I. Idris Assembler Conventions for MC68000

NAME

As.68k - The MC68000 Assembly Language

FUNCTION

The as.68k language facilitates writing machine level code for the MC68000 family of computers. Its primary design goal is to express the readable output of the C compiler code generator p2.68k in a form that can be translated as rapidly as possible to relocatable object form; but it also includes limited facilities for aiding the hand construction of machine code. What it does not include are listing output control, macro definitions, conditional assembly directives, or branch shortening logic.

SYNTAX

Source code is taken as a sequence of zero or more lines, where each line is a sequence of no more than 511 characters terminated by either a newline or a semicolon. The asterisk '*' is a comment delimiter; all characters between the asterisk and the next newline are equivalent to a single newline. Each line may contain at most one command, which is specified by a sequence of zero or more tokens; an empty line is the null command, which does nothing.

A token is any one of the characters "()!+-,:=", the sequence "-(", the sequence ")+", a number, or an identifier. A number is the ASCII value of a character x when written as 'x, or a digit string. A character x may be replaced by the escape sequence '\c, where c is in the sequence "btnvfr" (in either case) for the octal values [010, 015], or where c is one to three decimal digits taken as an octal number giving the value of the character. A digit string is a decimal digit, followed by zero or more letters and digits. If the leading characters are "0x" or "0X", the constant is a hexadecimal number and may contain the letters 'a' through 'f', in either case, to represent the digit values 10 through 15, respectively. Otherwise a leading '0' implies an octal number, which nevertheless may contain the digits '8' and '9'. A non-zero leading digit implies a decimal number.

IDENTIFIERS

An identifier is one or more letters and digits, beginning with a letter; letters are characters in the set [a-zA-Z._~], with uppercase distinguished from lowercase; at most eleven characters of an identifier are retained. Tokens that might be otherwise confused must be separated by whitespace, which consists of the ASCII space ' ' and all non-printing characters.

There are many pre-defined identifiers, most of which represent machine instructions. New identifiers are defined either in a label prefix or in a defining command; the scope of a definition is from its defining occurrence through the end of the source text. A label prefix consists of an identifier followed by a colon ':'; any number of labels may occur at the beginning of a command line. A label prefix defines the identifier as being equal to the (relocatable) address of the next byte for which code

is to be generated. A defining command consists of an identifier, followed by an equals '=', followed by an expression; it defines the identifier as being equal to the value of the expression, which must contain no undefined identifiers.

There are also twenty numeric labels, identified by a digit sequence whose value is in the range [0, 9], followed immediately by a 'b' or an 'f'. A definition of the form "#:" or "# = expr", where # is in the range [0, 9] makes the corresponding #b numeric label defined and renders the corresponding #f undefined; further, any previous references to #f are given this definition. Thus, #b always refers to the last definition of #, which must exist, and #f always refers to the next definition, which must eventually be defined.

EXPRESSIONS

Expressions consist of an alternation of terms and operators. A term is a number, a numeric label, or an identifier. There are three operators: plus "+" for addition, minus "-" for subtraction, and not "!" for bitwise equivalence. If an expression begins with an operator, an implicit zero term is prepended to the expression; if an expression ends with an operator, an error is reported. Thus plus, minus, and not have their usual unary meanings.

Two terms may be added if at most one is undefined or relocatable; two terms may be subtracted if the right is absolute, relocatable to the same base as the left, or involves the same undefined symbol as the left; two terms may be equivalenced only if both are absolute. The base of each term is examined only when an expression is first encountered, so that symbols named as forward references are always considered undefined in this context.

RELOCATION

Relocation is always in terms of one of three code sections: the text section, which normally receives all executable instructions and is normally read only; the data section, which normally receives all initialized data areas and is normally read/write; and the bss section, which can accept only space reservations and is initialized to zeros at start of execution. The pre-defined variable dot "." is maintained as the location counter; its value is the (relocatable) address of the next byte for which code is to be generated at the start of each command line.

The commands ".text", ".data", and ".bss" switch between sections; each sets dot to wherever code generation left off for that section and forces an even byte boundary. Initially, all three sections are of length zero and dot is in the text section. Space may be reserved by a definition of the form ". = . + #", where # is an absolute expression whose value advances the location counter. One byte of space may be conditionally reserved by the command ".even", which ensures that dot is henceforth on an even byte boundary, relative to the start of the current section.

GLOBALS

The command ".globl ident [, ident]*" declares the one or more identifiers to be globally known, i.e., available for reference by other modules. Undefined identifiers may thus be given definitions by other modules at load time, provided there is exactly one globally known defining instance for each such identifier.

A special form of global reference is given by the command ".comm ident, #", which makes ident globally known and treats it as undefined in the current module; if no defining instance exists at link time, however, then this command requests that at least # bytes be reserved in the combined bss section and that the identifier be defined as the address of the first byte of that area (thus bss for "Block Started by Symbol"). As before, # represents an expression whose value is absolute.

CODE GENERATION

Code may be generated into the current section, provided it is text or data, a byte, word, or longword at a time, by one of the commands ".byte # [, #]*", ".word # [, #]*", or ".long # [, #]*". For byte and word generation, each # is an absolute expression whose least significant byte or word defines the next code to be generated; longword generation may also involve relocatable expressions. A word or longword must begin on an even byte boundary.

To simplify generating ASCII strings, an arbitrary string within double quotes, such as "string", may be used as a shorthand for, in this case, ".byte 's, 't, 'r, 'i, 'n, 'g". The usual escape sequences for characters apply.

ADDRESS MODES

All remaining commands are machine instructions. Each is introduced by an identifier defined as an instruction and, depending upon the format defined for that instruction, is followed by zero, one, or two address modes. Some address modes are constrained to be absolute expressions, others must be relocatable addresses within some narrow range of the current location counter, others must be certain machine registers, still others are relatively general.

General address modes are built from expressions and the pre-defined register identifiers "d0" through "d7", "a0" through "a7", "d0.l" through "d7.l", "a0.l" through "a7.l", "d0.w" through "d7.w", "a0.w" through "a7.w", "sp", "sp.l", "sp.w", "pc", "sr", and "ccr". The ".l" (for longword) and ".w" (for word) modifiers are only significant when used with second index registers; "sp" and its variants are synonyms for "a7"; "pc" is the program counter designator for program counter with index mode, and "sr" and "ccr" both designate the status register.

A register address mode consists of simply a register expression. Autoincrement is indicated by the form "(reg)+", autodecrement by "-(reg)", in-

dexing by "expr(reg)", absolute or program counter relative by "expr", and immediate by "#expr", where expr is any expression and reg is any A register expression. The form "expr" becomes program counter relative if it refers to a defined symbol within reach in the same section as the instruction, or if the flag -s was specified to the translator and the symbol is undefined; a non-relocatable expression is made either absolute short or absolute long as needed; an undefined symbol is made absolute if program counter relative cannot be used. In this last case, the mode absolute short is used if -a was specified to the assembler, and absolute long otherwise.

The other index modes are indicated by "expr(pc, reg)" or "expr(reg1, reg2)", where expr is an absolute expression that can be represented within a signed byte, reg and reg2 are any short or long registers, and reg1 is any A register.

Another form of address mode is a mask, for use with the movem instruction. A mask may consist of any A or D register alone, a range of A or D registers such as "d0-d3", or a union of masks such as "a2/d0-d3". The complement of a mask may be obtained by writing "d3-d5 ! 0", which selects all registers except d3, d4, and d5.

PRE-DEFINED IDENTIFIERS

Pre-defined identifiers fall into three groups: registers, command keywords, and instructions. The registers are listed above; other names for registers may only be formed by definitions of the form "temp = d3". The keywords, and the command formats they imply are:

```
.bss
.byte   # [, #]*
.comm   ident, #
.data
.even
.globl  ident [, ident]*
.long   # [, #]*
.text
.word   # [, #]*
```

where # implies an expression, ident is an identifier, and "[x]*" implies zero or more repetitions of x.

To complete the list of commands other than instructions:

```
ident = expr    / defines ident as expr
"string"        / generates code for string characters
```

Instructions frequently have several alternate encodings, depending upon the combination of operands used. Hence, only the names pre-defined in as.68k are listed here. They are:

abcd	ble.s	dbhi	movep.w	sbcd
add.b	bls	dble	moveq	scc
add.l	bls.s	dbls	moveq.l	scs
add.w	blt	dblt	mtccr	seq
adda.l	blt.s	dbmi	mtsr	sf
adda.w	bmi	dbne	mtusp	sge
addi.b	bmi.s	dbpl	muls	sgt
addi.l	bne	dbra	mulu	shi
addi.w	bne.s	dbt	nbcd	sle
addq.b	bpl	dbvc	neg.b	sls
addq.l	bpl.s	dbvs	neg.l	slt
addq.w	bra	divs	neg.w	smi
addx.b	bra.s	divu	negx.b	sne
addx.l	bset	eor.b	negx.l	spl
addx.w	bsr	eor.l	negx.w	st
and.b	bsr.s	eor.w	nop	stop
and.l	btst	eori.b	not.b	sub.b
and.w	bvc	eori.l	not.l	sub.l
andi.b	bvc.s	eori.w	not.w	sub.w
andi.l	bvs	exg	or.b	suba.l
andi.w	bvs.s	ext.l	or.l	suba.w
asl.b	chk	ext.w	or.w	subi.b
asl.l	clr.b	jmp	ori.b	subi.l
asl.w	clr.l	jsr	ori.l	subi.w
asr.b	clr.w	lea	ori.w	subq.b
asr.l	cmp.b	link	pea	subq.l
asr.w	cmp.l	lsl.b	reset	subq.w
bcc	cmp.w	lsl.l	rol.b	subx.b
bcc.s	cmpa.l	lsl.w	rol.l	subx.l
bchg	cmpa.w	lsr.b	rol.w	subx.w
bclr	cmpi.b	lsr.l	ror.b	svc
bcs	cmpi.l	lsr.w	ror.l	svs
bcs.s	cmpi.w	mfsr	ror.w	swap
beq	cmpm.b	mfusp	roxl.b	tas
beq.s	cmpm.l	move.b	roxl.l	trap
bge	cmpm.w	move.l	roxl.w	trapv
bge.s	dbcc	move.w	roxr.b	tst.b
bgt	dbcs	movea.l	roxr.l	tst.l
bgt.s	dbeq	movea.w	roxr.w	tst.w
bhi	dbf	movem.l	rte	unlk
bhi.s	dbge	movem.w	rtr	
ble	dbgt	movep.l	rts	

Note that special mnemonics "mfusp" and "mtusp" are used to access the user stack pointer, rather than a "move" instruction with a special register name. Special mnemonics "mfsr", "mtsr", and "mtccr" are included for the operations move from/to status register and move to condition codes. All five of these mnemonics are followed by a single effective address giving the second operand for the move. Also, note that as.68k requires the size field on instructions that can operate on different sized data; there are no default sizes.

as.68k is not guaranteed to pick the optimal instruction where there is a choice. For example, add.l #2,d0 will not select the addq.l operation. In general, it is wise always to state exactly what you mean.

ERROR MESSAGES

Errors are reported by the as.68k translator in English, albeit terse, as opposed to the conventional assembler flags. Any message ending in an exclamation mark '!' indicates problems with the translator per se (they should "never happen") and hence should be reported to the maintainers. Here is a complete list of errors that can be produced by erroneous input or by an inadequate operating environment:

```
#b undefined
.comm defined
A reg or PC required
absolute expr required
address register required
bad #:
bad #f or #b
bad .comm size
bad command
bad index or indirect
bad mask range
bad movem mask
bad operand(s)
bad temp file read
byte pc range
can't backup .
can't create temp file
can't create xxx
can't load .bss
can't read xxx
can't write object file
data register required
illegal .=
illegal character
missing / operand
missing immediate
missing term
missing xxx
redefinition of xxx
register required
reloc + reloc
relocatable byte
relocatable word
string too large
too many symbols
undefined #f
unknown instruction
word on odd boundary
word pc range
x ! reloc
x - reloc
```

In all cases, "xxx" stands for an actual character, identifier, or filename supplied by as.68k.