```
#!/usr/bin/perl -w
#
# Copyright (c) 2004 Kelly Yancey
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ‘‘AS IS’’ AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
#
# $kbyanc: dyntrace/tools/extract.pl,v 1.8 2004/12/27 10:32:28 kbyanc Exp $
#

#
# Extracts opcodes from Intel’s Instruction Set Reference.
# ftp://download.intel.com/design/Pentium4/manuals/25366714.pdf
#
# Requires graphics/xpdf port (for pdftotext utility) and
# textproc/p5-XML-Writer port to be installed.
#

# Here there be dragons.

use strict;

use POSIX qw(strftime);
use XML::Writer;                                # textproc/p5-XML-Writer

my $pdftotext   = ’/usr/X11R6/bin/pdftotext’;   # graphics/xpfd
my $source      = ’../reference/25366714.pdf’;

my @sections = (
        {
                # General-purpose instructions.
                firstpage       => 320,
                lastpage        => 332,
        },
        {
                # Instructions introduced with Pentium (P5).
                firstpage       => 333,
                lastpage        => 333,
        },
        {
                # MMX instructions.
                group           => ’MMX’,
                firstpage       => 334,
                lastpage        => 337,
        },
        {
                # Instructions introduced with Pentium Pro/Pentium II (P6)
                firstpage       => 338,
                lastpage        => 338,
        },
        {
                # SSE instructions.
                group           => 'SSE',
                firstpage       => 339,
                lastpage        => 346,
        },
        {
                # SSE2/3 instructions.
                group           => 'SSE',
                firstpage       => 347,
                lastpage        => 362,
        },
        {
                # FP instructions.
                group           => 'FP',
                firstpage       => 363,
                lastpage        => 367,
        }
);


my $REG_SEGMENT = 'SRx';
my $REG_DEBUG   = 'DRx';
my $REG_CONTROL = 'CRx';
my $REG_GENERAL = 'reg';
my $REG_FP      = 'ST(i)';
my $REG_MMX     = 'mmxreg';
my $REG_XMM     = 'xmmreg';
my $IMMEDIATE   = 'imm';
my $MEMADDR     = 'mem';


#
# From section B.1.6: Conditional Test Field (tttn).
#
my @condition_list = (
    { 'bitstr' => '0000', 'char' => 'O',  'condition' => 'Overflow'     },
    { 'bitstr' => '0001', 'char' => 'NO', 'condition' => 'No overflow'  },
    { 'bitstr' => '0010', 'char' => 'B',  'condition' => 'Below'        },
    { 'bitstr' => '0011', 'char' => 'NB', 'condition' => 'Not below'    },
    { 'bitstr' => '0100', 'char' => 'E',  'condition' => 'Equals'       },
    { 'bitstr' => '0101', 'char' => 'NE', 'condition' => 'Not equals'   },
    { 'bitstr' => '0110', 'char' => 'NA', 'condition' => 'Not above'    },
    { 'bitstr' => '0111', 'char' => 'A',  'condition' => 'Above'        },
    { 'bitstr' => '1000', 'char' => 'S',  'condition' => 'Sign'         },
    { 'bitstr' => '1001', 'char' => 'NS', 'condition' => 'Not sign'     },
    { 'bitstr' => '1010', 'char' => 'P',  'condition' => 'Parity'       },
    { 'bitstr' => '1011', 'char' => 'NP', 'condition' => 'Not parity'   },
    { 'bitstr' => '1100', 'char' => 'L',  'condition' => 'Less than'    },
    { 'bitstr' => '1101', 'char' => 'NL', 'condition' => 'Not less than'},
    { 'bitstr' => '1110', 'char' => 'NG', 'condition' => 'Not greater than' },
    { 'bitstr' => '1111', 'char' => 'G',  'condition' => 'Greater than' }
);


#
# NO USER-SERVICEABLE PARTS BEYOND HERE
#

my $TRUE  = (1 == 1);
my $FALSE = (1 == 0);
my @prefixes = ();              # Optional prefixes before instructions
my %ops = ();                   # Encodings of instructions themselves
my $lastop;


sub SwapOpArgs($) {
        my $op = shift;

        my $temp = $op->{'args_in'};
        $op->{'args_in'} = $op->{'args_out'};
        $op->{'args_out'} = $temp;

#       die if scalar(@$temp) > 1;

        return $op;
}


sub CopyOp($) {
        my $origop = shift;
        my %newophash = %$origop;

        # XXX Assumes args lists are not modified as both the new
        #     and the original Op's lists refer to the same arrays.

        return (\%newophash);
}


sub AddOp($);   # Prototype to avoid superfluous warnings.
sub AddOp($) {
        my $op = shift;

        $op->{'bitstr'} =~ s!s!x!go;    # Ignore sign-extend.
        $op->{'bitstr'} =~ s!w!x!go;    # Don't care about word size.
        $op->{'bitstr'} =~ s!gg!xx!go;  # Grouping also just specifies
                                        # word size; ignore for now.

        # Condition mask bits.
        if ($op->{'bitstr'} =~ /tttn/o) {
                foreach my $cond (@condition_list) {
                        my $newop = CopyOp($op);
                        $newop->{'bitstr'} =~ s!tttn!$cond->{'bitstr'}!e;
                        $newop->{'description'} .= ': ' . $cond->{'condition'};
                        $newop->{'conditional'} = $cond->{'char'};
                        AddOp($newop);
                }
                return;
        }

        # Direction bit is present in instruction.
        # Record the opcode for each direction separately.
        if ($op->{'bitstr'} =~ /d/o) {
                my $newop = SwapOpArgs(CopyOp($op));
                $newop->{'bitstr'}  =~ s!d!0!o;
                $op->{'bitstr'} =~ s!d!1!o;
                AddOp($newop);
                # FALLTHROUGH
        }

        # Trim all trailing don't-care bits.
        $op->{'bitstr'} =~ s!x+$!!o;

        # Ensure no other characters slipped though.  This catches bugs in
        # the cleanup_bitstr logic, but mostly detects typos in the original
        # Intel manual.
        die 'unknown chars in bit string "', $op->{'bitstr'},
            '", for opcode ', $op->{'opcode'}, ', stopped'
            unless $op->{'bitstr'} =~ /^[01x]+$/o;

        # If two instructions have the same bitstr, then combine their
        # opcodes into one.  We'll take the best list of the arguments
        # available (using length of the list to judge which is better).
        my $existing_op = $ops{$op->{'bitstr'}};
        if (defined $existing_op) {
                my %names = map { $_ => 1 } split('/', $existing_op->{'opcode'});
                $op->{'opcode'} = $existing_op->{'opcode'} . '/' .
                                  $op->{'opcode'}
                    unless $names{$op->{'opcode'}};

                $op->{'detail'} = $existing_op->{'detail'} . ', ' .
                                  $op->{'detail'}
                    unless ($op->{'detail'} eq $existing_op->{'detail'});

                $op->{'args_in'} = $existing_op->{'args_in'}
                    if $#{$existing_op->{'args_in'}} > $#{$op->{'args_in'}};
                $op->{'args_out'} = $existing_op->{'args_out'}
                    if $#{$existing_op->{'args_out'}} > $#{$op->{'args_out'}};
        }

        $lastop = $op;
        $ops{$op->{'bitstr'}} = $op;
}


my @input_only = ('CALL', 'JMP', 'PUSH', 'VERR', 'VERW', 'MONITOR', 'MWAIT');
my @output_only = ('POP', 'SETcc');

my %input_only_hash = map { $_ => 1 } @input_only;
my %output_only_hash = map { $_ => 1 } @output_only;

sub ParseOp($$$$$) {
        my ($bitstr, $opcode, $detail, $description, $group) = @_;
        my @args_in = ();
        my @args_out = ();
        my $conditional;

        $bitstr = $1 . $bitstr if ($detail =~ s!([01]{3,})$!!o);
        $detail = '' if ($detail =~ /^$opcode - /);

        # Do nothing if the bit string contains invalid characters.
        # These characters are indicative of a false match while scanning.
        return if ($bitstr =~ /[=\.]/o);

        # Fix instances of words with superfluous spaces injected
        # between the letters.
        foreach my $word qw(reg memory) {
                my $pattern = join('\s?', split('', $word));
                $detail =~ s!$pattern!$word!g;
        }

        # Save the 'original' detail string to include the final output.
        my $orig_detail = $detail;

        # Some opcode descriptions list multiple bit strings that only differ
        # in the order of the operands.  Detect those cases by looking for
        # opcodes sharing the same opcode and description (both inherited from
        # the parent header).
        if (!$detail && $lastop &&
            $opcode eq $lastop->{'opcode'} &&
            $description eq $lastop->{'description'}) {
                @args_in = @{$lastop->{'args_out'}};
                @args_out = @{$lastop->{'args_in'}};
                goto cleanup_bitstr;
        }

        # If there is no detail description to parse to get the arguments,
        # then try to take them from the bit string itself.  Unfortunately,
        # we have no information about the usage of the arguments in this
        # case, so we have to rely on the @input_only and @output_only hacks.
        if (!$detail) {
                $detail = $bitstr;
                $detail =~ s!\b[01][01cdstw]*\b!!og;
                $detail =~ s!:!,!og;
                $detail =~ s!^[,\s]+!!og;
                $detail =~ s!\b(mod[Ae]?|r/m|rm)\b!!og;
        }

        goto extract_args unless ($detail);

cleanup_detail:

        $detail =~ s!\bimm(ediate)!$IMMEDIATE!go;
        $detail =~ s!\bsegment reg(ister)?(\s?[CDEFGS]S,?)*!$REG_SEGMENT!go;
        $detail =~ s!\bmmxreg(\d)?!$REG_MMX!go;
        $detail =~ s!\bxmmreg(\d)?!$REG_XMM!go;
        $detail =~ s!\bxmmxreg(\d)?!$REG_XMM!go; # Work around typo in doc.
        $detail =~ s!\bST\(\w\)!$REG_FP!go;
        $detail =~ s!\breg(ister)?!$REG_GENERAL!go;
        $detail =~ s!\bmem(ory)?!$MEMADDR!go;

        # Don't care about operand size right now.
        $detail =~ s!\b(m|mem)(8|16|32|64|128)(int)?!$MEMADDR!go;
        $detail =~ s!\b(r|reg)(8|16|32|64|128)!$REG_GENERAL!go;
        $detail =~ s!\bimm\d+!$IMMEDIATE!go;
        $detail =~ s!\b(\d{1,2}-bit|full) displacement!$IMMEDIATE!go;

        # Don't really care which register.
        $detail =~ s!\b(AL, AX, or EAX|AX or EAX|E?[ABCD]X|SP|CL)!$REG_GENERAL!goi;
        $detail =~ s!\bSS!$REG_SEGMENT!go;
        $detail =~ s!\bDR\d!$REG_DEBUG!go;
        $detail =~ s!\bCR\d!$REG_CONTROL!go;
        $detail =~ s!\b(\w+)-\1!$1!go;          # Remove ranges now made redundant.

        # Miscellanious cruft; must go after operand size checks if
        # we ever want to be able to support recording such info.
        $detail =~ s!\bimm count!$IMMEDIATE!go;
        $detail =~ s!\breg\d?!$REG_GENERAL!go;
        $detail =~ s!\(alternate encoding\)!!go;
        $detail =~ s!\bno argument!!go;
        $detail =~ s!\bintersegment!!go;
        $detail =~ s!\b(reg|mem)\s+indirect!$1!go;
        $detail =~ s!\bindirect!$REG_GENERAL!go;
        $detail =~ s!\bdirect!$IMMEDIATE!go;    # e.g. CALL
        $detail =~ s!\bshort!$IMMEDIATE!go;     # e.g. JMP
        $detail =~ s!\btype!$IMMEDIATE!go;      # e.g. INT n
        $detail =~ s!\b\d{1,2}-bit level!$IMMEDIATE!go;
        $detail =~ s!\b\d{1,2}-bit!!go;         # e.g. FICOM
        $detail =~ s!\W\([[:upper:]]\)!!go if ($opcode eq 'ENTER');

        $detail =~ s!\s+! !go;

extract_args:
        if ($opcode eq 'IN') {
                push @args_in, $IMMEDIATE;
                push @args_out, $REG_GENERAL;   # Really AX/EAX but don't care.
                goto cleanup_bitstr;
        } elsif ($opcode eq 'OUT') {
                push @args_in, $REG_GENERAL;    # Really AX/EAX but don't care.
                push @args_out, $IMMEDIATE;
                goto cleanup_bitstr;
        }

        if ($opcode eq 'FCMOVcc') {
                die "unexpected format \"$detail\"" unless $detail =~ /\((\w+)\)/o;
                $conditional = $1;
                push @args_in, $REG_FP;
#               $opcode =~ s!cc$!$condition!;
                $description .= ': ' . $detail;
                goto cleanup_bitstr;
        }

        if ($opcode eq 'FCOMI') {
                push @args_in, $REG_FP;
                goto cleanup_bitstr;
        }

        # Special handling for exchange instructions.
        if ($opcode =~ /XCHG|XADD/o) {
                die "unexpected format: $detail" unless $detail =~ /\b(.+?)(, | with )(.+?)\
b/o;
                push @args_in, $1, $3;
                push @args_out, $1, $3;
                $detail = '';
        }

        # Special handling for comparison and test instructions.
        # e.g. CMP, BT, TEST, etc.
        if ($opcode =~ /CMP/o || $opcode eq 'TEST' ||
            $description =~ /\b(COMPARE|TEST)\b/oi) {
                push @args_in, split(/,|and|to|with/, $detail);
                $detail = '';
        }

        # Special handling for logical operators.
        if ($opcode =~ /^AND/o || $opcode =~ /^OR/o  ||
            $opcode =~ /^XOR/o) {
                die "unexpected format" unless $detail =~ /\b(.+?) to (.+?)\b/o;
                push @args_in, $2, $1;
                push @args_out, $2;
                $detail = '';
        }

        # e.g. FADD "ST(0) = ST(0) + mem"
        if ($detail =~ s!(ST\(\w\)) = (.*) [-+*\/] (.*)!!o) {
                push @args_out, $1;
                push @args_in, $2, $3;
        }

        # e.g. FLDL2T "Load log2(10) into ST(0)"
        if ($detail =~ s!into (\S+)!!o ||
            $description =~ /into (\S+)/o) {
                push @args_out, $1;
        }

        # e.g. RET "adding immediate to SP"
        if ($detail =~ s!\badding (.+?) to (.+?)\b!!o) {
                push @args_in, $1, $2;
        }

        # e.g. IMUL "memory with immediate to register"
        if ($detail =~ s!\b(.+?) with (.+?) to (.+?)\b!!o) {
                push @args_in, $1, $2;
                push @args_out, $3;
        }

        # e.g. IMUL "register with memory"
        if ($detail =~ s!\b(.+?) with (.+?)\b!!o) {
                push @args_in, $1, $2;
                push @args_out, $1;
        }

        # e.g. MOV "CR0 from register"
        if ($detail =~ s!\b(.+?) from (.+?)\b!!o) {
                push @args_out, $1;
                push @args_in, $2;
        }

        # e.g. ADC "register1 to register2"
        if ($detail =~ s!\b(.+?) to (.+?)\b!!o) {
                push @args_in, $1;
                push @args_out, $2;
        }

        # e.g. LAR "from register"
        if ($detail =~ s!\bfrom (.+?)\b!!o) {
                push @args_in, $1;
        }

        # e.g. SMSW "to memory"
        if ($detail =~ s!\bto (.+?)\b!!o) {
                push @args_out, $1;
        }

        if ($detail =~ s!\b(.+?) and (.+?)\b!!o) {
                warn "\"$opcode\" is likely comparison opcode";
                push @args_in, $1, $2;
        }

        # e.g. RCL "register by CL" or "register by 1"
        if ($detail =~ s!\b(.+?) by (.+?)\b!!o) {
                push @args_in, $1;
                push @args_out, $1;
                push @args_in, $2 unless $2 =~ /^\d+$/o;
        }

        if ($input_only_hash{$opcode}) {
                push @args_in, split(/,\s*/, $detail);
                $detail = '';
        }
        elsif ($output_only_hash{$opcode}) {
                push @args_out, split(/,\s*/, $detail);
                $detail = '';
        }

        if ($detail =~ s!\b(.+?)\s?,\s?(.+?)\b!!o) {
                if ($1 eq $IMMEDIATE) {
                        push @args_in, $1;
                } else {
                        push @args_out, $1;
                }
                push @args_in, $2;
        }
        # Add any remaining text in the detail as a single argument.
        #   * Except for special cases handled above, there can never be more
        #     than a single destination argument.
        #   * Immediate arguments can never be the destination.
        #   * The destination of floating-point opcodes is always a register
        #     in the FP stack unless otherwise specified (handled above).
        if ($detail !~ /^\s*$/o) {
                push @args_in, $detail;
                if (!@args_out && $detail ne $IMMEDIATE) {
                        push @args_out, $group eq 'FP' ? $REG_FP : $detail;
                }
        }

        foreach my $arg (@args_in, @args_out) {
                $arg =~ s!^[\s,]+!!o;
                $arg =~ s![\s,]+$!!o;
        }

cleanup_bitstr:
        $bitstr =~ s!:\s+\d+-bit\s+disp.*!!go;

        $bitstr =~ s!:|,! !go;
        $bitstr =~ s!mod[Ae]?!99!goi;
        $bitstr =~ s!r/?m\b!999!goi;            # r/m field
        $bitstr =~ s!(x)?mmreg ?\d?!999!goi;    # SSE xmmregs
        $bitstr =~ s!mmxreg\d?!999!goi;         # MMX mmxregs
        $bitstr =~ s!sreg\d?!99!goi;            # Segment registers.
        $bitstr =~ s!r(eg)?\d{0,2}!999!goi;     # General registers.
        $bitstr =~ s!ST\((i|\d)\)!999!goi;      # FP stack registers.
        $bitstr =~ s!eee!999!goi;               # Control/debug registers.

        # Ignore most other text in the bit string.
        pos($bitstr) = undef;
        while (scalar ($bitstr =~ /\b([[:alpha:]]\w+,?)\b/go)) {
                next if ($1 eq 'tttn');
                $bitstr =~ s!$1!!g;
        }

        $bitstr =~ s!\s!!go;
        $bitstr =~ s!9!x!go;

        AddOp({
                bitstr          => $bitstr,
                opcode          => $opcode,
                description     => $description,
                args_in         => \@args_in,
                args_out        => \@args_out,
                conditional     => $conditional,
                detail          => $orig_detail
        });
}


sub Extract($$) {
        local (*DATA) = shift;
        my $group = shift;
        $group = '' unless $group;

        my $addOK = $FALSE;
        my $savebits;

        my $opcode;
        my $description;
        my $line;                       # Separate line so 'redo' works.
        while ($line = <DATA>) {
                chomp $line;

                my $setop = $FALSE;

                # Stop appending to the description text once we enounter
                # an empty line or a line with a page-feed in it.
                # Includes hack for lone "1" after MOVQ entry.
                if ($line =~ /^\s*$/o ||
                    $line =~ /\cL.*/o ||
                    $line =~ /^\s+\w\s*$/o) {
                        $addOK = $FALSE;
                        next;
                }

                # Work around error in UD2 bit encoding specification (someone
                # confused binary and hexadecimal).
                $line =~ s!\bFFFF\b!1111!go;

                # The Intel PDF uses different characters for hyphens
                # depending on the section the instruction appears in.
                # Make them all consistent (it makes the regexes easier too).
                $line =~ s!\xAD!-!go;           # Used in 'General-Purpose' and FP formats.
                $line =~ s! -- ! - !go;         # Used in SSE3 formats.
                $line =~ s!--! - !go;           # Used in SSE formats.

                # Special handling to parse list of instruction prefixes
                # These appear in the opcode list even though they aren't
                # actually opcodes.
                if ($line =~ /^\s*Prefix Bytes\s*$/oi) {
                        while ($line = <DATA>) {
                                last if ($line =~ /^\s*$/o);
                                next unless ($line =~ /\s*(\w.*?)\s{2,}([01]{3,}.*)/o);
                                my ($detail, $bitstr) = ($1, $2);
                                $bitstr =~ s!\s+!!go;
                                push @prefixes, {
                                        'detail'        => $detail,
                                        'bitstr'        => $bitstr
                                };
                        }
                        $addOK = $FALSE;
                        next;
                }

                # Special handling to undo munging of the Floating-Point
                # instruction encoding text from the PDF-to-text conversion.
                if ($group eq 'FP') {
                        $line =~ s!\xf7!/!go;   # e.g. FIDIV
                        $line =~ s!\xd7!*!go;   # e.g. FIMUL
                        $line =~ s!\xa8! !go;   # e.g. FDIVRP
                        $line =~ s!^\s*([[:upper:]]+)\s([[:upper:]]+)!$1$2!go;

                        $line =~ s!^\s*(ST\(\w\))\s+(\w.*) ([-+*\/]) (.*)!$1 = $2 $3 $4!o;

                        if ($line !~ / - /o &&
                            $line =~ /^\s*(F[[:upper:][:digit:]]+)/o) {
                                $line = $1 . ' - ' . $line;
                        }
                }

reparse:
                if ($line =~ /^\s*([[:upper:]][\w\s\/]+?) - (\w.*\d{0,1})/o) {
                        ($opcode, $description) = ($1, $2);
                        $description =~ s!\s+[01]{3,}.*!!o;

                        if ($savebits) {
                                ParseOp($savebits, $opcode, '', $description,
                                        $group);
                                $savebits = undef;
                        }

                        $setop = $TRUE;
                        $addOK = $TRUE;
                }

                # All lines describing an instruction end with a bit string
                # starting with at least 3 zeros or ones.
                if ($line =~ /^\s*([[:alpha:]].+?)\s+([01]{3,}.*)/o ||
                    $line =~ /^\s*(\d{1,2}-bit\s+[[:alpha:]].+?)\s+([01]{3,}.*)/o) {
                        my ($bitstring, $detail) = ($2, $1);

                        # Hack to parse the F2XM1 instruction; made generic
                        # in case other instances arise in the future.  The
                        # pdf-to-text conversion appears to split lines with
                        # superscripts such that everything after the
                        # superscript appears on a line before the initial
                        # text.  Re-glue the lines back together and insert an
                        # '^' to indicate the original superscript.
                        if (!$opcode) {
                                my $nextline = <DATA>;
                                chomp $nextline;
                                $line =~ s!^\s+!!go;
                                $line = $nextline . '^' . $line;
                                redo;
                        }

                        if ($savebits && $lastop) {
                                ParseOp($savebits,
                                        $lastop->{'opcode'},
                                        $lastop->{'detail'},
                                        $lastop->{'description'},
                                        $group);
                                $savebits = undef;
                        }

                        ParseOp($bitstring, $opcode, $detail, $description,
                                $group);
                        $addOK = $TRUE;
                        next;
                }

                if ($savebits) {
                        # Append any saved text to the current line.  This is to
                        # workaround an artifact of the PDF-to-text conversion where
                        # text in adjacent columns does not always end up on the same
                        # text line (it looks like the second column always comes first
                        # in this scenario).
                        $line .= $savebits if ($savebits);
                        $savebits = undef;
                        goto reparse;
                }

                # As an artifact of the PDF-to-text conversion, sometimes text
                # in the second column of the instruction table appears on line
                # before the text in the first column.  To work around this
                # issue, if we find a bit string on a line by itself, save the
                # text to append to the next line that we read.
                if ($line =~ /^\s*([01]{3,}.*)/o) {
                        $savebits = $line;
                        $addOK = $FALSE;
                        next;
                }

                next if ($setop);
                next unless ($addOK);

                if ($lastop && $description && $line !~ /\w+\s{3,}\w/o) {
                        $lastop->{'description'} .= ' ' . $line
                            if ($lastop->{'description'} eq $description);
                        $description .= ' ' . $line;
                        next;
                }
        }

        close(DATA);
}


sub Output {

        my $xml = new XML::Writer(DATA_MODE => $TRUE, DATA_INDENT => 4);
        $xml->xmlDecl('UTF-8');
        $xml->doctype('oplist', undef, 'oplist.dtd');
        $xml->comment('Automatically generated ' .
                      strftime("%F %T %z", gmtime) .
                      ' from ' . $source);
        $xml->startTag('oplist');

        # Output list of optional instruction prefixes.
        foreach my $prefix (@prefixes) {
                $xml->emptyTag('prefix',
                               'bitmask' => $prefix->{'bitstr'},
                               'detail'  => $prefix->{'detail'});
        }

        # Output list of instruction opcodes sorted by mnemonic.
        foreach my $op (sort {
                                $a->{'opcode'} cmp $b->{'opcode'} or
                                $a->{'bitstr'} cmp $b->{'bitstr'}
                        } values %ops) {
                my $arg;

                my @opargs;
                push @opargs, 'bitmask'         => $op->{'bitstr'};
                push @opargs, 'mnemonic'        => $op->{'opcode'};
                push @opargs, 'detail'          => $op->{'detail'}
                        if ($op->{'detail'});
                push @opargs, 'conditional'     => $op->{'conditional'}
                        if ($op->{'conditional'});

                $xml->startTag('op', @opargs);

                # Output opcode input/output arguments.
                foreach $arg (@{$op->{'args_in'}}) {
                        $xml->emptyTag('arg', 'direction' => 'input',
                                       'type' => $arg);
                }
                foreach $arg (@{$op->{'args_out'}}) {
                        $xml->emptyTag('arg', 'direction' => 'output',
                                       'type' => $arg);
                }

                # Cleanup the description text a little before outputting it.
                my $desc = $op->{'description'};
                $desc =~ s!\(Note:.*\)!!goi;
                $desc =~ s!\s+! !og;
                $desc =~ s!\b- \b!-!og;

                $xml->dataElement('description', $desc) if ($desc);

                $xml->endTag('op');
        }

        $xml->endTag('oplist');
        $xml->end();
}


# --- main ---
{

        foreach my $section (@sections) {
                my $first       = $section->{'firstpage'};
                my $last        = $section->{'lastpage'};

                open(DATA, '-|', "$pdftotext -f $first -l $last -layout $source -")
                    or die;

                Extract(*DATA, $section->{'group'});
        }

        Output();
}
```

```
#!/usr/bin/perl -w
#
# Copyright (c) 2004 Kelly Yancey
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ‘‘AS IS’’ AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
#
# $kbyanc: dyntrace/tools/opfreqs.pl,v 1.1 2004/12/27 12:24:23 kbyanc Exp $
#

#
# Utility for annotating a trace file with the relative frequency and relative
# time of each of the opcodes.
#
# Requires the textproc/p5-XML-Twig port and its dependencies to be installed.
#

use strict;

use IO::Handle;
use XML::Twig;                                # textproc/p5-XML-Twig

my %resolutions = (
        # Name            Tag in trace XML
        'region'       => 'region',
        'program'      => 'program',
        'trace'        => 'dyntrace'
);

my @nodes = ();
my $total_n = 0;
my $total_cycles = 0;


sub UpdateNodes($$) {
        while (my $opcount = pop @nodes) {
                my $n = $opcount->att('n');
                $opcount->set_att('relfreq', sprintf("%0.8f", $n / $total_n))
                    if ($total_n);

                my $cycles = $opcount->att('cycles');
                $cycles = 0 unless $cycles;
                $opcount->set_att('reltime',
                                  sprintf("%0.8f", $cycles / $total_cycles))
                    if ($total_cycles);
        }
        $total_n = 0;
        $total_cycles = 0;
}


sub usage {
        use FindBin qw($Script);

        print STDERR << "EOU" ;
usage: $Script

$Script reads an XML trace file as input, annotates the opcounts with their
relative frequencies and relative timings, and writes the updated trace file
to output.

For example:
        $Script program < myprog.trace > myprog-withfreqs.trace

The level of trace detail used in calculating relative values is determined
by the resolution parameter.  If the resolution is set for 'region', then
the sum of relative values in each region will be 1, meaning the sum for
the entire trace may be larger.  If the resolution is set for 'process',
then the sub of all relative values in the entire trace will be 1.

The supported resolutions are:
EOU

        foreach my $resolution (sort keys %resolutions) {
                print STDERR "\t$resolution\n";
        }

        exit(1);
}


# --- main ---
{

        my $io = new IO::Handle;
        $io->fdopen(fileno(STDIN), 'r');

        usage() unless scalar(@ARGV) == 1;
        my $parentTag = $resolutions{$ARGV[0]};
        usage() unless $parentTag;

        my $twig = XML::Twig->new(
                discard_spaces  => 'true',
                pretty_print    => 'indented',
                keep_atts_order => 'true',
                twig_handlers   => {
                        $parentTag      => \&UpdateNodes,
                        'opcount'       => sub {
                                my $opcount = $_;
                                my $n = $opcount->att('n');
                                my $cycles = $opcount->att('cycles');

                                push @nodes, $opcount;
                                $total_n += $n if $n;
                                $total_cycles += $cycles if $cycles;
                        }
                }
        );

        $twig->parse($io);
        $twig->flush();
}
```

```
#!/usr/bin/perl -w
#
# Copyright (c) 2004 Kelly Yancey
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
#
# THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ''AS IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
#
# $kbyanc: dyntrace/tools/opgroup.pl,v 1.1 2004/12/27 10:36:45 kbyanc Exp $
#

#
# Utility for grouping opcodes in trace files.
#
# Requires the textproc/p5-XML-Twig port and its dependencies to be installed.
#

use strict;

use IO::Handle;
use XML::Twig;                                  # textproc/p5-XML-Twig


# All implemented grouping methods and the routine to call per opcount tag
# to perform the grouping.
my %groupings = (
        'opcode'        => \&GroupByOpcode,
        'mnemonic'      => \&GroupByMnemonic
);


# List of all opcount attributes we can combine the values of and the routine
# used to do the combining.
my %combineMap = (
        'n'             => \&CombineAdd,
        'cycles'        => \&CombineAdd,
        'min'           => \&CombineMin,
        'max'           => \&CombineMax
);


# List of all groups as defined by the grouping routine.
# Reset for each region tag in the trace file.
my %groups = ();


sub CombineAdd($$) {
        return $_[0] unless $_[1];
        return $_[1] unless $_[0];
        return $_[0] + $_[1];
}


sub CombineMin($$) {
        return $_[0] unless ($_[1] and $_[1] < $_[0]);
        return $_[1];
}


sub CombineMax($$) {
        return $_[0] unless ($_[1] and $_[1] > $_[0]);
        return $_[1];
}


sub Combine($$) {
        my ($group, $opcode) = @_;
        my $val;

        # Iterate through all of the attributes listed in the combineMap
        # and call the appropriate combiner routine for updating the group’s
        # counter attribute to include any value in the opcode.
        while (my ($attrname, $combiner) = each %combineMap) {
                $val = &$combiner($group->att($attrname),
                                  $opcode->att($attrname));
                $group->set_att($attrname => $val) if $val;
        }
}


#
# Group all counters for the same opcode, regardless of any prefixes.
# Opcodes are identified by their unique bitmask.  The prefix attribute
# is lost in the grouping.
#
sub GroupByOpcode($$) {
        my ($twig, $opcount) = @_;

        my $bitmask = $opcount->att(’bitmask’);

        # If we have not seen an opcode with this bitmask before, add it to
        # the group hash as a new group.  Remove the prefixes attribute as it
        # won’t be meaningful once we are done grouping.
        if (!exists($groups{$bitmask})) {
                $groups{$bitmask} = $opcount;
                $opcount->del_att(’prefixes’);
                return 1;
        }

        # If we have already have a group for this opcode, integrate the
        # opcode’s counters into the group’s counters and discard the opcode.
        Combine($groups{$bitmask}, $opcount);
        $opcount->delete();
        return 1;
}


#
# Group all counters for opcodes with the same mnemonic.
# The prefix, detail, and bitmask attributes are lost in this grouping.
#
sub GroupByMnemonic($$) {
        my ($twig, $opcount) = @_;

        my $mnemonic = $opcount->att('mnemonic');

        if (!exists($groups{$mnemonic})) {
                $groups{$mnemonic} = $opcount;
                $opcount->del_att('prefixes', 'detail', 'bitmask');
                return 1;
        }

        Combine($groups{$mnemonic}, $opcount);
        $opcount->delete();
        return 1;
}


sub usage {
        use FindBin qw($Script);

        print STDERR << "EOU" ;
usage: $Script group-method

$Script reads an XML trace file as input, groups the opcodes in the trace
per the method specified by the group-method argument, and writes the updated
trace file to output.

For example:
        $Script mnemonic < myprog.trace > myprog-grouped.trace

The supported group-method values are:
EOU

        foreach my $key (keys %groupings) {
                print STDERR "\t$key\n";
        }

        exit(1);
}


# --- main ---
{

        my $io = new IO::Handle;
        $io->fdopen(fileno(STDIN), 'r');

        usage() unless scalar(@ARGV) == 1;
        my $grouper = $groupings{$ARGV[0]};
        usage() unless $grouper;

        my $twig = XML::Twig->new(
                discard_spaces  => 'true',
                pretty_print    => 'indented',
                keep_atts_order => 'true',
                twig_handlers   => {
                        'prefix'        => sub { $_->delete(); },
                        'region'        => sub { %groups = (); },
                        'opcount'       => $grouper
                }
        );

        $twig->parse($io);
        $twig->flush();
}
```

```
<!--
        XSLT stylesheet for producing a simple HTML document from a
        opcode list XML file.

        e.g.: xsltproc -o oplist.html oplist-to-html.xsl oplist-x86.xml

        $kbyanc: dyntrace/tools/oplist-to-html.xsl,v 1.2 2004/12/27 10:33:03 kbyanc Exp $
 -->

<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns="http://www.w3.org/TR/xhtml1/strict">

<xsl:output
        method="html"
        indent="yes"
        encoding="iso-8859-1"
/>

<xsl:template match="oplist">
<html>
    <h1>Instruction Prefixes:</h1>
    <table border="1">
    <tr>
        <td>Description</td>
        <td>Encoding</td>
    </tr>
    <xsl:apply-templates select="prefix"/>
    </table>

    <br />

    <h1>Opcodes:</h1>
    <table border="1">
    <tr>
        <td>Mnemonic</td>
        <td>Description</td>
        <td>Detail</td>
        <td>Encoding</td>
    </tr>
    <xsl:apply-templates select="op"/>
    </table>
</html>
</xsl:template>

<xsl:template match="oplist/prefix">
    <tr>
        <td><xsl:value-of select="@detail"/></td>
        <td><xsl:value-of select="@bitmask"/></td>
    </tr>
</xsl:template>

<xsl:template match="oplist/op">
    <tr>
        <td>
            <xsl:value-of select="@mnemonic"/>
            <xsl:if test="@conditional">
                (<xsl:value-of select="@conditional"/>)
            </xsl:if>
        </td>
        <td><xsl:apply-templates select="description"/></td>
        <td><xsl:value-of select="@detail"/></td>
        <td><xsl:value-of select="@bitmask"/></td>
    </tr>
</xsl:template>
```

```
</xsl:stylesheet>
```

```
<!--
        XSLT stylesheet for producing a simple HTML document from a
        dyntrace results file.

        e.g.: xsltproc -o my-prog.html trace-to-html.xsl my-prog.trace

        $kbyanc: dyntrace/tools/trace-to-html.xsl,v 1.3 2004/12/27 10:33:51 kbyanc Exp $
 -->

<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns="http://www.w3.org/TR/xhtml1/strict">

<xsl:output
        method="html"
        indent="yes"
        encoding="iso-8859-1"
/>

<xsl:template match="dyntrace">
<html>
    <h1>Instruction Prefixes:</h1>
    <table border="1">
    <tr>
        <td>Id</td>
        <td>Description</td>
        <td>Encoding</td>
    </tr>
    <xsl:for-each select="prefix">
        <tr>
            <td><xsl:value-of select="@id"/></td>
            <td><xsl:value-of select="@detail"/></td>
            <td><xsl:value-of select="@bitmask"/></td>
        </tr>
    </xsl:for-each>
    </table>

    <br />

    <xsl:for-each select="program">
        <h1><xsl:value-of select="@name"/></h1>
        <xsl:apply-templates select="region"/>
    </xsl:for-each>

</html>
</xsl:template>


<xsl:template match="dyntrace/program/region">
    <h2>Region: <xsl:value-of select="@type"/></h2>
    <table border="1">
    <tr>
        <td>Mnemonic</td>
        <td>Prefixes</td>
        <td align="right">N</td>
        <td>Description</td>
        <td>Encoding</td>
    </tr>

    <xsl:for-each select="opcount">
        <tr>
            <td><xsl:value-of select="@mnemonic"/></td>
            <td><xsl:value-of select="@prefixes"/></td>
            <td align="right"><xsl:value-of select="@n"/></td>
            <td><xsl:value-of select="@detail"/></td>
            <td><xsl:value-of select="@bitmask"/></td>
        </tr>
    </xsl:for-each>

    </table>
</xsl:template>

</xsl:stylesheet>
```