|  |  |  |  |
| --- | --- | --- | --- |
| File Name | Line (Final) | Why We Changed it | What we Changed it to |
| coder.ssl | 32-122 | Added the input T-codes to match the output T-codes in semantic.ssl | tMultiply  firstOutputToken = tMultiply  tDivide  tModulus  tAdd  tSubtract  tEQ  tNE  tGT  tGE  tLT  tLE  tAnd  tInfixAnd  tOr  tInfixOr  tNegate  tNot  tChr  tOrd  tEoln  tEOF  tVarParm  tFetchAddress  tFetchInteger  tFetchString  tFetchBoolean  tAssignBegin  tAssignAddress  tAssignInteger  tAssignString  tAssignBoolean  tStoreParmAddress  tStoreParmInteger  tStoreParmString  tStoreParmBoolean  tSubscriptBegin  tSubscriptAddress  tSubscriptInteger  tSubscriptString  tSubscriptBoolean  tArrayDescriptor  tFileDescriptor  tIfBegin  tIfEnd  tConcatenate  tSubstring  tLength  tStringEqual  tCaseBegin  tCaseElse  %tWhileBegin  %tRepeatBegin  %tRepeatControl  tLoopBegin  tLoopBreakWhen  tLoopTest  tLoopEnd  tCallBegin  tParmEnd  tProcedureEnd  tWriteBegin  tReadBegin  tTrapBegin  tWriteEnd  tReadEnd  % Compound T-codes are those that take operands  tLiteralAddress  firstCompoundOutputToken = tLiteralAddress  tLiteralInteger  tLiteralBoolean  tLiteralString  %tStringDescriptor  %tSkipString  tIfThen  tIfMerge  tCaseSelect  tCaseMerge  tCaseEnd  %tWhileTest  %tWhileEnd  %tRepeatTest  tSkipProc  tCallEnd  tLineNumber  tTrap  lastCompoundOutputToken = tTrap  tEndOfFile  lastOutputToken = tEndOfFile; |
| coder.ssl | 168- 175 | Added the trap codes for the new JT run time monitor string operation traps | trAssignString = 101  trChrString = 102  trConcatenate = 103  trSubstring = 104  trLength = 105  trStringEqual = 106  trReadTring = 107  trWriteString = 108 |
| coder.ssl | 181 | Added a string kind | string = 3; |
| coder.ssl | 283 | Removed oOperandPushChar | % %oOperandPushChar |
| coder.ssl | 1501, 1502, 1986, 1987 | Removed handling of tLiteralChar | % | tLiteralChar:  % oOperandPushChar |
| coder.ssl | 845, 846 | Removed handling of tStoreParmChar | % | tStoreParmChar, tStoreParmBoolean:  | tStoreParmBoolean: |
| coder.ssl | 1509-  1517 | Changed handling of tSkipstring to tLiteralString | | tLiteralString:  % Emit string literal to data area  oEmitNone(iData) % .data  % Emit the string  oEmitString % sNNN: .asciz "SSSSS"  oEmitNone(iText) % .text  % Get the string literal's address  oOperandPushStringDescriptor  @EmitStringDescriptor % lea sNNN, %T |
| coder.ssl | 739- 743 | Changed handling of tSkipstring to tLiteralString | | tLiteralString:  % Emit a string constant to the data area  oEmitNone(iData) % .data  oEmitString % sNN: .asciz "SSSSS"  oEmitNone(iText) % .text |
| coder.ssl | 1595-1600, 1993-1995, 2077, 2080 | Removed Char operations tAssignChar and tSubscriptChar |  |
| coder.ssl | 1999 | Added logic for tAssignString |  |
| coder.ssl | 2152- 2182 | Added logic for substring | OperandSubstringPopPop:  @SaveTempregsToStack  @OperandForceToStack  @OperandPopAndFreeTemp  @OperandForceToStack  @OperandPopAndFreeTemp  @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp  oOperandPushMode(mTrap)  oOperandSetValue(trSubstring)  oOperandSetLength(two)  oEmitSingle(iCall)  oOperandPop  oOperandPushMode(mStackReg)  oOperandSetLength(two)  oOperandPushMode(mManifest)  oOperandSetLength(two)  oOperandSetValue(twelve)  oEmitDouble(iAdd)  oOperandPop  oOperandPop  oOperandPushMode(mScratchReg1)  oOperandSetLength(two)  oOperandPushMode(mResultReg)  oOperandSetLength(two)  oEmitDouble(iMov)  oOperandPop  @RestoreTempRegsFromStack  @OperandForceIntoTemp  oOperandSetLength(two) |
| coder.ssl | 2020- 2023 | Added tLength and tSubscript to the OperandPushExpressionAssignPopPop | | tLength:  @OperandLength  | tSubstring:  @OperandSubstringPopPop |
| coder.ssl | 150 | Added a ten attribute for shifting for string size | ten = 10 |
| coder.ssl | 782 - 823 | Changed logic for Block to account for tLiteralString | {[  %%%%%%%%%%%%%EDITED%%%%%%%%%%%%%%  | tLiteralString:  % Emit a string constant to the data area  oEmitNone(iData) % .data  oEmitString % sNN: .asciz "SSSSS"  oEmitNone(iText) % .text  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  | tLiteralInteger:  % initialize an array or file descriptor  oOperandPushInteger % ... lower bound / file number  tLiteralAddress  oOperandPushVariable % ... lb/fn, desc  oOperandSwap % ... desc, lb/fn  oEmitDouble(iMov) % movl lb/fn, desc  [  | tArrayDescriptor:  % it's an array - save the lower bound and get the upper  oArrayEnterLowerBound  oOperandPop % ... array desc  @OperandPopAndFreeTemp % ...  tLiteralInteger  oOperandPushInteger % ... upper bound  oArrayEnterUpperBound  tLiteralAddress  oOperandPushVariable % ... ub, arraydesc+4  oOperandSwap % ... arraydesc+4, ub  oEmitDouble(iMov) % movl ub, desc+4  tArrayDescriptor  | tFileDescriptor:  ]  oOperandPop % ... desc  oOperandPop % ...  | tSkipProc:  % Branch around procedure definitions.  oEmitUnconditionalForwardBranch  oFixPushLastAddress    @Routine  {[  | tSkipProc:  @Routine  | \*:  >  ]}    oFixForwardBranch % Branch around procedure defn's  oFixPopAddress    | tAssignBegin:  @AssignStmt  | tCallBegin:  @CallStmt  | tIfBegin:  @IfStmt  | tLoopBegin:  @WhileStmt  % | tRepeatBegin:  % @RepeatStmt  | tCaseBegin:  @CaseStmt  | tWriteBegin:  @WriteProc  | tReadBegin:  @ReadProc  | tTrapBegin:  @TrapStmt  | \*:    >  ]}; |
| coder.ssl | 879 - 884 | Added logic for tStoreParmString and removed logic for tStoreParmChar | | tStoreParmString:  oOperandSwap  oOperandPushCopy  oOperandSwapLeftAnddest  oOperandSetLength(string)  @OperandAssignStringPopPop |
| coder.ssl | 924 | Removed tWhileBegin and added tLoopBegin | | tLoopBegin:  @WhileStmt |
| coder.ssl | 1109 – 1115 | Added logic for loop breaks | oFixPushAddress % bNNN:  oEmitMergeSourceCoordinate  @Statements  tLoopBreakWhen  @OperandPushBooleanControlExpression % ... cond |
| coder.ssl | 1141 | Removed tWhileEnd and added tLoopEnd |  |
| coder.ssl | 1150 | Added logic to SkipToEndWhile for loop breaks | {[  | tLoopEnd: %%tWhileEnd:  >  | tLoopBegin: %%tWhileBegin: % ignore nested while statements  @SkipToEndWhile  | \*:  ? % accept and ignore anything else  ]}; |
| coder.ssl | 1269-1294 | Added logic for case statements | EmitDefaultCaseAbort:  % Emit a case abort alternative to handle selector values  % which do not match a label.  oCaseEnterAbortAddress % aNNN:  % Save the abort address for use by the selector out-of-range error  % checks in the EmitCaseSubscriptJump rule.  oFixPushAddress % bNNN: (label for abort)  oFixSwapAddresses % (keep branch around case body on top)  oEmitMergeSourceCoordinate  [  | tCaseElse:  oEmitMergeSourceCoordinate  @Statements  tCaseMerge  oEmitCaseMergeBranch  | \*:  oOperandPushMode(mLineNum) % ... n  oOperandSetLength(word)  @OperandForceToStack % pushl n  oOperandPop % ...  oOperandPushMode(mTrap)  oOperandSetValue(trCaseAbort)  oEmitSingle(iCall) % call caseAbort  oOperandPop  ]; |
| coder.ssl | 1550 – 1576 | Added logic for tLiteralString. TstringEqual, and removed logic for tStringDescriptor and tLiteralChar | OperandPushExpression:  {[  | tLiteralAddress:  @OperandPushVariable  | tLiteralInteger:  oOperandPushInteger  % | tLiteralChar:  % oOperandPushChar  | tLiteralBoolean:  oOperandPushBoolean  % | tStringDescriptor:  % oOperandPushStringDescriptor  % % Get a string literal's address  % @EmitStringDescriptor % lea sNNN, %T  | tLiteralString:  % Emit string literal to data area  oEmitNone(iData) % .data  % Emit the string  oEmitString % sNNN: .asciz "SSSSS"  oEmitNone(iText) % .text  % Get the string literal's address  oOperandPushStringDescriptor  @EmitStringDescriptor %lea sNNN, %T  | tStringEqual:  @OperandStringEqualPop  | tNegate:  @OperandNegate  | tAdd:  @OperandAddPop |
| coder.ssl | 1595 – 1600 | Added string operations to OperandPushExpression | | tLength:  @OperandLength  | tSubstring:  @OperandSubstringPopPop  | tConcatenate:  @OperandConcatenatePop |
| coder.ssl | 1653 | Removed tSubscriptChar and replaced it with tSubscriptString |  |
| coder.ssl | 1671 | Added tFetchString logic |  |
| coder.ssl | 1677-1758 | Added OperandSubscriptStringPop,  OperandSubscriptNonManifestStringPop, OperandUncheckedSubscriptNonManifestStringPop,  OperandCheckedSubscriptNonManifestStringPop, logic | OperandSubscriptStringPop:  % if the subscript is manifest fold it out,  % otherwise generate subscripting code  [ oOperandChooseMode  | mManifest:  oOperandSwap  [ oOperandChooseMode  | mTempIndirect:  % var parameter subscripting cannot be folded  oOperandSwap  @OperandSubscriptNonManifestStringPop  | \*:  oOperandSwap % ... array, subscript  oOperandFoldManifestSubscript  oOperandPop % ... array[subscript]  ]  | \*:  @OperandSubscriptNonManifestStringPop  ]  oOperandSetLength(word);  OperandSubscriptNonManifestStringPop:  [ oOptionTestChecking  | yes:  @OperandCheckedSubscriptNonManifestStringPop  | \*:  @OperandUncheckedSubscriptNonManifestStringPop  ];  OperandUncheckedSubscriptNonManifestStringPop:  % Optimized non-bounds checking subscript operation  oOperandSwap % ... subscript, arraydesc  [ oOperandChooseMode  | mTempIndirect:  % Var parameter array - don't know the characteristics  % until run time, so give up and use regular checked subscripting  oOperandSwap  @OperandCheckedSubscriptNonManifestIntegerPop  | mStatic:  % Any other array - know all the characteristics now,  % so optimize subscripting as best we can  oOperandSwap % ... arraydesc, subscript  % Scale subscript by integer element size  @OperandForceIntoTemp % movl subscript, %T  oOperandPushMode(mManifest)  oOperandSetLength(word)  oOperandSetValue(ten)  oEmitDouble(iShl) % shl $2, %T  oOperandPop  oOperandSwap % ... %T, arraydesc  % Fold lower bound into array address to avoid normalizing  % subscript at run time  oOperandPushArrayLowerBound % ... %T, arraydesc, lower  oOperandSwap % ... %T, lower, arraydesc  oOperandSetMode(mManifest) % (eliminate indirection)  oOperandPushMode(mManifest) % ... %T, lower, arraydesc, 8  oOperandSetLength(word)  oOperandSetValue(eight)  oOperandAddManifestValues % ... %T, lower, arraydesc+8, 8  oOperandPop % ... %T, lower, arraydesc+8  oOperandSwap % ... %T, arraydesc+8, lower  oOperandPushCopy % ... %T, arraydesc+8, lower, lower  oOperandAddManifestValues % (scale lower bound by integer size)  oOperandPop  oOperandPushCopy  oOperandAddManifestValues % ... %T, arraydesc+8, lower\*4, lower  oOperandPop % ... %T, arraydesc+8, lower\*4  oOperandSubtractManifestValues % ... %T, arraydesc+8-lower\*4, lower\*4  oOperandPop % ... %T, arraydesc+8-lower\*4  % Add array base to subscript  oOperandSetMode(mStaticManifest) % (u+normalizedArrayBase)  oEmitDouble(iAdd) % addl $u+normalizedArrayBase, %T  oOperandPop % ... %T  % Element address is in %T  oOperandSetMode(mTempIndirect) % ... (%T)  oOperandSetLength(word)  ];OperandCheckedSubscriptNonManifestStringPop:  % Default bounds checking subscript operation  % Get subscript % ... arraydesc, subscript  @OperandForceIntoTemp % ... arraydesc, %T  % Check range if checking, otherwise don't bother  [ oOptionTestChecking  | yes:  oOperandSwap % ... %T, arraydesc  @OperandPushArrayUpperBound % ... %T, arraydesc, upper  oOperandSwapLeftAndDest % ... arraydesc, %T, upper  @EmitCmp % cmpl upper, %T  oOperandPushJumpCondition(iJle) % if subscript <= upper  oEmitConditionalForwardBranch % jle fNN  oFixPushLastAddress  oOperandPop % ... arraydesc, %T, upper  oOperandPop % ... arraydesc, %T  oOperandSwap % ... %T, arraydesc  @OperandPushArrayLowerBound % ... %T, arraydesc, lower  oOperandSwapLeftAndDest % ... arraydesc, %T, lower  @EmitCmp % cmpl lower, %T  oOperandPushJumpCondition(iJge) % if subscript >= lower  oEmitConditionalForwardBranch % jge fMM  oFixPushLastAddress  oOperandPop % ... arraydesc, %T, lower  oOperandPop % ... arraydesc, %T  @EmitSubscriptAbort % call subscriptAbort  oFixForwardBranch % fMM:  oFixPopAddress  oFixForwardBranch % fNM:  oFixPopAddress  | \*:  ]  % Normalize subscript % ... arraydesc, %T  oOperandSwap % ... %T, arraydesc  @OperandPushArrayLowerBound % ... %T, arraydesc, lower  oOperandSwapLeftAndDest % ... arraydesc, %T, lower  @OperandSubtractPop % subl lower, %T  % Scale subscript by word size % ... arraydesc, %T  oOperandPushMode(mManifest)  oOperandSetLength(word)  oOperandSetValue(ten) % ... arraydesc, %T, 2  oEmitDouble(iShl) % shll $2, %T  oOperandPop % ... arraydesc, %T  % Add normalized and scaled subscript to array address  oOperandSwap % ... %T, arraydesc  [ oOperandChooseMode  | mStatic:  % Optimize by folding array offset into array descriptor address  oOperandPushMode(mManifest)  oOperandSetLength(word)  oOperandSetValue(eight)  oOperandAddManifestValues % ... %T, arraydesc+8, 8  oOperandPop % ... %T, arraydesc+8  @OperandForceAddressIntoTemp % mov $arraydesc+8, %T2  % ... %T, %T2  % Add array address to normalized and scaled subscript  oEmitDouble(iAdd) % addl %T2, %T  @OperandPopAndFreeTemp % ... %T  | \*:  % Can't optimize  @OperandForceAddressIntoTemp % ... %T, %T2  oOperandPushMode(mManifest)  oOperandSetLength(word)  oOperandSetValue(eight) % ... %T, %T2, 8  oEmitDouble(iAdd) % addl $8, %T2  oOperandPop % ... %T, %T2  oEmitDouble(iAdd) % addl %T2, %T  @OperandPopAndFreeTemp % ... %T  ]  % Result element address is in %T  oOperandSetMode(mTempIndirect) % ... (%T)  oOperandSetLength(word); |
| coder.ssl | 2219-2244 | Added tLiteralString Logic, along with String Operations | | tLiteralString:  oEmitNone(iData)  oEmitString  oEmitNone(iText)  oOperandPushStringDescriptor  @EmitStringDescriptor  | tLiteralBoolean:  oOperandPushBoolean  | tAssignInteger:  @OperandAssignIntegerPopPop  >  % | tAssignChar:  % @OperandAssignCharPopPop  % >  | tAssignString:  @OperandAssignStringPopPop  >  | tAssignBoolean:  @OperandAssignBooleanPopPop  >  | tLength:  @OperandLength  | tSubstring:  @OperandSubstringPopPop  | tConcatenate:  @OperandConcatenatePop |
| coder.ssl | 2345-2367 | Implemented OperandAssignStringPopPop | OperandAssignStringPopPop:    @SaveTempRegsToStack  @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp  @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp  oOperandPushMode(mTrap)  oOperandSetValue(trAssignString)  oOperandSetLength(two)  oEmitSingle(iCall)  oOperandPop  oOperandPushMode(mStackReg)  oOperandSetLength(two)  oOperandPushMode(mManifest)  oOperandSetLength(two)  oOperandSetValue(eight)  oEmitDouble(iAdd)  oOperandPop  oOperandPop  @RestoreTempRegsFromStack; |
| coder.ssl | 2387 – 2420 | Implemented OperandConcatenatePop | OperandConcatenatePop:  @SaveTempRegsToStack    @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp  @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp    oOperandPushMode(mTrap)  oOperandSetValue(trConcatenate)  oOperandSetLength(word)  oEmitSingle(iCall)  oOperandPop  oOperandPushMode(mStackReg)  oOperandSetLength(word)  oOperandPushMode(mManifest)  oOperandSetLength(word)  oOperandSetValue(eight)  oEmitDouble(iAdd)  oOperandPop  oOperandPop  oOperandPushMode(mScratchReg1)  oOperandSetLength(word)  oOperandPushMode(mResultReg)  oOperandSetLength(word)  oEmitDouble(iMov)  oOperandPop    @RestoreTempRegsFromStack  @OperandForceIntoTemp  oOperandSetLength(word); |
| coder.ssl | 2435 – 2465 | Implemented OperandSubstringPopPop | OperandSubstringPopPop:  @SaveTempregsToStack  @OperandForceToStack  @OperandPopAndFreeTemp  @OperandForceToStack  @OperandPopAndFreeTemp  @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp  oOperandPushMode(mTrap)  oOperandSetValue(trSubstring)  oOperandSetLength(two)  oEmitSingle(iCall)  oOperandPop  oOperandPushMode(mStackReg)  oOperandSetLength(two)  oOperandPushMode(mManifest)  oOperandSetLength(two)  oOperandSetValue(twelve)  oEmitDouble(iAdd)  oOperandPop  oOperandPop  oOperandPushMode(mScratchReg1)  oOperandSetLength(two)  oOperandPushMode(mResultReg)  oOperandSetLength(two)  oEmitDouble(iMov)  oOperandPop  @RestoreTempRegsFromStack  @OperandForceIntoTemp  oOperandSetLength(two); |
| coder.ssl | 2966 – 2993 | Implemented OperandChr | OperandChr:  @SaveTempRegsToStack  @OperandForceToStack  @OperandPopAndFreeTemp  oOperandPushMode(mTrap)  oOperandSetValue(trChrString)  oOperandSetLength(two)  oEmitSingle(iCall)  oOperandPop  oOperandPushMode(mStackReg)  oOperandSetLength(two)  oOperandPushMode(mManifest)  oOperandSetLength(two)  oOperandSetValue(four)  oEmitDouble(iAdd)  oOperandPop  oOperandPop  oOperandPushMode(mScratchReg1)  oOperandSetLength(two)  oOperandPushMode(mResultReg)  oOperandSetLength(two)  oEmitDouble(iMov)  oOperandPop  @RestoreTempRegsFromStack  @OperandForceIntoTemp  oOperandSetLength(word); |
| coder.ssl | 3014-3017 | Edited OperandOrd | OperandOrd:  % Byte operand is on top of operand stack  @OperandForceAddressIntoTemp  oOperandSetMode(mTempIndirect)  oOperandSetLength(one)  oOperandPushMode(mManifest)  oOperandSetValue(zero)  oOperandSetLength(word)  @OperandForceIntoTemp % movl $0, %T  oOperandSetLength(byte)  oOperandSwap  oEmitDouble(iMov) % movb y, %T  @OperandPopAndFreeTemp  oOperandSetLength(word); |
| coder.ssl | 3184 | Implemented OperandLength | OperandLength:  @SaveTempRegsToStack  @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp  oOperandPushMode(mTrap)  oOperandSetValue(trLength)  oOperandSetLength(word)  oEmitSingle(iCall)  oOperandPop  oOperandPushMode(mStackReg)  oOperandSetLength(word)  oOperandPushMode(mManifest)  oOperandSetLength(word)  oOperandSetValue(four)  oEmitDouble(iAdd)  oOperandPop  oOperandPop  oOperandPushMode(mScratchReg1)  oOperandSetLength(word)  oOperandPushMode(mResultReg)  oOperandSetLength(word)  oEmitDouble(iMov)  oOperandPop  @RestoreTempRegsFromStack  @OperandForceIntoTemp  oOperandSetLength(word); |
| coder.ssl | 3365 - 3395 | Implemented OperandStringEqualPop | OperandStringEqualPop:  @SaveTempRegsToStack  @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp  @OperandForceAddressIntoTemp  @OperandForceToStack  @OperandPopAndFreeTemp  oOperandPushMode(mTrap)  oOperandSetValue(trStringEqual)  oOperandSetLength(word)  oEmitSingle(iCall)  oOperandPop  oOperandPushMode(mStackReg)  oOperandSetLength(word)  oOperandPushMode(mManifest)  oOperandSetLength(word)  oOperandSetValue(eight)  oEmitDouble(iAdd)  oOperandPop  oOperandPop  oOperandPushMode(mScratchReg1)  oOperandSetLength(word)  oOperandPushMode(mResultReg)  oOperandSetLength(word)  oEmitDouble(iMov)  oOperandPop  @RestoreTempRegsFromStack  @OperandForceIntoTemp  oOperandSetLength(byte); |
| coder.ssl | 3537 | Added mString logic in OperandForceIntoTemp | | mString:  @OperandForceAddressIntoTemp |
| coder.ssl | 3669 | Added logic for regCX | | regCX:  @OperandForceAddressIntoTemp |
| coder.pt | 58-441 | Updated token values |  |
| coder.pt | 975 | Changed logic to account for string size 0 | procedure EmitX86StringLiteral;  { Emit an x86 assembly code string constant }  var  i: 1 .. maxLiteralLength;  {i: 0 .. maxLiteralLength;} |
| coder.pt | 1574 | Changed Assert from tStringDescriptor to tLiteralString | Assert ( (compoundToken = tLiteralString), assert28); |
| coder.pt | 1876 – 1881 | Removed unused tokens from case and added new tokens to case | case compoundToken of  tLiteralAddress, tLiteralInteger,  tLiteralBoolean, {tStringDescriptor, tSkipString,}  tCallEnd, tLoopTest, tLoopEnd,  tLineNumber, tTrap, tIfThen, tIfMerge, tSkipProc,  tCaseSelect, tCaseMerge: |
| coder.pt | 1900 | Changed tLiteralString logic to account for string size 0 | tLiteralString:  begin  compoundTokenLength := compoundTokenValue;  Assert ( (compoundTokenLength >= 0), assert5);  compoundTokenValue := tCodeAddress;  i := 1;  { i := 0;} |
| coder.pt | 2224 | Removed AssertTempsAreAllFree |  |
| coder.pt | 2273 | Removed oOperandPushChar logic |  |