Stack Machine Commands

Sharon Correll
Last modified: 14 December 2005

This document describes the commands that are defined in Graphite's stack machine, which are used to run rules and test their constraints.

All arguments are 8-bit values. Angle brackets (<arg>) indicate a signed value, curly braces ({arg}) indicate an unsigned value, and double curly braces ({{arg}}) indicate a 16-bit unsigned value.

1 General arithmetic operations

NOP

Do nothing.

PushByte <byte>

Push the given 8-bit signed number onto the stack.

PushByteU {byte}

Push the given 8-bit unsigned number onto the stack.

PushShort <byte1> <byte2>

Treat the two arguments as a 16-bit signed number, with byte1 as the most significant. Push the number onto the stack.

PushShortU {byte1} {byte2}

Treat the two arguments as a 16-bit unsigned number, with byte1 as the most significant. Push the number onto the stack.

PushLong <byte1> <byte2> <byte3> <byte4>

Treat the four arguments as a 32-bit number, with byte1 as the most significant. Push the number onto the stack.

Add

Pop the top two items off the stack, add them, and push the result.

Sub

Pop the top two items off the stack, subtract the first (top-most) from the second, and push the result.

Mul

Pop the top two items off the stack, multiply them, and push the result.

Div

Pop the top two items off the stack, divide the second by the first (top-most), and push the result.

Min

Pop the top two items off the stack and push the minimum.

Max

Pop the top two items off the stack and push the maximum.

Neg

Pop the top item off the stack and push the negation.

Trunc8

Pop the top item off the stack and push the value truncated to 8 bits.

Trunc16

Pop the top item off the stack and push the value truncated to 16 bits.

Cond

Pop the top three items off the stack. If the first == 0 (false), push the third back on, otherwise push the second back on.

And

Pop the top two items off the stack and push their logical *and*. Zero is treated as false; all other values are treated as true.

Or

Pop the top two items off the stack and push their logical *or*. Zero is treated as false; all other values are treated as true.

Not

Pop the top item off the stack and push its logical negation (1 if it equals zero, 0 otherwise.

Equal

Pop the top two items off the stack and push 1 if they are equal, 0 if not.

NotEq

Pop the top two items off the stack and push 0 if they are equal, 1 if not.

Less

Pop the top two items off the stack and push 1 if the next-to-the-top is less than the top-most; push 0 othewise.

Gtr

Pop the top two items off the stack and push 1 if the next-to-the-top is greater than the top-most; push 0 othewise.

LessEq

Pop the top two items off the stack and push 1 if the next-to-the-top is less than or equal to the top-most; push 0 otherwise.

GtrEq

Pop the top two items off the stack and push 1 if the next-to-the-top is greater than or equal to the top-most; push 0 otherwise

2 Rule processing and constraints

Next

Move the current slot pointer forward one slot (used after we have finished processing that slot).

CopyNext

Copy the current slot from the input to the output and move the current slot pointer forward one slot.

PutGlyph {{output-class}}

Put the first glyph of the specified class into the output. Normally used when there is only one member of the class, and when inserting.

PutSubs <slot-offset> {{input-class}} {{output-class}}

Determine the index of the glyph that was the input in the given slot within the input class, and place the corresponding glyph from the output class in the current slot. The slot number is relative to the current input position.

PutSubs2 <slot-offset-1> {{input-class-1}} <slot-offset-2> {{input-class-2}} {{output-class}}

Determine the indices of the glyph that was the input in the given slots within the input classes, and place the corresponding glyph from the output class in the current slot. The formula for determing the output is

```
(index1 * length-of-input-class-2) + index2
```

where the indices are zero-based.

The slot numbers are relative to the current input position.

PutSubs3 <slot-offset-1> {{input-class-1}} <slot-offset-2> {{input-class-2}} <slot-offset-3> {{input-class-3}} {{output-class}}

Determine the indices of the glyph that was the input in the given slots within the input classes, and place the corresponding glyph from the output class in the current slot. The formula for determing the output is

```
(((index1 * length-of-input-class-2)
      + index2) * length-of-input-class-3) + index3
```

where the indices are zero-based.

The slot numbers are relative to the current input position.

PutCopy <slot-offset>

Copy the glyph that was in the input in the given slot into the current output slot. The slot number is relative to the current input position.

Insert

Insert a new slot before the current slot and make the new slot the current one.

Delete

Delete the current item in the input stream.

Assoc {cnt} <slot-1> .. <slot-cnt>

Set the associations for the current slot to be the given slot(s) in the input. The first argument indicates how many slots follow. The slot offsets are relative to the current input slot.

ContextItem <slot-offset> {byte-cnt}

If the slot currently being tested is not the slot specified by the <slot-offset> argument (relative to the stream position, the first modified item in the rule), skip the given number of bytes of stack-machine code. These bytes represent a test that is irrelevant for this slot.

AttrSet {slot-attr}

Pop the stack and set the value of the given attribute to the resulting numerical value.

AttrAdd {slot-attr}

Pop the stack and adjust the value of the given attribute by adding the popped value.

AttrSub {slot-attr}

Pop the stack and adjust the value of the given attribute by subtracting the popped value.

AttrSetSlot {slot-attr}

Pop the stack and set the given attribute to the value, which is a reference to another slot, making an adjustment for the stream position. The value is relative to the current stream position. [Note that corresponding add and subtract operations are not needed since it never makes sense to add slot references.]

IAttrSet {slot-attr} {index}

Pop the stack and set the value of the given indexed attribute to the resulting numerical value. Not to be used for attributes whose value is a slot reference. [Currently the only non-slot-reference indexed slot attributes are *userX*.]

Not supported in version 1.0 of the font tables.

IAttrAdd {slot-attr} {index}

Pop the stack and adjust the value of the given indexed slot attribute by adding the popped value. Not to be used for attributes whose value is a slot reference. [Currently the only non-slot-reference indexed slot attributes are *userX*.]

Not supported in version 1.0 of the font tables.

IAttrSub {slot-attr} {index}

Pop the stack and adjust the value of the given indexed slot attribute by subtracting the popped value. Not to be used for attributes whose value is a slot reference. [Currently the only non-slot-reference indexed slot attributes are *userX*.]

Not supported in version 1.0 of the font tables.

IAttrSetSlot {slot-attr} {index}

Pop the stack and set the given indexed attribute of the current slot to the value, which is a reference to another slot, making an adjustment for the stream position. The value is relative to the current stream position. [Currently the only indexed slot attributes are *component.X.ref*.]

PushSlotAttr {slot-attr} <slot-offset>

Look up the value of the given slot attribute of the given slot and push the result on the stack. The slot offset is relative to the current input position.

PushISlotAttr {slot-attr} <slot-offset> <index>

Push the value of the indexed slot attribute onto the stack. [The current indexed slot attributes are *component.X.ref* and *userX*.]

PushGlyphAttr {{glyph-attr}} <slot-offset>

Look up the value of the given glyph attribute of the given slot and push the result on the stack. The slot offset is relative to the current input position.

PushGlyphMetric {glyph-metric} <slot-offset> <level>

Look up the value of the given glyph metric of the given slot and push the result on the stack. The slot offset is relative to the current input position. The level indicates the attachment level for cluster metrics.

PushFeat {feat}

Push the value of the given feature for the current slot onto the stack.

PushAttToGlyphAttr {{glyph-attr}} <slot-offset>

Look up the value of the given glyph attribute for the slot indicated by the given slot's *attach.to* attribute. Push the result on the stack.

PushAttToGlyphMetric {glyph-metric} <slot-offset> <level>

Look up the value of the given glyph metric for the slot indicated by the given slot's *attach.to* attribute. Push the result on the stack.

PushProcState {procState}

Push the value of the given processing state item onto the stack. (Examples of processing state items are JustifyMode and JustifyLevel).

PushVersion

Get the version number of the stack machine language handled by the engine and push it on the stack.

PopRet

No more processing is needed for this rule. Pop the top of the stack and return that value. For rule action code, the return value is the number of positions to move the stream position forward (or backward, if the number is negative) for the next rule. For constraint code, the return value is a boolean indicating whether the constraint succeeded.

RetZero

Terminate the processing and return zero.

RetTrue

Terminate the processing and return true (1).

3 Not yet implemented

NextN <count>

Move the current slot pointer by the given number of slots (used after we have finished processing the current slot). The count may be positive or negative. Should not be used to copy a range of slots; CopyNext is needed for that.

PushIGlyphAttr {glyph-attr} <slot-offset> <index>

Push the value of the indexed glyph attribute onto the stack. [Examples of indexed glyph attributes are *component.X.box.top, component.X.box.bottom,* etc.]

4 File Name

Stack Machine Commands.doc