of nibbles. Two fiel have different lengths.

In a Saturn register, there are six fixed fields and two varia register):

15	14	13	12	11	10	9	8	7	6	5	4
		(With P=9)				**P**					



king with the A field only affects the 20 lowest bits of the register.

means register C, field B.

gister which indicates the WP field length, the P field location, and the nibble for the instructions LA and LC.

r

ster is a 16-bit register, composed of 16 flags, which can be only 0 or 1. milar to RPL flags (CF, SF...).

ack is a set of eight 20-bit registers. Only one of them is accessible e RPL stack).

utine is called (with GOSUB), the return address (just after the GOSUB) o this stack. And when a RTN is encountered, the Saturn pops this from the stack and jumps to it.

are only eight levels, building large programs with a lot of subroutine t overload the return stack.

ble to push a value with RSTK=C and to pop a value with C=RSTK.

nters

emory pointers D0, D1 and PC are 20-bit registers, which contains

e used to access memory. You can load any address in them and then

ogram Counter, it holds the address of the next instruction to be s modified at every jump (GOTO, GOSUB...), or directly with instructions .

ing pointers, let's see how memory is seen through HP asm.

an be compared to a ring of cells. Each cell contains one nibble. They from 0 to 524287 (20 bits). It is a ring, so the cell 524287 is followed by number of a cell is called its address.

register that contains an address. On HP48, a pointer is 20 bits (5 Therefore we can access any memory cell with one pointer (Whereas ed two pointers).

c) Memory accesses

On the HP48, memory inputs and outputs are made throu
D0 and D1 indicating which cell will be accessed.

It is possible to access more than one cell at one time, as
nibbles) can be loaded.

A=DAT0 A reads 5 nibbles at the address pointed by D0, t
A field of A (Aa).

DAT1=C 12 writes 12 nibbles at the address pointed by D
12 lowest nibbles of C (nibbles 0 to 11).

Note that the Saturn processor inverses the order of the
Ca contains 84571 and D1 contains 80200, DAT1=C A wil

80200	80201	80202	80203
1	7	5	4

The inversion is done at each read or write, so if you write
address, using the same field, you will retrieve the same dat

1.4 Starting and stopping a Program

Starting to write an asm program looks difficult to begin
comprehension of the RPL system, in order to avoid destroy

To start a program, all the system data must be saved som
they will be read back, after your program has finished, in
operation. These data are contained in the registers Ba, Da,

This saving is done with the Masd instruction SAVE (or the
=SAVPTR). Do it only one time, before doing any other proc

At the end of your program, you have to restore the systen
LOAD (or GOSBVL =GETPTR).

You can then exit to the RPL system with RPL (or
PC=(A)).

These two instructions can be compacted into the single
GOVLNG =GETPTRLOOP).

The shortest program is:

```
"SAVE
% This program does nothing !
LOADRPL
@"
```

1.5 Working with the RPL stack

The RPL stack (the stack displayed on the screen) is store
addresses, each address points to an object in RAM or R
object, its address must be first read.

The stack pointer is in D1, pointing to the address of the fir
to the second level, and so on. A 00000 marks the end of
SAVE saves D1 (and other RPL registers) in reserved RAM
then need to read the RPL stack, do a LOAD before, whi

ą LOAD before exiting from your program, otherwise D1 could point memory, that may crash your HP.

ırogram needs to read the stack, it is recommended to do the argument efore starting the asm code.

ead the value of a system binary on the first stack level, do the following:

% Reads the address of the first level in Ca
. now points on the prolog of this object
ę that D1 is saved in Ca
D1 points on the value of the SB
% Reads the value in Aa
ıstores D1

P, you just have to make the stack start 5 nibbles upper:

The stack starts upper, so lower levels are dropped
Da contains the number of free 5 nibbles-blocks,
o you shall increment it
ed by SAVE and restored by LOAD.

)AD after, the old value of D1 is retrieved. In order to effectively drop an _OAD D1+5 D+1.A at the end of your program (before the RPL do LOAD D1+5 D+1.A SAVE anywhere in your program.

ment on the stack, you just have to create a new stack level. E.g. if the s is in Aa:

s the RPL registers
K I PNC % Tests free memory
C 1 ERREUR C } % Insufficient mem
Adds a stack level
% Puts the object address
es the RPL registers

uction set

· columns in this table.

mn may be empty, or contains a M or a star. M indicates that the ɔde is used (SETDEC and SETHEX). A star indicates that the carry is e instruction.

ɔlumn contains the instruction mnemonic.

mn indicates the fields which can be used with the instruction. f is any any field but A, d is a nibble value (0-15 or 1-16), x is a nibble value (0-

ın is the execution length in Saturn cycles. n is the number of nibbles in . If the time is not integer, take IP(time) if the instruction is on an even (time)+1 if the instruction is on an odd address. For tests instructions, times separated by a slash, the first is for a true test, the second for a memory instructions, the two times indicate the instruction length and ɔeration length.

.en from the book 'Voyage au centre de la HP48 GX' by Paul Courbis, ns.

*	?A#0	A	21.5/13.5
*	?A#0	a	16.5/8.5+q
*	?A#C	A	21.5/13.5
*	?A#C	a	16.5/8.5+q
*	?A<=B	A	21.5/13.5
*	?A<=B	b	16.5/8.5+q
*	?A<B	A	21.5/13.5
*	?A<B	b	16.5/8.5+q
*	?A=0	A	21.5/13.5
*	?A=0	a	16.5/8.5+q
*	?A=C	A	21.5/13.5
*	?A=C	a	16.5/8.5+q
*	?A>=B	A	21.5/13.5
*	?A>=B	b	16.5/8.5+q
*	?A>B	A	21.5/13.5
*	?A>B	b	16.5/8.5+q
*	?ABIT=0	d	20.5/12.5
*	?ABIT=1	d	20.5/12.5
*	?B#0	A	21.5/13.5
*	?B#0	a	16.5/8.5+q
*	?B#A	A	21.5/13.5
*	?B#A	a	16.5/8.5+q
*	?B<=C	A	21.5/13.5
*	?B<=C	b	16.5/8.5+q
*	?B<C	A	21.5/13.5
*	?B<C	b	16.5/8.5+q
*	?B=0	A	21.5/13.5
*	?B=0	a	16.5/8.5+q
*	?B=A	A	21.5/13.5
*	?B=A	a	16.5/8.5+q
*	?B>=C	A	21.5/13.5
*	?B>=C	b	16.5/8.5+q
*	?B>C	A	21.5/13.5
*	?B>C	b	16.5/8.5+q
*	?C#0	A	21.5/13.5
*	?C#0	a	16.5/8.5+q
*	?C#B	A	21.5/13.5
*	?C#B	a	16.5/8.5+q
*	?C#D	A	21.5/13.5
*	?C#D	a	16.5/8.5+q
*	?C<=A	A	21.5/13.5
*	?C<=A	b	16.5/8.5+q
*	?C<A	A	21.5/13.5
*	?C<A	b	16.5/8.5+q
*	?C=0	A	21.5/13.5
*	?C=0	a	16.5/8.5+q
*	?C=B	A	21.5/13.5
*	?C=B	a	16.5/8.5+q
*	?C=D	A	21.5/13.5
*	?C=D	a	16.5/8.5+q
*	?C>=A	A	21.5/13.5
*	?C>=A	b	16.5/8.5+q
*	?C>A	A	21.5/13.5
*	?C>A	b	16.5/8.5+q
*	?CBIT=0	d	20.5/12.5
*	?CBIT=1	d	20.5/12.5
*	?D#0	A	21.5/13.5
*	?D#0	a	16.5/8.5+q
*	?D<=C	A	21.5/13.5
*	?D<=C	b	16.5/8.5+q
*	?D<C	A	21.5/13.5
*	?D<C	b	16.5/8.5+q
*	?D=0	A	21.5/13.5
*	?D=0	a	16.5/8.5+q

)1.5/13.5
.6.5/8.5+q
)1.5/13.5
.6.5/8.5+q
.5.5/7.5
.5.5/7.5
.5.5/7.5
.5.5/7.5
.5.5/7.5
.5.5/7.5
.6.5/8.5
.6.5/8.5
.5.5/7.5
1.5+q
}
1.5+q
}
1.5+q
}
}+q
}+q
}+q
}+q
1.5+q
}
1.5+q
}
1.5+q
}
1.5+q
}
}+q
1.5+q
}
1.5+q
}
1.5+q
}
}+q
1.5+q
}
1.5+q
}
1.5+q
}
2.5,3.5
)0+p,1+q/2
.9.5
.9+q,1+q/2
)3.5,3.5
)0+q,1+q/2
.9.5
.9+q,1+q/2
3.5
.1
)+q
)0.5
)+q
)0.5
)+q
)0.5
)+q
)0.5
)+q
)0.5

	ABEX	a	4.5+q
	ABEX	A	8
	ABIT=0	d	7.5
	ABIT=1	d	7.5
	ACEX	a	4.5+q
	ACEX	A	8
	AD0EX		9.5
	AD0XS		8.5
	AD1EX		9.5
	AD1XS		8.5
	APCEX		19
	AR0EX	f	9+q
	AR0EX	W	20.5
	AR1EX	f	9+q
	AR1EX	W	20.5
	AR2EX	f	9+q
	AR2EX	W	20.5
	AR3EX	f	9+q
	AR3EX	W	20.5
	AR4EX	f	9+q
	AR4EX	W	20.5
*	ASL	a	4.5+q
*	ASL	A	8
	ASLC		22.5
	ASR	a	4.5+q
	ASR	A	8
	ASRB	f	8.5+q
	ASRB	W	21.5
	ASRC		22.5
*M	B=-B	a	4.5+q
*M	B=-B	A	8
*M	B=-B-1	a	4.5+q
*M	B=-B-1	A	8
	B=0	a	4.5+q
	B=0	A	8
	B=A	a	4.5+q
	B=A	A	8
	B=B!A	f	6+q
	B=B!C	f	6+q
	B=B&A	f	6+q
	B=B&C	f	6+q
*M	B=B+1	a	4.5+q
*M	B=B+1	A	8
*M	B=B+A	a	4.5+q
*M	B=B+A	A	8
*M	B=B+B	a	4.5+q
*M	B=B+B	A	8
*M	B=B+C	a	4.5+q
*M	B=B+C	A	8
*	B=B+x+1	f	8+q
*M	B=B-1	a	4.5+q
*M	B=B-1	A	8
*M	B=B-A	a	4.5+q
*M	B=B-A	A	8
*M	B=B-C	a	4.5+q
*M	B=B-C	A	8
*	B=B-x-1	f	8+q
	B=C	a	4.5+q
	B=C	A	8
*M	B=C-B	a	4.5+q
*M	B=C-B	A	8
	BCEX	a	4.5+q
	BCEX	A	8
*	BSL	a	4.5+q
*	BSL	A	8

)2.5
).5+q
)
).5+q
)1.5
)2.5
.0
).5
.0
).5
).5+q
)
).5+q
)
).5+q
)
).5+q
)
).5+q
)
).5+q
)
).5+q
)
)+q
)+q
)+q
)+q
)+q
)+q
).5+q
)
).5+q
)
).5+q
)
).5+q
)
).5+q
)
).5+q
)
)+q
).5+q
)
).5+q
)
).5+q
)
).5+q
)
)+q
).5+q
)
)3.5,3.5
)0+q,1+q/2
).9.5
).9+q,1+q/2
)3.5,3.5
)0+q,1+q/2
).9.5
).9+q,1+q/2
).3.5
).5
)
).1
)+q
)0.5
)+q

	C=R1	W	20.5
	C=R2	f	9+q
	C=R2	W	20.5
	C=R3	f	9+q
	C=R3	W	20.5
	C=R4	f	9+q
	C=R4	W	20.5
	C=RSTK		9
	C=ST		7
	CBIT=0	d	7.5
	CBIT=1	d	7.5
	CD0EX		9.5
	CD0XS		8.5
	CD1EX		9.5
	CD1XS		8.5
	CDEX	a	4.5+q
	CDEX	A	8
	CETDEC		4
	CETHEX		4
	CLRHST		4.5
	CLRST		7
	CONFIG		13.5
	CPCEX		19
	CPEX	x	8
	CR0EX	f	9+q
	CR0EX	W	20.5
	CR1EX	f	9+q
	CR1EX	W	20.5
	CR2EX	f	9+q
	CR2EX	W	20.5
	CR3EX	f	9+q
	CR3EX	W	20.5
	CR4EX	f	9+q
	CR4EX	W	20.5
*	CSL	a	4.5+q
*	CSL	A	8
	CSLC		22.5
	CSR	a	4.5+q
	CSR	A	8
	CSRB	f	8.5+q
	CSRB	W	21.5
	CSRC		22.5
	CSTEX		7
	D=0	a	4.5+q
	D=0	A	8
	D=C	a	4.5+q
	D=C	A	8
*M	D=C-A	a	4.5+q
*M	D=C-A	A	8
*M	D=-D	a	4.5+q
*M	D=-D	A	8
*M	D=-D-1	a	4.5+q
*M	D=-D-1	A	8
	D=D!C	f	6+q
	D=D&C	f	6+q
*M	D=D+1	a	4.5+q
*M	D=D+1	A	8
*M	D=D+C	a	4.5+q
*M	D=D+C	A	8
*M	D=D+D	a	4.5+q
*M	D=D+D	A	8
*	D=D+x+1	f	8+q
*M	D=D-1	a	4.5+q
*M	D=D-1	A	8
*M	D=D-C	a	4.5+q

	PC=(C)		26,3.5
	PC=A		19
	PC=C		19
	R0=A	f	9+q
	R0=A	W	20.5
	R0=C	f	9+q
	R0=C	W	20.5
	R1=A	f	9+q
	R1=A	W	20.5
	R1=C	f	9+q
	R1=C	W	20.5
	R2=A	f	9+q
	R2=A	W	20.5
	R2=C	f	9+q
	R2=C	W	20.5
	R3=A	f	9+q
	R3=A	W	20.5
	R3=C	f	9+q
	R3=C	W	20.5
	R4=A	f	9+q
	R4=A	W	20.5
	R4=C	f	9+q
	R4=C	W	20.5
	RESET		7.5
	RSI		8.5
	RSTK=C		9
	RTI		11
	RTN		11
	RTNC		12.5/4.5
*	RTNCC		11
	RTNNC		12.5/4.5
*	RTNSC		11
	RTNSXM		11
	SB=0		4.5
	SHUTDN		6.5
	SR=0		4.5
	SREQ?		9.5
	ST=0	d	5.5
	ST=1	d	5.5
	ST=C		7
	UNCNFG		14.5
	XM=0		4.5

Index